

2-Day Introduction to Agent-Based Modelling

Day 1: Session 2

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



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...

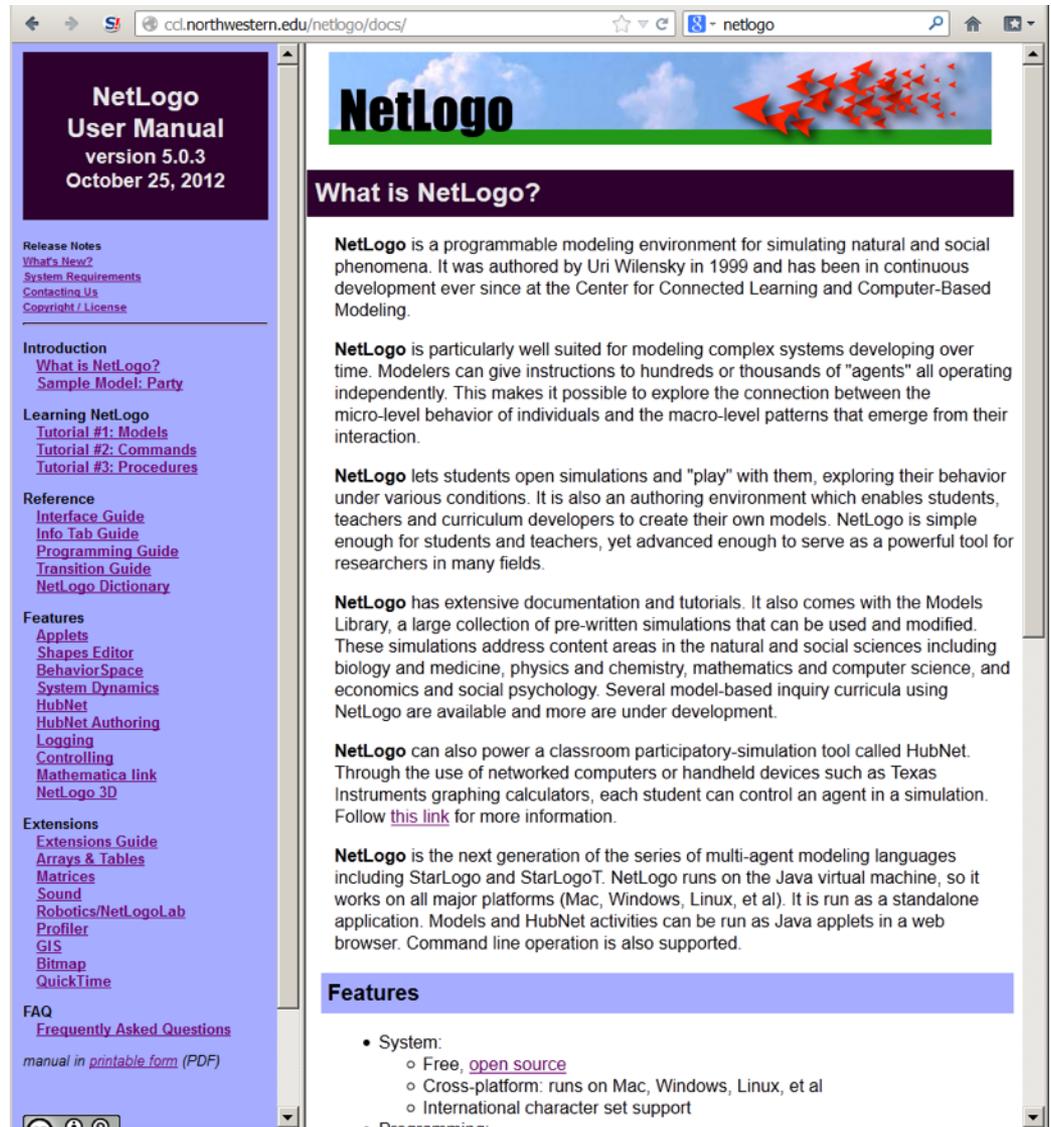
...like a language there will be several phases you will go through:

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



NetLogo
User Manual
version 5.0.3
October 25, 2012

[Release Notes](#)
[What's New?](#)
[System Requirements](#)
[Contacting Us](#)
[Copyright / License](#)

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Info Tab Guide](#)
[Programming Guide](#)
[Transition Guide](#)
[NetLogo Dictionary](#)

Features
[Applets](#)
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica link](#)
[NetLogo 3D](#)

Extensions
[Extensions Guide](#)
[Arrays & Tables](#)
[Matrices](#)
[Sound](#)
[Robotics/NetLogoLab](#)
[Profiler](#)
[GIS](#)
[Bitmap](#)
[QuickTime](#)

FAQ
[Frequently Asked Questions](#)
[manual in printable form \(PDF\)](#)

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and "play" with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo can also power a classroom participatory-simulation tool called HubNet. Through the use of networked computers or handheld devices such as Texas Instruments graphing calculators, each student can control an agent in a simulation. Follow [this link](#) for more information.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogoT. NetLogo runs on the Java virtual machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a standalone application. Models and HubNet activities can be run as Java applets in a web browser. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support

The Main page

A simple, walk-through tutorial

NetLogo User Manual
version 5.0.3
October 25, 2012

[Release Notes](#)
[What's New?](#)
[System Requirements](#)
[Contacting Us](#)
[Copyright / License](#)

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Info Tab Guide](#)
[Programming Guide](#)
[Transition Guide](#)
[NetLogo Dictionary](#)

Features
[Applets](#)
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica link](#)
[NetLogo 3D](#)

Extensions
[Extensions Guide](#)
[Arrays & Tables](#)
[Matrices](#)
[Sound](#)
[Robotics/NetLogoLab](#)
[Profiler](#)
[GIS](#)
[Bitmap](#)
[QuickTime](#)

FAQ
[Frequently Asked Questions](#)
[manual in printable form \(PDF\)](#)

NetLogo

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and "play" with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo can also power a classroom participatory-simulation tool called HubNet. Through the use of networked computers or handheld devices such as Texas Instruments graphing calculators, each student can control an agent in a simulation. Follow [this link](#) for more information.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogoT. NetLogo runs on the Java virtual machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a standalone application. Models and HubNet activities can be run as Java applets in a web browser. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support

The Main page

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

NetLogo User Manual
version 5.0.3
October 25, 2012

[Release Notes](#)
[What's New?](#)
[System Requirements](#)
[Contacting Us](#)
[Copyright / License](#)

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Info Tab Guide](#)
[Programming Guide](#)
[Transition Guide](#)
[NetLogo Dictionary](#)

Features
[Applets](#)
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica link](#)
[NetLogo 3D](#)

Extensions
[Extensions Guide](#)
[Arrays & Tables](#)
[Matrices](#)
[Sound](#)
[Robotics/NetLogoLab](#)
[Profiler](#)
[GIS](#)
[Bitmap](#)
[QuickTime](#)

FAQ
[Frequently Asked Questions](#)
[manual in printable form \(PDF\)](#)

NetLogo

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of “agents” all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and “play” with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo can also power a classroom participatory-simulation tool called HubNet. Through the use of networked computers or handheld devices such as Texas Instruments graphing calculators, each student can control an agent in a simulation. Follow [this link](#) for more information.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogoT. NetLogo runs on the Java virtual machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a standalone application. Models and HubNet activities can be run as Java applets in a web browser. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support

The Main page

NetLogo User Manual
version 5.0.3
October 25, 2012

[Release Notes](#)
[What's New?](#)
[System Requirements](#)
[Contacting Us](#)
[Copyright / License](#)

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Info Tab Guide](#)
[Programming Guide](#)
[Transition Guide](#)
[Net.ogo Dictionary](#)

Features
[Applets](#)
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica link](#)
[Net.ogo 3D](#)

Extensions
[Extensions Guide](#)
[Arrays & Tables](#)
[Matrices](#)
[Sound](#)
[Robotics/Net.ogoLab](#)
[Profiler](#)
[GIS](#)
[Bitmap](#)
[QuickTime](#)

FAQ
[Frequently Asked Questions](#)
[manual in printable form \(PDF\)](#)

NetLogo

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and "play" with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo can also power a classroom participatory-simulation tool called HubNet. Through the use of networked computers or handheld devices such as Texas Instruments graphing calculators, each student can control an agent in a simulation. Follow [this link](#) for more information.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogoT. NetLogo runs on the Java virtual machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a standalone application. Models and HubNet activities can be run as Java applets in a web browser. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support

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*

NetLogo User Manual
version 5.0.3
October 25, 2012

Release Notes
[What's New?](#)
[System Requirements](#)
[Contacting Us](#)
[Copyright / License](#)

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Info Tab Guide](#)
[Programming Guide](#)
[Troubleshooting Guide](#)
[Netlogo Dictionary](#)

