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

Reference Index
What's New?
System Requirements
Contact Us
Version History

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 the 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, etc.). 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, etc.
- International character set support
The Main page

A simple, walk-through tutorial
General introductions to features – good to browse, especially look at the “Programming Guide” to understand how NetLogo does things.
Advanced stuff, only read if you have got the basics and need bits from here
But **THIS** is what you will keep referring to... the dictionary of all the Netlogo commands. *Please click on this*
The NetLogo Dictionary

NetLogo Dictionary

Alphabetical: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Categories: Turtle - Patch - Agentset - Color - Task - Control/Logic - World - Perspective
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 (ct)
die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch
hatch <breeds> hide-turtle (ht) home inspect is <breeds> is-turtle? jump layout-circle left (lt)
motion-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-at turtles-here
turtles-on turtles-own until uphill uphill4

Patch-related
The NetLogo Dictionary

Alphabetic Index to Primitives

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 (ct) die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch home inspect is-<breeds>? 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 towardsxxy turtle turtle-set turtles-at turtles-here turtles-on turtles-own until 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 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”

```
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

```
ask
ask agentset [commands]
ask agent [commands]
```

A brief explanation of the primitive

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
```

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

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

```
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 agent [commands]
ask agentset [commands]
```

**A brief explanation of the primitive**
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
```

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

**Notes** – these explain potential “gotchas” and common mistakes

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.

**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

```plaintext
;; 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

```nlogo
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]
person-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

```plaintext
;; 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

These are the various declarations

These just comments to help you understand the code

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

These just comments to help you understand the code

The code – the procedure definitions are here onwards

```lisp
;; 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)
```
Two kinds of agent are defined: "people" and "others"
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”.

```plaintext
;; 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
to checkerboard-patches ;; colours background (see below for code)
```
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”.

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

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

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.
The “setup” procedure

```
;; Next we have the procedure to_setup
;; this is executed when one player starts the simulation

to setup
clear-all ;; this clears everything
checkers patch ;; colour

;; the list of colors object
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 “checkerboards-patches”
The “setup” procedure

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

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

    ; first create the other people, shown as circles,
    ; 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-kinds
        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-kinds
        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

```plaintext
; ; 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 backgroun (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 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 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

```plaintext
;;; 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 [ ;; do the same for the focus
        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 [ ;; this initialises the simulation time sys
        set shape "person"
        set size 0.2
        set age 50
        set size 0.2
        set color random-kind shift-randomly
    ]

    reset-ticks
end
```

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

This command starts the simulation time going

```
; ; 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 initialiseds the simulation time sys
```

Some Program Code

```plaintext
;;; 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
```
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.
Some Program Code

```plaintext
;; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;
;; 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 more
        ;; if with-probability
    ]

    ; 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 count
```

This asks all the “others” to do some commands (at the moment there are no commands)
Some Program Code

This asks all the “people” to do some commands (at the moment there is only one person).
Some Program Code

```lisp
;;; 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
        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

This command progresses the simulation time one unit.
Scroll down some more for...

```plaintext
; ; 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
```
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)

```plaintext
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...

```plaintext
; Finally we have definitions of the various action words/command 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-
  ;; 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.
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.
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.
Adding a slider
Adding a slider

Right-click (Mac: ctrl+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”.

2-Day Introduction to Agent-Based Modelling, Session 2, slide 14
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: 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