CS 790R Seminar
Modeling & Simulation

Introduction to NetLogo

~ Lecture 4: Tutorial based on Uri Wilensky (1999)
http://ccl.northwestern.edu/netlogo/docs ~

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Spring 2005

Introduction to NetLogo

- What is NetLogo?
- Graphical interface
- Programming concepts
- Tutorial: termites

Introduction to NetLogo

- **What is NetLogo?**
  - Modeling complex systems
  - Flash history
  - The world of NetLogo
- **Graphical interface**
- **Programming concepts**
- **Tutorial: termites**

What is NetLogo?
Modeling complex systems

- ✓ programmable modeling environment for simulating natural and social phenomena
  - well suited for modeling complex systems evolving over time
  - hundreds or thousands of independent agents operating concurrently
  - exploring the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals

What is NetLogo?
Modeling complex systems

- ✓ easy-to-use application development environment
  - opening simulations and playing with them
  - creating custom models: quickly testing hypotheses about self-organized systems
  - models library: large collection of pre-written simulations in natural and social sciences that can be used and modified
  - simple scripting language
  - user-friendly graphical interface

What is NetLogo?

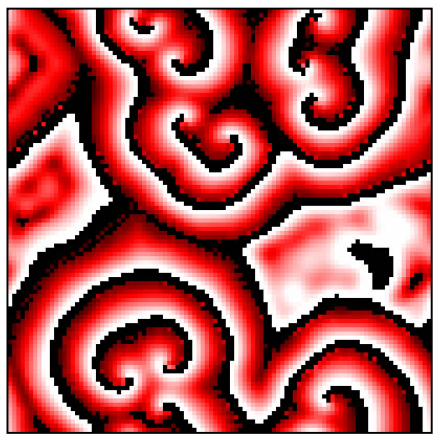
Flash history

- LOGO (Papert & Minsky, 1967)
  - theory of education based on Piaget's construc-tionism ("hands-on" creation and test of concepts)
  - simple language derived from LISP
  - turtle graphics and exploration of "microworlds"
- StarLogo (Resnick, 1991), MacStarLogo, StarLogoT
  - agent-based simulation language
  - exploring the behavior of decentralized systems through concurrent programming of 100s of turtles
- NetLogo (Wilensky, 1999)
  - further extending StarLogo (continuous turtle coordinates, cross-platform, networking, etc.)
  - most popular today (growing cooperative library of models)

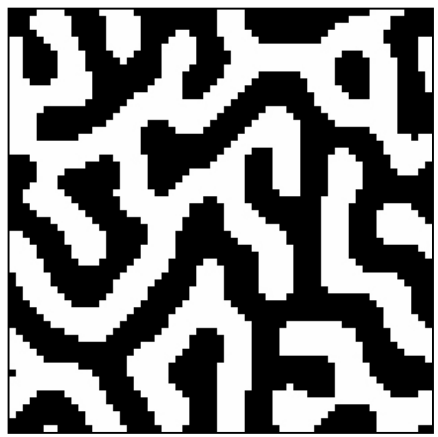
What is NetLogo?
The world of NetLogo

- ✓ NetLogo is a 2-D world made of 3 kinds of agents:
  - *patches* – make up the background or "landscape"
  - *turtles* – move around on top of the patches
  - *the observer* – oversees everything going on in the world