Features
[Applets](#)
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica link](#)
[NetLogo 3D](#)

Extensions
[Extensions Guide](#)
[Arrays & Tables](#)
[Matrices](#)
[Sound](#)
[Robotics/NetlogoLab](#)
[Profiler](#)
[GIS](#)
[Bitmap](#)
[QuickTime](#)

FAQ
[Frequently Asked Questions](#)
[manual in printable form \(PDF\)](#)

NetLogo

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and "play" with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo can also power a classroom participatory-simulation tool called HubNet. Through the use of networked computers or handheld devices such as Texas Instruments graphing calculators, each student can control an agent in a simulation. Follow [this link](#) for more information.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogoT. NetLogo runs on the Java virtual machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a standalone application. Models and HubNet activities can be run as Java applets in a web browser. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support



The NetLogo Dictionary

NetLogo 5.0.1 User Manual: Net... +

NetLogo Dictionary

NetLogo 5.0.1 User Manual

Alphabetical: [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [?](#)

Categories: [Turtle](#) - [Patch](#) - [Agentset](#) - [Color](#) - [Task](#) - [Control/Logic](#) - [World](#) - [Perspective](#)
[Input/Output](#) - [File](#) - [List](#) - [String](#) - [Math](#) - [Plotting](#) - [Links](#) - [Movie](#) - [System](#) - [HubNet](#)

Special: [Variables](#) - [Keywords](#) - [Constants](#)

Categories

This is an approximate grouping. Remember that a turtle-related primitive might still be used by patches or the observer, and vice versa. To see which agents (turtles, patches, links, observer) can actually run a primitive, consult its dictionary entry.

Turtle-related

[back](#) (bk) [<breeds>-at](#) [<breeds>-here](#) [<breeds>-on](#) [can-move?](#) [clear-turtles](#) (ct)
[create-<breeds>](#) [create-ordered-<breeds>](#) [create-ordered-turtles](#) (cro) [create-turtles](#) (crt)
[die](#) [distance](#) [distancexy](#) [downhill](#) [downhill4](#) [dx](#) [dy](#) [face](#) [facexy](#) [forward](#) (fd) [hatch](#)
[hatch-<breeds>](#) [hide-turtle](#) (ht) [home](#) [inspect](#) [is-<breed>?](#) [is-turtle?](#) [jump](#) [layout-circle](#) [left](#) (lt)
[move-to](#) [myself](#) [nobody](#) [no-turtles](#) [of](#) [other](#) [patch-ahead](#) [patch-at](#) [patch-at-heading-](#)
[and-distance](#) [patch-here](#) [patch-left-and-ahead](#) [patch-right-and-ahead](#) [pen-down](#) (pd)
[pen-erase](#) (pe) [pen-up](#) (pu) [random-xcor](#) [random-ycor](#) [right](#) (rt) [self](#) [set-default-shape](#) [set-](#)
[line-thickness](#) [setxy](#) [shapes](#) [show-turtle](#) (st) [sprout](#) [sprout-<breeds>](#) [stamp](#) [stamp-erase](#)
[subject](#) [subtract-headings](#) [tie](#) [towards](#) [towardsxy](#) [turtle](#) [turtle-set](#) [turtles](#) [turtles-at](#) [turtles-here](#)
[turtles-on](#) [turtles-own](#) [untie](#) [uphill](#) [uphill4](#)

Patch-related

The NetLogo Dictionary

Alphabetic Index to Primitives

The screenshot shows the NetLogo Dictionary interface. At the top, there is a title bar with the text "NetLogo 5.0.1 User Manual: Net..." and a "+" button. Below this is a dark purple header with the text "NetLogo Dictionary" and "NetLogo 5.0.1 User Manual" in the top right corner. A green box on the left contains the text "Alphabetic Index to Primitives" with a green arrow pointing to the "Alphabetical:" link in the dictionary. The dictionary content is as follows:

Alphabetical: [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [?](#)

Categories: [Turtle](#) - [Patch](#) - [Agentset](#) - [Color](#) - [Task](#) - [Control/Logic](#) - [World](#) - [Perspective](#)
[Input/Output](#) - [File](#) - [List](#) - [String](#) - [Math](#) - [Plotting](#) - [Links](#) - [Movie](#) - [System](#) - [HubNet](#)

Special: [Variables](#) - [Keywords](#) - [Constants](#)

Categories

This is an approximate grouping. Remember that a turtle-related primitive might still be used by patches or the observer, and vice versa. To see which agents (turtles, patches, links, observer) can actually run a primitive, consult its dictionary entry.

Turtle-related

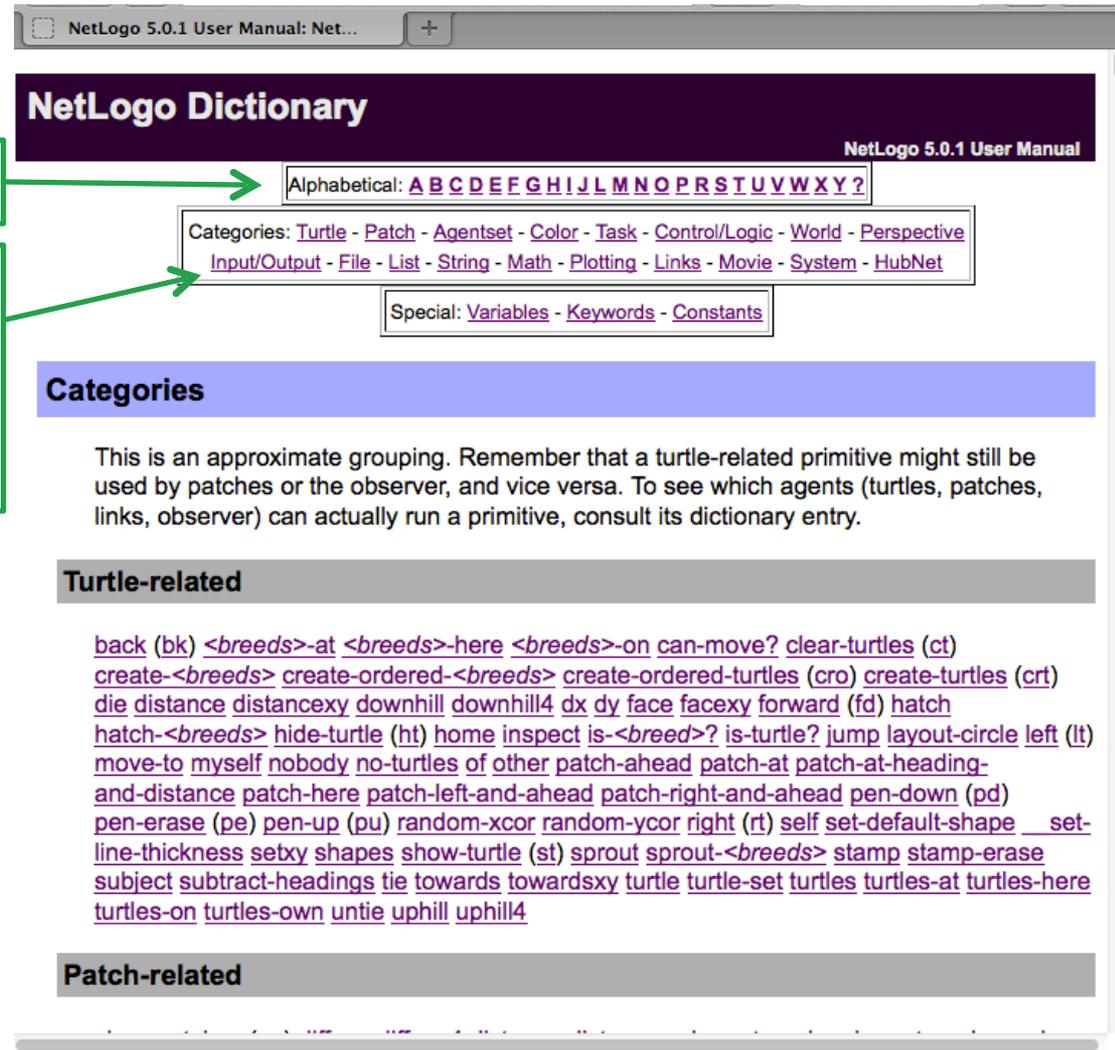
[back](#) (bk) [<breeds>-at](#) [<breeds>-here](#) [<breeds>-on](#) [can-move?](#) [clear-turtles](#) (ct)
[create-<breeds>](#) [create-ordered-<breeds>](#) [create-ordered-turtles](#) (cro) [create-turtles](#) (crt)
[die](#) [distance](#) [distancexy](#) [downhill](#) [downhill4](#) [dx](#) [dy](#) [face](#) [facexy](#) [forward](#) (fd) [hatch](#)
[hatch-<breeds>](#) [hide-turtle](#) (ht) [home](#) [inspect](#) [is-<breed>?](#) [is-turtle?](#) [jump](#) [layout-circle](#) [left](#) (lt)
[move-to](#) [myself](#) [nobody](#) [no-turtles](#) [of](#) [other](#) [patch-ahead](#) [patch-at](#) [patch-at-heading-](#)
[and-distance](#) [patch-here](#) [patch-left-and-ahead](#) [patch-right-and-ahead](#) [pen-down](#) (pd)
[pen-erase](#) (pe) [pen-up](#) (pu) [random-xcor](#) [random-ycor](#) [right](#) (rt) [self](#) [set-default-shape](#) [set-](#)
[line-thickness](#) [setxy](#) [shapes](#) [show-turtle](#) (st) [sprout](#) [sprout-<breeds>](#) [stamp](#) [stamp-erase](#)
[subject](#) [subtract-headings](#) [tie](#) [towards](#) [towardsxy](#) [turtle](#) [turtle-set](#) [turtles](#) [turtles-at](#) [turtles-here](#)
[turtles-on](#) [turtles-own](#) [untie](#) [uphill](#) [uphill4](#)