- ➢ examples of patch-only models



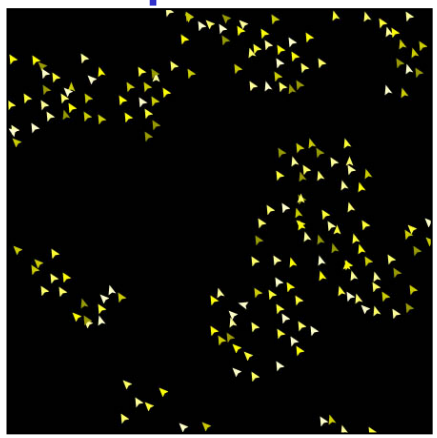
B-Z reaction



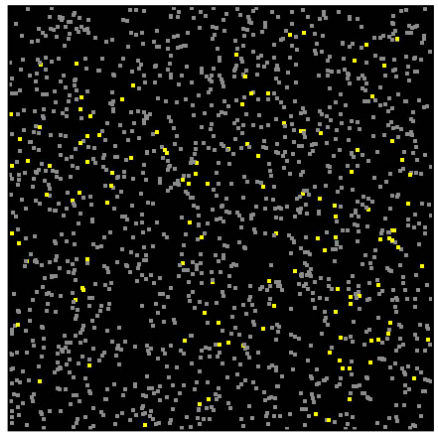
Fur

What is NetLogo?
The world of NetLogo

- examples of turtle-only models

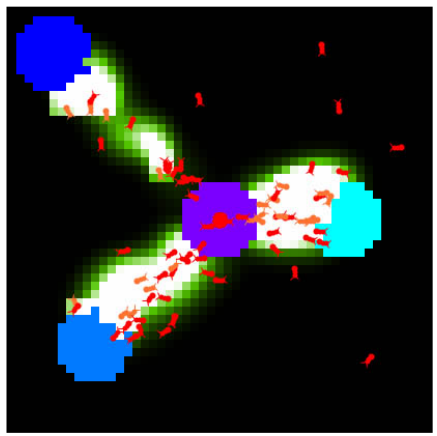


Flocking

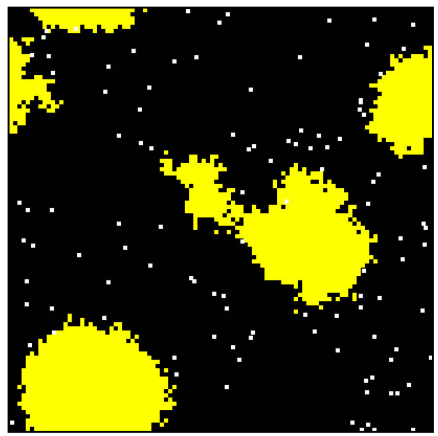


Fireflies

- examples of patch-&-turtle models



Ants



Termites

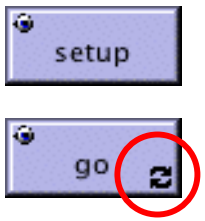
Introduction to NetLogo

- **What is NetLogo?**
- **Graphical interface**
  - Controls
  - Settings
  - Views
- **Programming concepts**
- **Tutorial: termites**

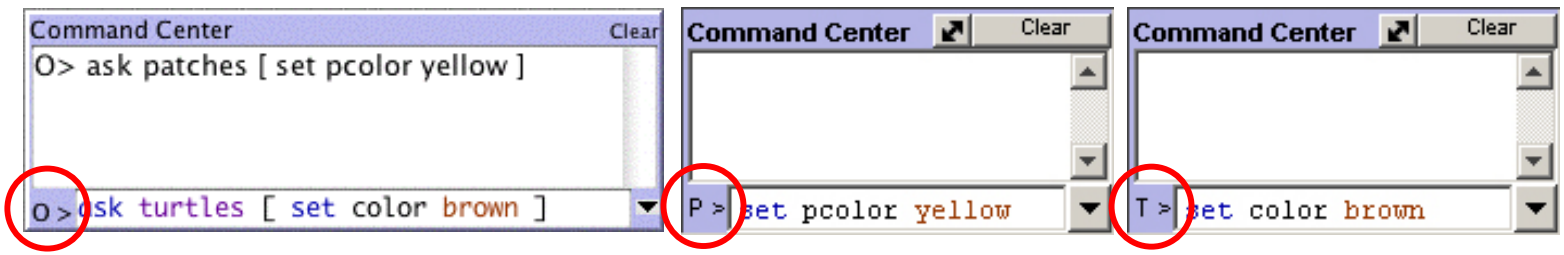
Graphical interface

Controls

- ✓ controls (BLUE) – allow to run and control the flow of execution
  - buttons
  - command center
- ➢ buttons – initialize, start, stop, step through the model