Patch-related

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 screenshot shows the NetLogo Dictionary interface. At the top, there is a title bar with the text "NetLogo 5.0.1 User Manual: Net...". Below this is a dark purple header with the text "NetLogo Dictionary" and "NetLogo 5.0.1 User Manual" in the top right corner. The main content area is white and contains several sections:

- Alphabetical:** A row of letters from A to Z, with a question mark at the end, all enclosed in a box. A green arrow points from the text "Alphabetic Index to Primitives" to this box.
- Categories:** A list of categories: Turtle - Patch - Agentset - Color - Task - Control/Logic - World - Perspective - Input/Output - File - List - String - Math - Plotting - Links - Movie - System - HubNet. A green arrow points from the text "Primitives by functional category" to this list.
- Special:** A list of special items: Variables - Keywords - Constants, enclosed in a box.
- Categories:** A section header with a light blue background.
- Turtle-related:** A section header with a light grey background, followed by a list of primitive names and their arguments, such as `back (bk)`, `<breeds>-at <breeds>-here <breeds>-on`, `can-move?`, `clear-turtles (ct)`, `create-<breeds>`, `create-ordered-<breeds>`, `create-ordered-turtles (cro)`, `create-turtles (crt)`, `die`, `distance`, `distancexy`, `downhill`, `downhill4`, `dx`, `dy`, `face`, `facexy`, `forward (fd)`, `hatch`, `hatch-<breeds>`, `hide-turtle (ht)`, `home`, `inspect`, `is-<breed>?`, `is-turtle?`, `jump`, `layout-circle`, `left (lt)`, `move-to`, `myself`, `nobody`, `no-turtles`, `of`, `other`, `patch-ahead`, `patch-at`, `patch-at-heading-and-distance`, `patch-here`, `patch-left-and-ahead`, `patch-right-and-ahead`, `pen-down (pd)`, `pen-erase (pe)`, `pen-up (pu)`, `random-xcor`, `random-ycor`, `right (rt)`, `self`, `set-default-shape`, `set-line-thickness`, `setxy`, `shapes`, `show-turtle (st)`, `sprout`, `sprout-<breeds>`, `stamp`, `stamp-erase`, `subject`, `subtract-headings`, `tie`, `towards`, `towardsxy`, `turtle`, `turtle-set`, `turtles`, `turtles-at`, `turtles-here`, `turtles-on`, `turtles-own`, `untie`, `uphill`, `uphill4`.
- Patch-related:** A section header with a light grey background.

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 screenshot shows the NetLogo Dictionary interface. At the top, there is a title bar with the text "NetLogo 5.0.1 User Manual: Net...". Below this is a dark purple header with the text "NetLogo Dictionary" and "NetLogo 5.0.1 User Manual" on the right. The main content area is white and contains several sections:

- Alphabetical:** A row of letters from A to Z, with a question mark at the end, enclosed in a box. A green arrow points from the "Alphabetic Index to Primitives" text box to this section.
- Categories:** A list of categories: Turtle - Patch - Agentset - Color - Task - Control/Logic - World - Perspective - Input/Output - File - List - String - Math - Plotting - Links - Movie - System - HubNet. A green arrow points from the "Primitives by functional category" text box to this section.
- Special:** A list of special items: Variables - Keywords - Constants, enclosed in a box.
- Categories:** A section with a light blue background. Below the header, there is a paragraph: "This is an approximate grouping. Remember that a turtle-related primitive might still be used by patches or the observer, and vice versa. To see which agents (turtles, patches, links, observer) can actually run a primitive, consult its dictionary entry."
- Turtle-related:** A section with a grey background. Below the header, there is a list of primitives: [back \(bk\)](#) [<breeds>-at <breeds>-here <breeds>-on](#) [can-move?](#) [clear-turtles \(ct\)](#) [create-<breeds>](#) [create-ordered-<breeds>](#) [create-ordered-turtles \(cro\)](#) [create-turtles \(crt\)](#) [die](#) [distance](#) [distancexy](#) [downhill](#) [downhill4](#) [dx](#) [dy](#) [face](#) [facexy](#) [forward \(fd\)](#) [hatch](#) [hatch-<breeds>](#) [hide-turtle \(ht\)](#) [home](#) [inspect](#) [is-<breed>?](#) [is-turtle?](#) [jump](#) [layout-circle](#) [left \(lt\)](#) [move-to](#) [myself](#) [nobody](#) [no-turtles](#) [of](#) [other](#) [patch-ahead](#) [patch-at](#) [patch-at-heading-and-distance](#) [patch-here](#) [patch-left-and-ahead](#) [patch-right-and-ahead](#) [pen-down \(pd\)](#) [pen-erase \(pe\)](#) [pen-up \(pu\)](#) [random-xcor](#) [random-ycor](#) [right \(rt\)](#) [self](#) [set-default-shape](#) [set-line-thickness](#) [setxy](#) [shapes](#) [show-turtle \(st\)](#) [sprout](#) [sprout-<breeds>](#) [stamp](#) [stamp-erase](#) [subject](#) [subtract-headings](#) [tie](#) [towards](#) [towardsxy](#) [turtle](#) [turtle-set](#) [turtles](#) [turtles-at](#) [turtles-here](#) [turtles-on](#) [turtles-own](#) [untie](#) [uphill](#) [uphill4](#)
- Patch-related:** A section with a grey background.

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**”...

The screenshot shows the NetLogo Dictionary window. At the top, there's a title bar "NetLogo 5.0.1 User Manual: Net...". Below it, the main title "NetLogo Dictionary" is displayed. To the right of the title, it says "NetLogo 5.0.1 User Manual".

Below the title, there are three navigation boxes:

- Alphabetical: [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#) [?](#)
- Categories: [Turtle](#) - [Patch](#) - [Agentset](#) - [Color](#) - [Task](#) - [Control/Logic](#) - [World](#) - [Perspective](#) - [Input/Output](#) - [File](#) - [List](#) - [String](#) - [Math](#) - [Plotting](#) - [Links](#) - [Movie](#) - [System](#) - [HubNet](#)
- Special: [Variables](#) - [Keywords](#) - [Constants](#)

Below these boxes is a section titled "Categories" with a light blue background. The text below it reads: "This is an approximate grouping. Remember that a turtle-related primitive might still be used by patches or the observer, and vice versa. To see which agents (turtles, patches, links, observer) can actually run a primitive, consult its dictionary entry."

Underneath is a section titled "Turtle-related" with a grey background. It contains a list of primitive names with their arguments in angle brackets, such as [back \(bk\)](#), [clear-turtles \(ct\)](#), [create-ordered-turtles \(cro\)](#), [create-turtles \(crt\)](#), [die](#), [distance](#), [distancexy](#), [downhill](#), [downhill4](#), [dx](#), [dy](#), [face](#), [facexy](#), [forward \(fd\)](#), [hatch](#), [hatch-<breeds>](#), [hide-turtle \(ht\)](#), [home](#), [inspect](#), [is-<breed>?](#), [is-turtle?](#), [jump](#), [layout-circle](#), [left \(lt\)](#), [move-to](#), [myself](#), [nobody](#), [no-turtles](#), [of](#), [other](#), [patch-ahead](#), [patch-at](#), [patch-at-heading-and-distance](#), [patch-here](#), [patch-left-and-ahead](#), [patch-right-and-ahead](#), [pen-down \(pd\)](#), [pen-erase \(pe\)](#), [pen-up \(pu\)](#), [random-xcor](#), [random-ycor](#), [right \(rt\)](#), [self](#), [set-default-shape](#), [set-line-thickness](#), [setxy](#), [shapes](#), [show-turtle \(st\)](#), [sprout](#), [sprout-<breeds>](#), [stamp](#), [stamp-erase](#), [subject](#), [subtract-headings](#), [tie](#), [towards](#), [towardsxy](#), [turtle](#), [turtle-set](#), [turtles](#), [turtles-at](#), [turtles-here](#), [turtles-on](#), [turtles-own](#), [untie](#), [uphill](#), [uphill4](#).

Below the "Turtle-related" section is a section titled "Patch-related" with a grey background.

Green arrows point from the "Alphabetic Index to Primitives" box to the "Alphabetical" box, and from the "Primitives by functional category" box to the "Categories" box. A blue bracket points from the "Each category has a list of primitives to click on" box to the list of primitives in the "Turtle-related" section. A blue box points from the "Click on 'Control/Logic' then 'ask'..." box to the "Control/Logic" link in the "Categories" box.

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

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

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!*

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

ask

ask agentset [commands]
ask agent [commands]

A brief explanation of the primitive

The specified agent or agentset runs the given commands.

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

```
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

Notes – these explain potential “gotchas” and common mistakes

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

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-concur

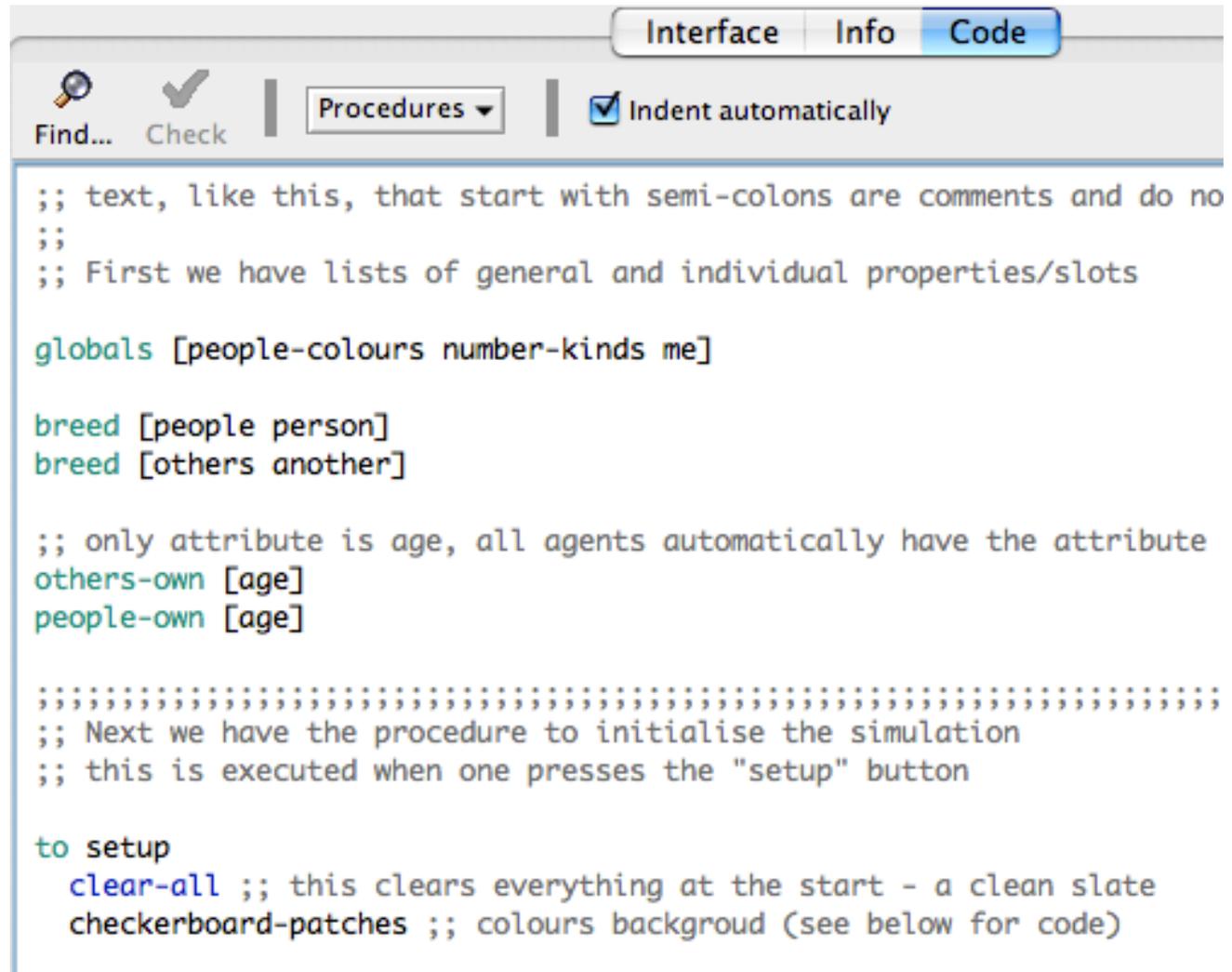
Try looking up the primitives:
“to”, “set”, and “if”

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

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



The screenshot shows a code editor window with tabs for 'Interface', 'Info', and 'Code'. The 'Code' tab is active. The editor has a toolbar with 'Find...', 'Check', a dropdown menu set to 'Procedures', and a checked checkbox for 'Indent automatically'. The code in the editor is as follows:

```
;; 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]

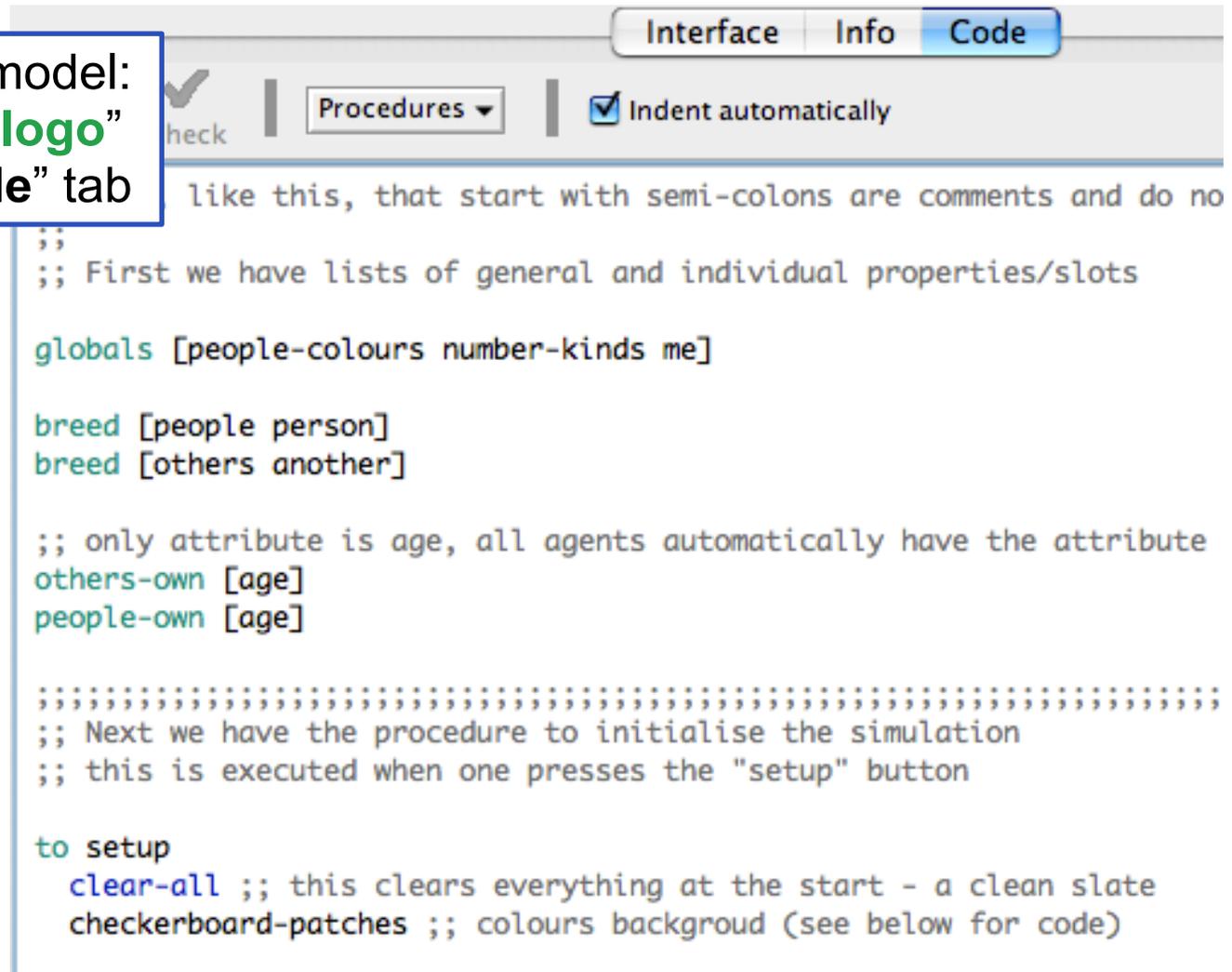
;; only attribute is age, all agents automatically have the attribute
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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 background (see below for code)
```

Other Declarations

Load the NetLogo model:
“2-friends-begin.nlogo”
and select the “Code” tab