  - "once" buttons execute one action (one piece of code)
  - "forever" buttons repeat the same action (the same piece of code) until pressed again
- ➢ command center – ask observer, patches or turtles to execute specific commands "on the fly"

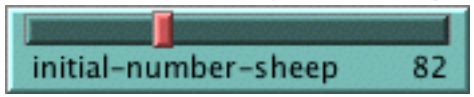


  - O> **ask patches [*commands*]** ⟺ P> ***commands***
  - O> **ask turtles [*commands*]** ⟺ T> ***commands***

Graphical interface

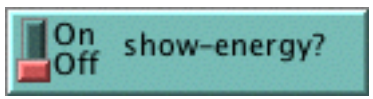
Settings

- ✓ settings (GREEN) – allow to modify parameters
  - sliders
  - switches
  - choosers
- ➢ sliders – adjust a quantity from *min* to *max* by an *increment*



- **`initial-number-sheep = 82`**

- ➢ switches – set a Boolean variable (true/false)



- **`show-energy? = false`**

- ➢ choosers – select a value from a list

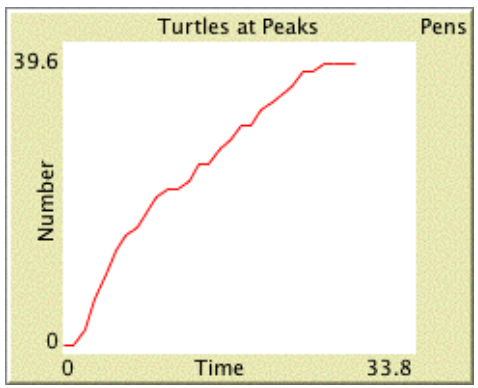
- **`file = “Beats/seth2.csv”`**

Graphical interface

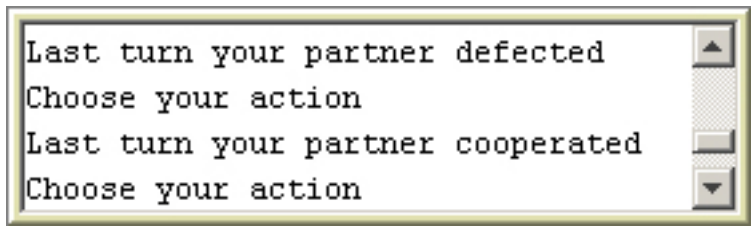
Views

- ✓ views (BEIGE) – allow to display information
  - monitors
  - plots
  - output text areas
  - graphics window
- ➢ monitors – display the current value of variables

- ➢ plots – display the history of a variable's value



- ➢ output text areas – log text info



Graphical interface

Views

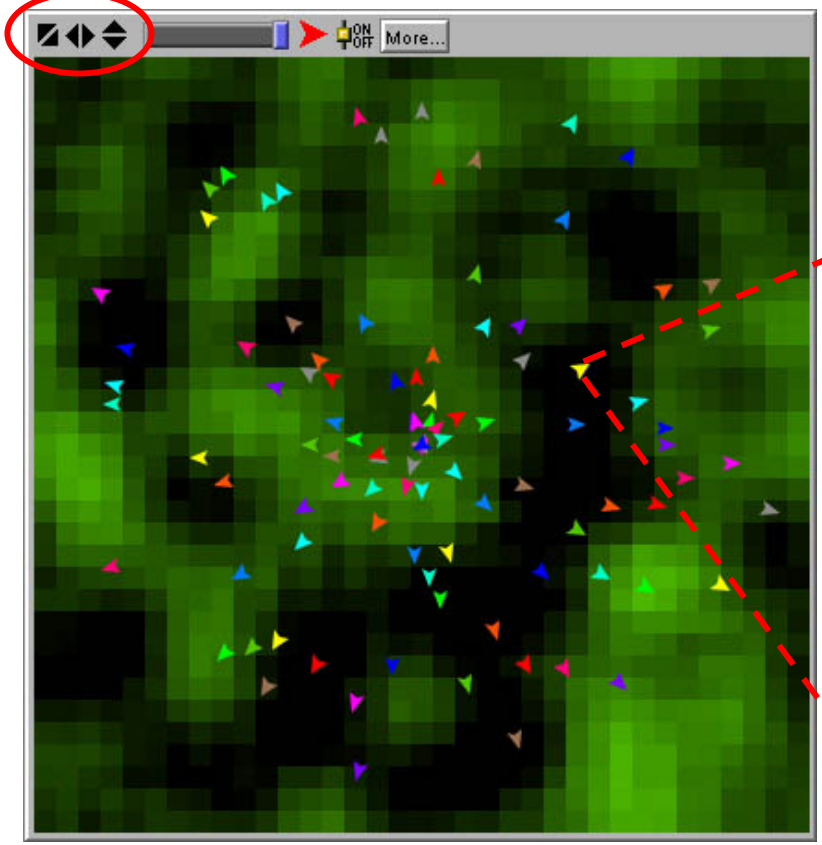
- graphics window – the main view of the 2-D NetLogo world