```
Interface | Info | Code
check | Procedures | Indent automatically
;;
;; First we have lists of general and individual properties/slots
globals [people-colours number-kinds me]

breed [people person]
breed [others another]

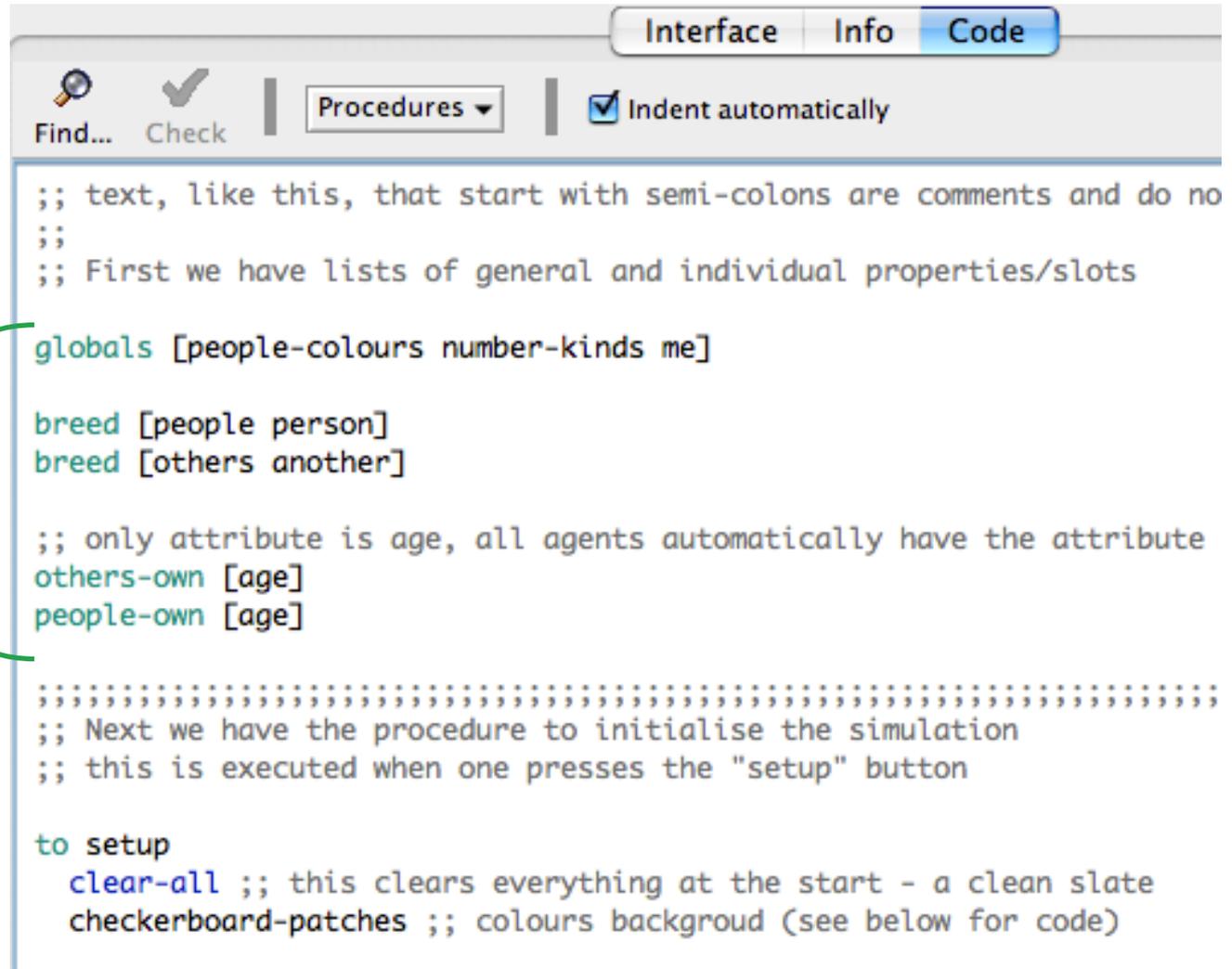
;; only attribute is age, all agents automatically have the attribute
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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 background (see below for code)
```

Other Declarations

These are the various declarations



The screenshot shows a code editor window with tabs for 'Interface', 'Info', and 'Code'. The 'Code' tab is active. The editor contains the following code:

```
;; 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 background (see below for code)
```

Other 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 background (see below for code)
```

These are the various declarations

These just comments to help you understand the code



Other 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 background (see below for code)
```

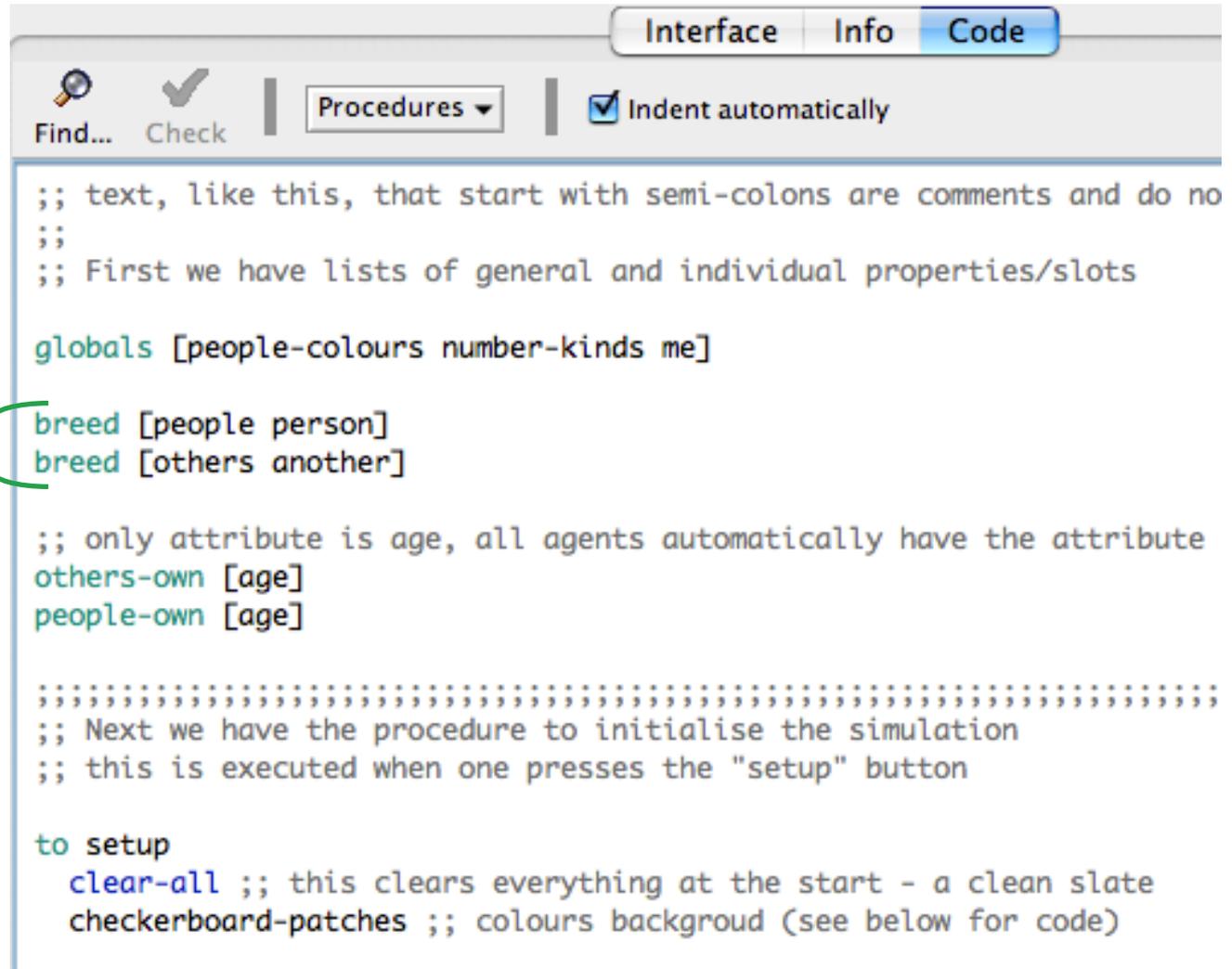
These are the various declarations

These just comments to help you understand the code

The code – the procedure definitions are here onwards

Other Declarations

Two kinds of agent are defined: “people” and “others”



```
;; 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 background (see below for code)
```

Other Declarations

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

```
Interface Info Code
Find... Check | Procedures | [x] 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]

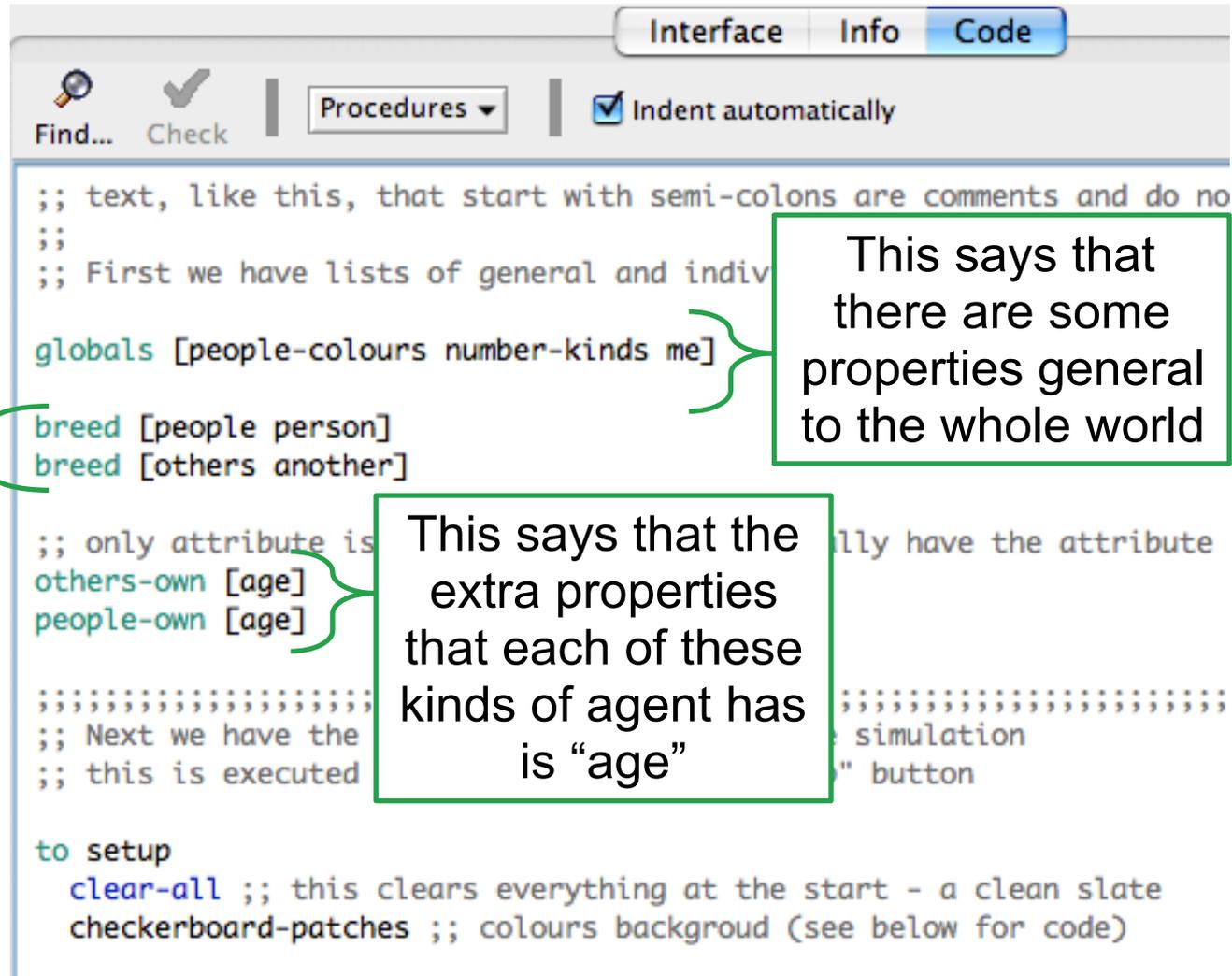
;; only attribute is
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Next we have the
;; this is executed
;;;;;;;;;;;;;;;;;;;;;;;;;;;;; simulation
" button

to setup
  clear-all ;; this clears everything at the start - a clean slate
  checkerboard-patches ;; colours background (see below for code)
```

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

Other Declarations



The screenshot shows a code editor window with tabs for 'Interface', 'Info', and 'Code'. The 'Code' tab is active. The code is as follows:

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

globals [people-colours number-kinds me]

breed [people person]
breed [others another]

;; only attribute is
others-own [age]
people-own [age]

;;;;;;;;;;;;;
;; Next we have the
;; this is executed

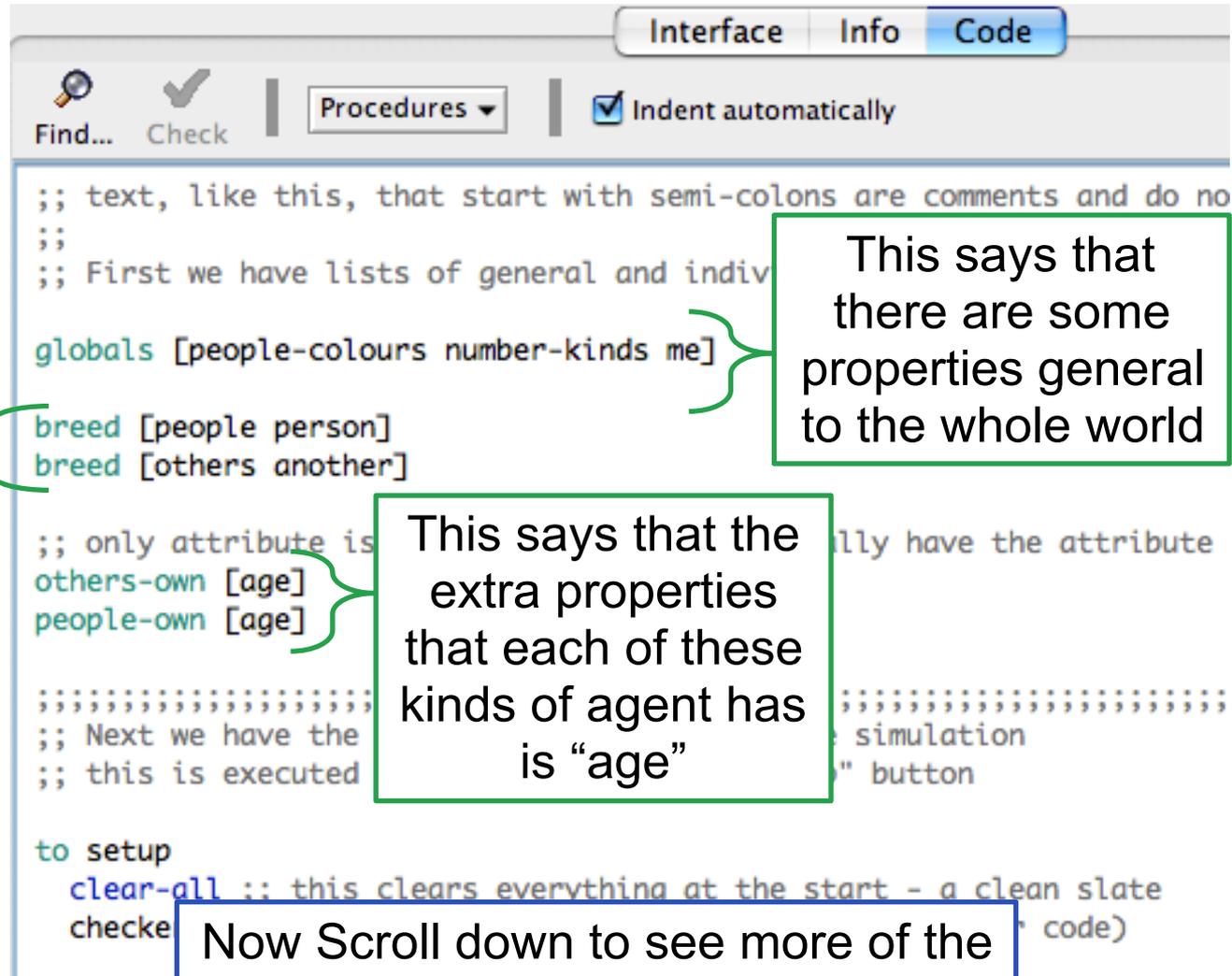
to setup
  clear-all ;; this clears everything at the start - a clean slate
  checkerboard-patches ;; colours background (see below for code)
```

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



The screenshot shows a code editor window with tabs for 'Interface', 'Info', and 'Code'. The 'Code' tab is active. The code is as follows:

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

globals [people-colours number-kinds me]

breed [people person]
breed [others another]

;; only attribute is
others-own [age]
people-own [age]

;;;;;;;;;;;;;
;; Next we have the
;; this is executed

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

Annotations in the image:

- A green box on the left points to the `breed` lines, containing the text: "Two kinds of agent are defined: 'people' and 'others'".
- A green box on the right points to the `globals` line, containing the text: "This says that there are some properties general to the whole world".
- A green box in the middle points to the `others-own` and `people-own` lines, containing the text: "This says that the extra properties that each of these kinds of agent has is 'age'".
- A blue box at the bottom points to the `clear-all` line, containing the text: "Now Scroll down to see more of the program code".