adjust speed

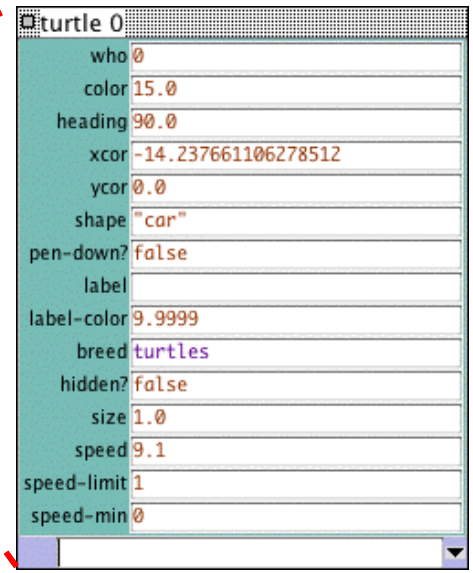
turn turtle shapes on/off

freeze/unfreeze display

change width & height in number of patches



right-click brings up turtle/patch inspector

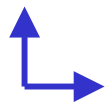


Introduction to NetLogo

- **What is NetLogo?**
- **Graphical interface**
- **Programming concepts**
  - Agents
  - Procedures
  - Variables
  - Ask
  - Agentsets
  - Breeds
  - Synchronization
- **Tutorial: termites**

Programming concepts
Agents

- ✓ agents – carry out their own activity, all simultaneously
  - patches
  - turtles
  - observer
- ➢ patches
  - don't move, form a 2-D wrap-around grid
  - have *integer* coordinates (**`pxcor`**, **`pycor`**)



- ➢ turtles
  - move on top of the patches, not necessarily in their center
  - have *decimal* coordinates (**`xcor`**, **`ycor`**) and orientation (**`heading`**)
- ➢ observer
  - can create new turtles
  - can have read/write access to all the agents and variables

Programming concepts
Procedures

- commands
  - actions for the agents to carry out (“void” functions)
  - example:

```
to setup
  ca
  crt 10
end
```

  - example with 2 input arguments:

```
to draw-polygon [ num-sides size ]
   pd
   repeat num-sides
     [ fd size
       rt (360 / num-sides) ]
end
```

Programming concepts
Procedures

- reporters
  - report a result value (functions with return type)
  - example with 1 input argument:

```
to-report absolute-value [ number ]
  ifelse number >= 0
    [ report number ]
    [ report 0 - number ]
end
```

- primitives
  - *built-in* commands or reporters (language keywords)
  - some have an abbreviated form: **`create-turtles`** ⇔ **`crt`**, **`clear-all`** ⇔ **`ca`**, etc.