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 background (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 background (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 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 background (see below

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange yie
  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
end

;; this initialises the simulation time sys
```

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 background (see below for code)

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

  ;; 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 persons
  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
end

;; this initialises the simulation time system
```

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.

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 background (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
end
;; this initialises the simulation time sys

```

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
```

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.

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
    ;;if with-probab
  ]
  ;; 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
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
end
;; this progresses the tick cou
```

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
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
end

;; this progresses the tick counter
```

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 be  
;; 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
```

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 be
;; 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
```

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 be  
;; 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  
  ;; returns value TRUE with probability determined by 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.

The screenshot shows the NetLogo 'friends' simulation interface. At the top, there are tabs for 'Interface', 'Info', and 'Code'. Below the tabs is a toolbar with 'Edit', 'Delete', and 'Add' buttons, a dropdown menu showing 'abc Button', a speed slider set to 'normal speed', a 'view updates' checkbox which is checked, and a 'continuous' button. A 'Settings...' button is also present. The main area contains three buttons: 'setup', 'step', and 'go'. Below these is a 'population' slider set to 30. The simulation area is a 2x2 grid of black squares, each containing a different group of colorful circles and one orange stick figure. The top-left square has a white, yellow, cyan, and green circle. The top-right square has an orange stick figure and a blue circle. The bottom-left square has a white, purple, pink, and brown circle. The bottom-right square has a cyan, red, white, and yellow circle. The top of the simulation area shows 'ticks: 0' and a '3D' button. At the bottom, there is a 'Command Center' with a 'Clear' button and a text input field containing 'observer>'. The 'Interface' tab is selected.

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.

The screenshot shows the NetLogo 'friends' simulation interface. At the top, there are tabs for 'Interface', 'Info', and 'Code'. Below the tabs is a toolbar with 'Edit', 'Delete', and 'Add' buttons, a dropdown menu showing 'abc Button', a speed slider set to 'normal speed', a 'view updates' checkbox which is checked, and a 'continuous' button. The main area contains three buttons: 'setup', 'step', and 'go'. Below these buttons is a 'population' slider set to 30. The simulation area is divided into four quadrants, each showing a different arrangement of colored circles (agents) and one orange stick figure agent. The top-right quadrant shows the stick figure with a blue circle. The bottom-right quadrant shows a cluster of colored circles. The bottom-left quadrant shows a cluster of colored circles. The top-left quadrant shows a cluster of colored circles. The 'Command Center' at the bottom shows 'observer>' and a 'Clear' 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.

The screenshot shows the NetLogo 'friends' simulation interface. The top toolbar includes tabs for 'Interface', 'Info', and 'Code'. Below the tabs are buttons for 'Edit', 'Delete', 'Add', and a dropdown menu showing 'abc Button'. A speed slider is set to 'normal speed', and there are checkboxes for 'view updates' (checked) and 'continuous' (unchecked). A 'Settings...' button is on the right. The main area contains a 'setup' button, a 'step' button, and a 'go' button. Below these is a 'population' slider set to 30. The simulation area is a 2x2 grid of black squares, each containing a different arrangement of colorful circles and one orange stick figure. The top right corner of the simulation area shows 'ticks: 0' and a '3D' button. At the bottom, there is a 'Command Center' with a 'Clear' button and a text input field containing 'observer>'.

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

The screenshot shows the NetLogo 'friends' simulation interface. At the top, there are tabs for 'Interface', 'Info', and 'Code'. Below the tabs is a toolbar with 'Edit', 'Delete', 'Add', and a 'Button' dropdown menu. A speed slider is set to 'normal speed', and there are checkboxes for 'view updates' (checked) and 'continuous' (unchecked). The main area is divided into four quadrants showing a simulation of agents (represented by colored circles and a stick figure) on a black background. A 'population' slider is set to 30. The 'Command Center' at the bottom shows 'observer>'.

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.

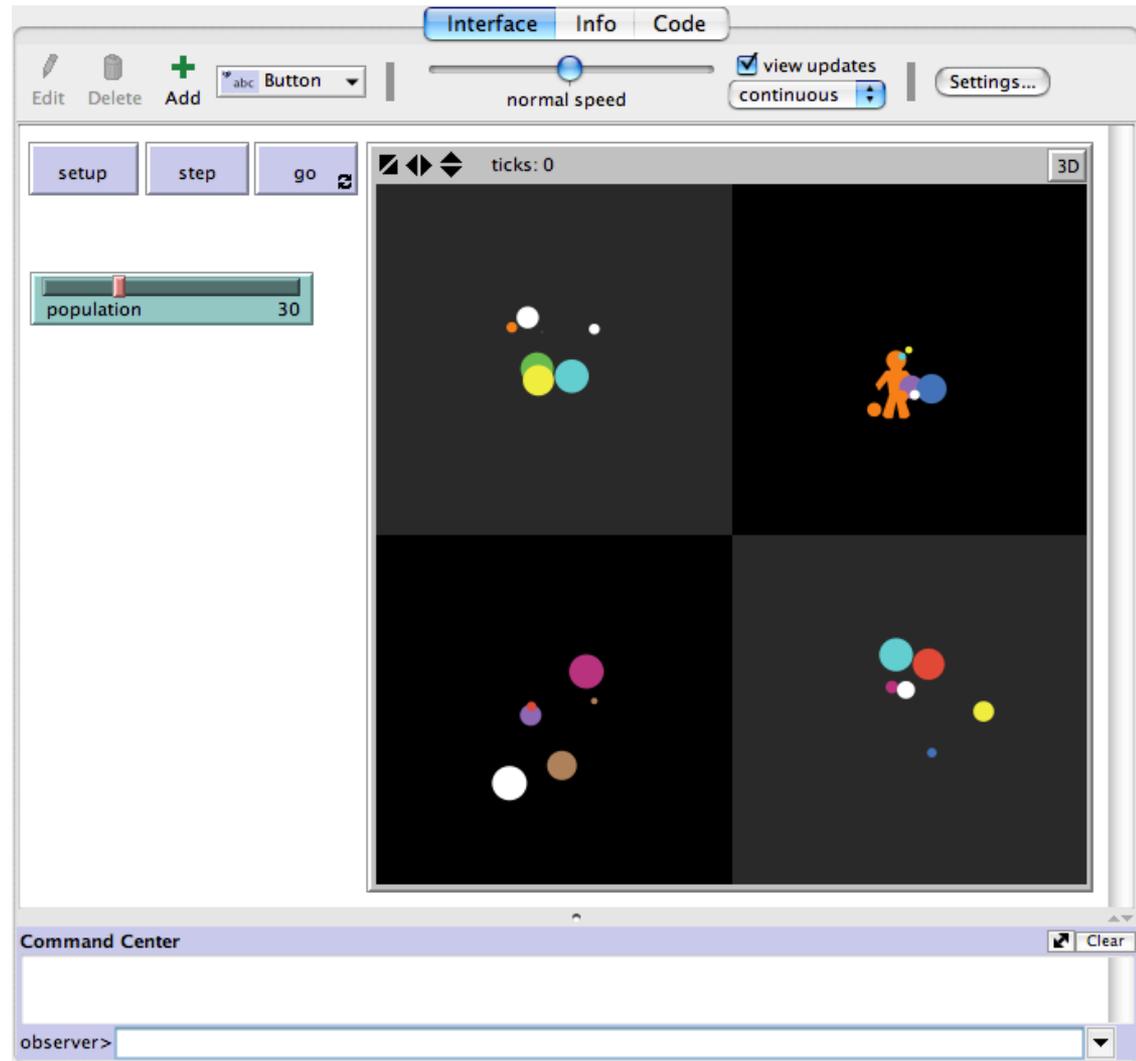
Change the population slider and then press “setup” again.

Try pressing the “step” button

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

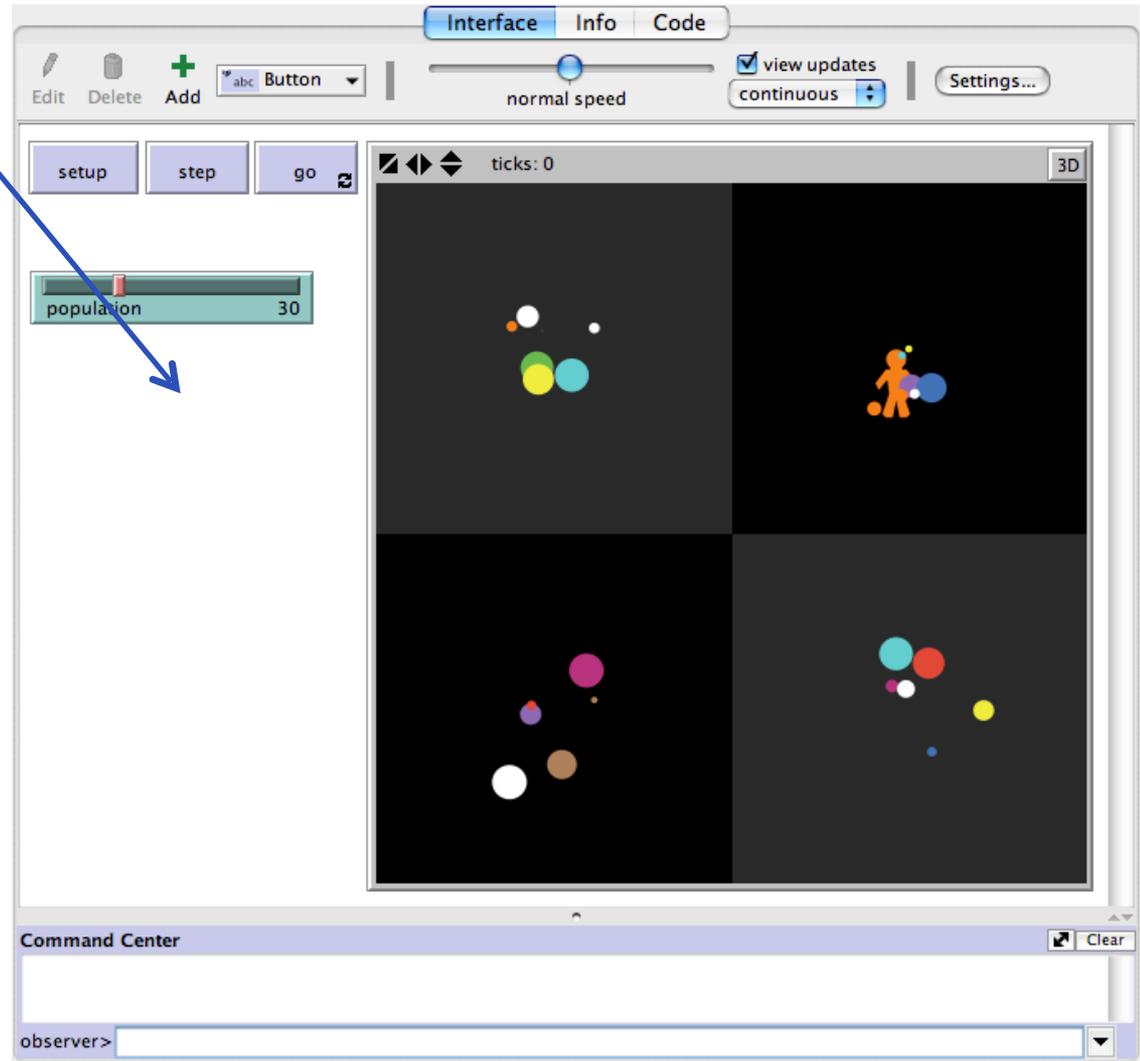
The screenshot shows the NetLogo 'friends' simulation interface. The top menu has 'Interface', 'Info', and 'Code' tabs. Below the menu are buttons for 'Edit', 'Delete', 'Add', and a dropdown menu showing 'abc Button'. A speed slider is set to 'normal speed', and there are checkboxes for 'view updates' (checked) and 'continuous' (unchecked). A 'Settings...' button is on the right. The main area contains a 'setup' button, a 'step' button, and a 'go' button. Below these is a 'population' slider set to 30. The simulation area is a 2x2 grid of black panels with colorful circles and a stick figure. The top right panel shows a stick figure with a blue circle. The bottom right panel shows several colorful circles. The bottom left panel shows a few colorful circles. The bottom right panel shows a 3D view of the simulation. At the bottom, there is a 'Command Center' with a 'Clear' button and a text input field containing 'observer>'.

Adding a slider



Adding a slider

Right-click (Mac: ctrl+click) on some empty space and select "Slider"



Adding a slider

The screenshot displays the NetLogo software interface. At the top, there are tabs for 'Interface', 'Info', and 'Code'. Below these are icons for 'Edit', 'Delete', and 'Add', along with a dropdown menu showing 'abc Button'. A slider control is visible, labeled 'normal speed', with a blue knob and a 'view updates' checkbox checked. A 'Settings...' button is also present.

In the main workspace, there are three buttons: 'setup', 'step', and 'go'. Below them is a slider labeled 'population' with a value of 30. To the right is a black canvas with several white and orange dots, and a 'ticks: 0' counter.

A 'Slider' dialog box is open in the foreground. It has a 'Global variable' field. Below it are three input fields: 'Minimum' (0), 'Increment' (1), and 'Maximum' (100). A note below these fields reads: 'min, increment, and max may be numbers or reporters'. There is also a 'Value' field (50) and a 'Units (optional)' field. A checkbox labeled 'vertical?' is unchecked. At the bottom right of the dialog are 'Cancel', 'Apply', and 'OK' buttons.

At the bottom of the NetLogo window is a 'Command Center' with a 'Clear' button and a text input field containing 'observer>'.

Adding a slider

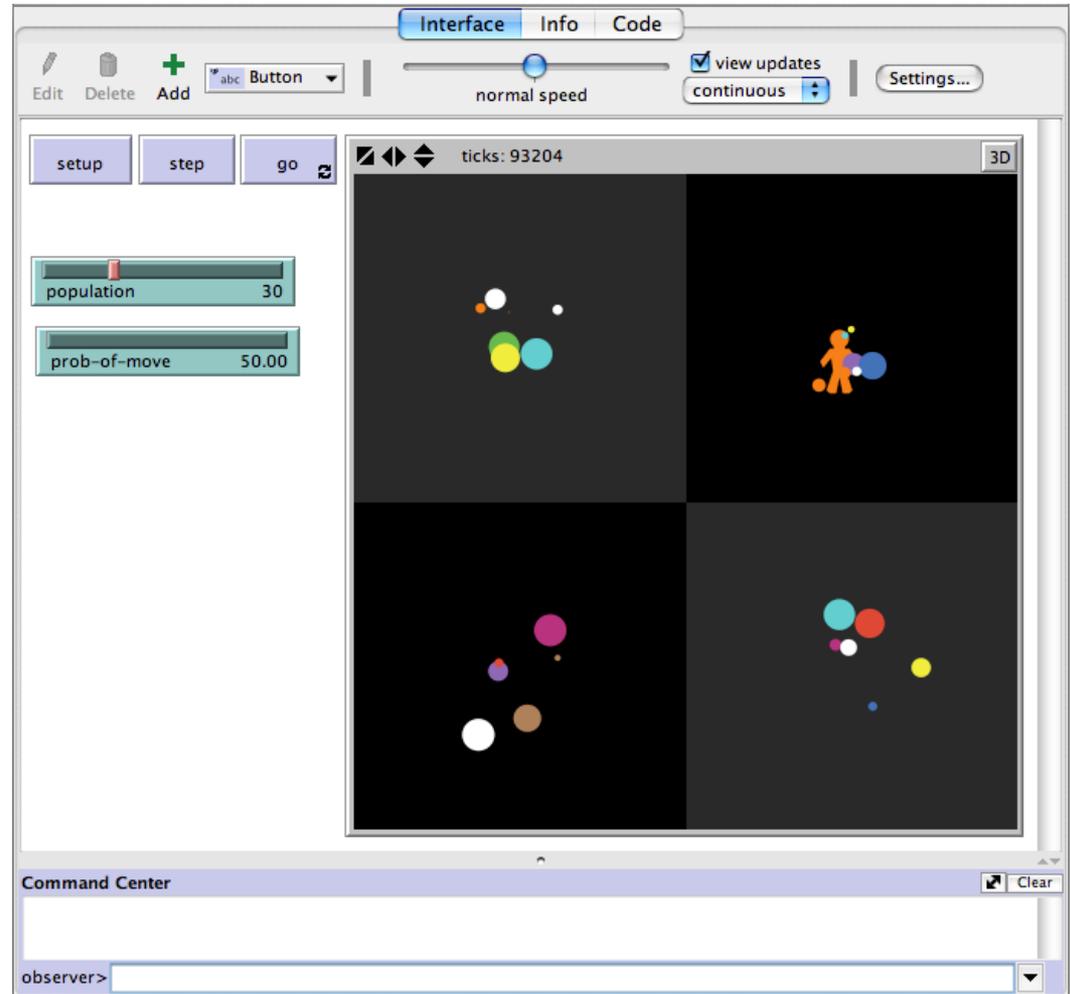
In the dialogue that appears... type “prob-of-move” in the “Global Variable” space, “0.01 in the “Increment” space and “1” in the “Maximum” space, then click “OK”.

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 semicolons 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: ctrl+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>