- ✓ procedures
  - *custom* commands or reporters (user-made)

Programming concepts
Variables

- ✓ variables – places to store values (such as numbers or text)
  - global variables
  - turtle & patch variables
  - local variables
- ➢ global variables
  - only one value for the variable
  - every agent can access it
- ➢ turtle & patch variables
  - each turtle/patch has its own value for every turtle/patch variable
- ➢ local variables
  - defined and accessible only inside a procedure
  - scope = narrowest square brackets or procedure itself

Programming concepts
Variables

- built-in variables
  - ex. of built-in turtle variables: **`color`**, **`xcor`**, **`ycor`**, **`heading`**, etc.
  - ex. of built-in patch variables: **`pcolor`**, **`pxcor`**, **`pycor`**, etc.
- custom variables
  - defining global variables:

```
global [ clock ]
```

  - defining turtle/patch variables:

```
turtles-own [ energy speed ]
patches-own [ friction ]
```

  - defining a local variable:

```
to swap-colors [ turtle1 turtle2 ]
  let temp color-of turtle1
  ...
```

Programming concepts
Variables

- setting variables
  - setting the color of all turtles:

```
ask turtles [ set color red ]
```

  - setting the color of all patches:

```
ask patches [ set pcolor red ]
```

  - setting the color of the patches under the turtles:

```
ask turtles [ set pcolor red ]
```

  - setting the color of one turtle:

```
ask turtle 5 [ set color green ]
```

    or:

```
set color-of turtle 5 red
```

  - setting the color of one patch:

```
ask patch 2 3 [ set pcolor green ]
```

Programming concepts
Ask

- ✓ "ask" – specify commands to be run by turtles or patches
  - asking all turtles:

```
ask turtles [ fd 50 ... ]
```

  - asking all patches:

```
ask patches [ diffuse ... ]
```

  - asking one turtle:

```
ask turtle 5 [ ... ]
```

- ✓ can be factored out in button specs

```
to go [
  ask turtles [ ... ]
]
```

- ➢ observer code *cannot* be inside any "ask" block
  - ex: creating 100 turtles:

```
crt 100
```

Programming concepts
Agentsets

- ✓ agentset – definition of a subset of agents (*not* a keyword)
  - all red turtles:

```
turtles with [ color = red ]
```

  - all red turtles on the patch of the current caller (turtle or patch):

```
turtles-here with [ color = red ]
```

  - all patches on right side of screen:

```
patches with [ pxcor > 0 ]
```

  - all turtles less than 3 patches away from caller (turtle or patch):

```
turtles in-radius 3
```

  - the four patches to the east, north, west, and south of the caller:

```
patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
```

  - shorthand for those four patches:

```
neighbors4
```

Programming concepts
Agentsets

- using agentsets
  - ask such agents to execute a command

```
ask <agentset> [ ... ]
```

  - check if there are such agents:

```
show any? <agentset>
```

  - count such agents:

```
show count <agentset>
```

  - example: remove the richest turtle (with the maximum “assets” value):

```
ask max-one-of turtles [ sum assets ] [ die ]
```

singleton agentset containing the richest turtle

Programming concepts
Breeds

- ✓ breed – a "natural" kind of agentset (other species than turtles)
  - example:

```
breed [ wolves sheep ]
```

  - a new breed comes with automatically derived primitives:

```
create-<breed>
create-custom-<breed>
<breed>-here
<breed>-at
...
```

  - the breed is a turtle variable:

```
ask turtle 5 [ if breed = sheep ... ]
```

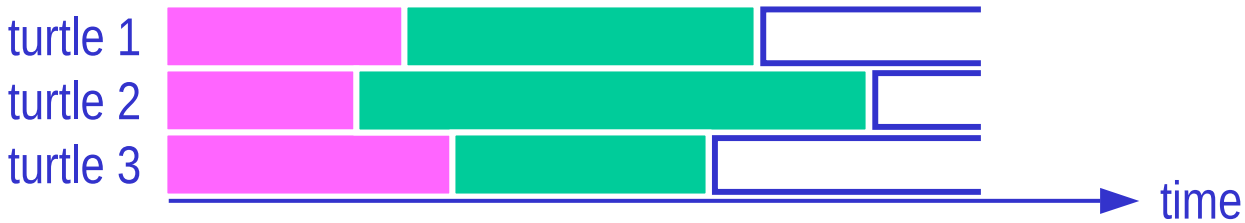
  - a turtle agent can change breed:

```
ask turtle 5 [ set breed sheep ]
```

Programming concepts
Synchronization

- ✓ agents run in parallel (each agent is an independent thread)
  - asynchronous commands:

```
ask turtles [ fd random 10
              do-calculation ... ]
```



- ✓ agent threads wait and “join” at the end of a block
  - synchronous commands:

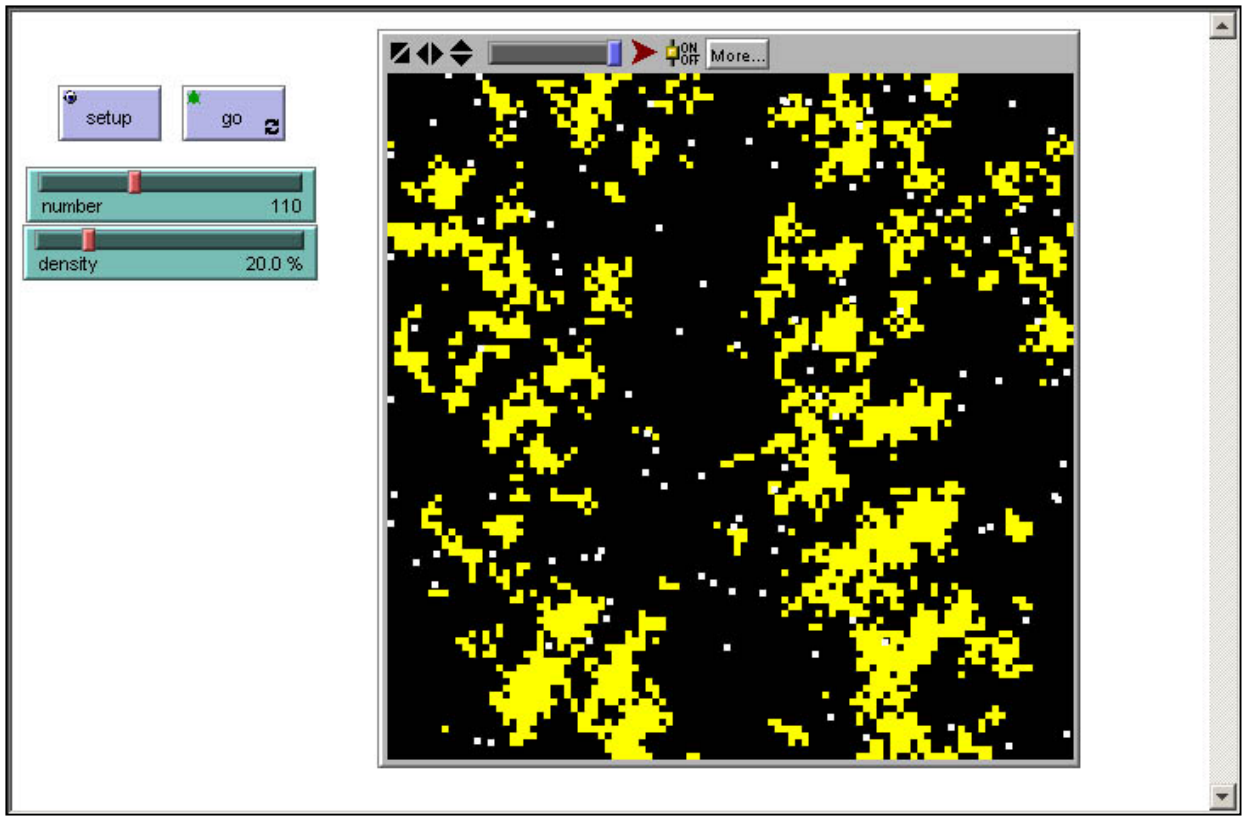
```
ask turtles [ fd random 10 ]
ask turtles [ do-calculation ] ...
```

Introduction to NetLogo

- **What is NetLogo?**
- **Graphical interface**
- **Programming concepts**
- **Tutorial: termites**
  - Interface
  - Setup
  - Go
    - Explore
    - Pick up chip
    - Find new pile
    - Drop off chip
  - Full code
  - Adding a plot

Tutorial: termites
Interface

✓ build interface



- two buttons
  - **`setup`**
    observer, once
  - **`go`**
    turtles, forever
- two sliders
  - **`number`**
    1 → 300 (1)
  - **`density`**
    0 → 100% (1)

Tutorial: termites
Setup

- ✓ randomly strew yellow wood chips (patches) with given density

```
to setup-chips
  ask patches [ if random-float 100 < density
                [ set pcolor yellow ] ]
end
```

- ✓ randomly position given number of white termites (turtles)

```
to setup-termites
  create-turtles number
  ask turtles [ set color white
                setxy random-float screen-size-x
                      random-float screen-size-y ]
end
```

- ✓ setup all

```
to setup  ca setup-chips setup-termites
end
```

Tutorial: termites
Go

- ✓ termites (turtles) follow 3 rules:
  1. look around for a wood chip and pick it up
  2. look around for a pile of wood chips
  3. look around for an empty spot in the pile and drop off the chip

```
to go                    ; turtle code
  pick-up-chip
  find-new-pile
  drop-off-chip
end
```

Tutorial: termites

Go – *Explore*

- ✓ termites (turtles) explore the environment through random walk

```
to explore
  fd 1
  rt random-float 50
  lt random-float 50
end
```

```
to explore
  fd 1
    rt random-float 50
       - random-float 50
end
```

Tutorial: termites

Go – *Pick up chip*

✓ find a wood chip, pick it up and turn orange (recursive versions)

```
to pick-up-chip
  ifelse pcolor = yellow
    [ stamp black
      set color orange ]
    [ explore
      pick-up-chip ]
end
```

```
to pick-up-chip
  if pcolor = yellow
    [ stamp black
      set color orange stop ]
  explore
  pick-up-chip
end
```

→ nonrecursive version

```
to pick-up-chip
  while [ pcolor != yellow ]
    [ explore ]
  stamp black
  set color orange
end
```

Tutorial: termites

Go – *Find new pile*

✓ find a new pile of chips (recursive versions)

```
to find-new-pile
  if pcolor != yellow
    [ explore
      find-new-pile ]
end
```

```
to find-new-pile
  if pcolor = yellow [ stop ]
  explore
  find-new-pile
end
```

→ nonrecursive version

```
to find-new-pile
  while [ pcolor != yellow ]
    [ explore ]
end
```

Tutorial: termites

Go – *Drop off chip*

✓ find an empty spot, drop off chip and get away (recursive versions)

```
to drop-off-chip
  ifelse pcolor = black
   [ stamp yellow
     set color white
     fd 20 ]
   [ explore
     drop-off-chip ]
end
```

```
to drop-off-chip
  if pcolor = black
    [ stamp yellow
      set color white
      fd 20 stop ]
  explore
  drop-off-chip
end
```

→ nonrecursive version

```
to drop-off-chip
  while [ pcolor != black ]
    [ explore ]
  stamp yellow
  set color white  fd 20
end
```

Tutorial: termites
Full code

```
to setup
  ca
  setup-chips
  setup-termites
end

to setup-chips
  ask patches [
    if random-float 100 < density
      [ set pcolor yellow ] ]
end

to setup-termites
  create-turtles number
  ask turtles [
    set color white
    setxy random-float screen-size-x
          random-float screen-size-y ]
end

to explore
  fd 1
  rt random-float 50
  lt random-float 50
end
```

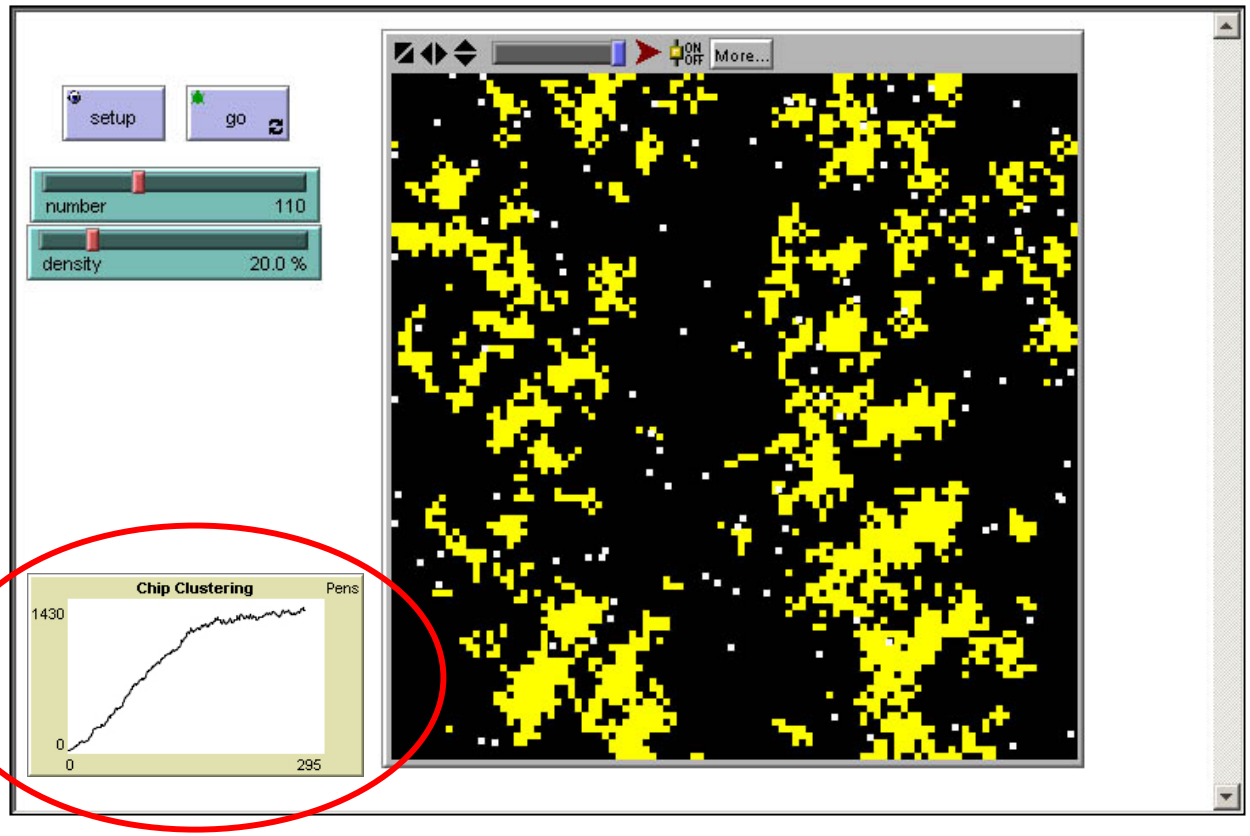
```
to go              ; turtle code
  pick-up-chip
  find-new-pile
  drop-off-chip
end

to pick-up-chip
  while [ pcolor != yellow ]
    [ explore ]
  stamp black
  set color orange
end

to find-new-pile
  while [ pcolor != yellow ]
    [ explore ]
end

to drop-off-chip
  while [ pcolor != black ]
    [ explore ]
  stamp yellow
  set color white
  fd 20
end
```

Tutorial: termites
Adding a plot



- one plot
  - "Chip Clustering"

Tutorial: termites
Adding a plot

- ✓ plotting

```
to draw-plot
  set-current-plot "Chip Clustering"
  plot count patches with
    [ count neighbors4 with [ pcolor = yellow] = 4 ]
end
```

- ✓ modifying “go” to become observer code

```
to go                ; turtle code
  ask turtles [
    pick-up-chip
    find-new-pile
    drop-off-chip ]
  draw-plot
end
```

Introduction to NetLogo

- What is NetLogo?
- Graphical interface
- Programming concepts
- Tutorial: termites