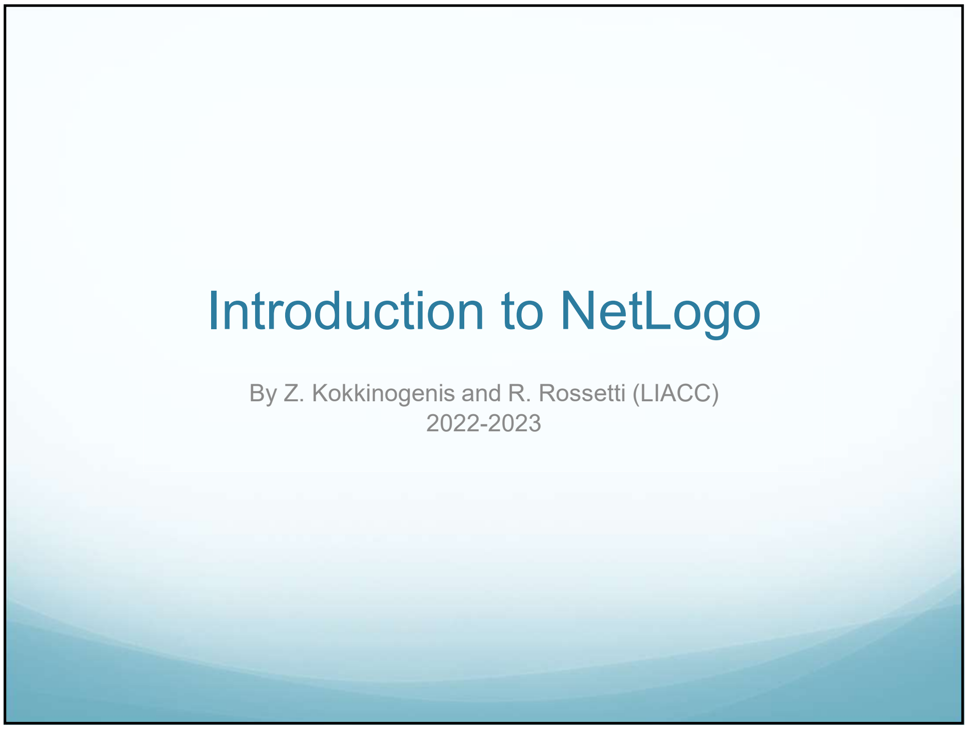
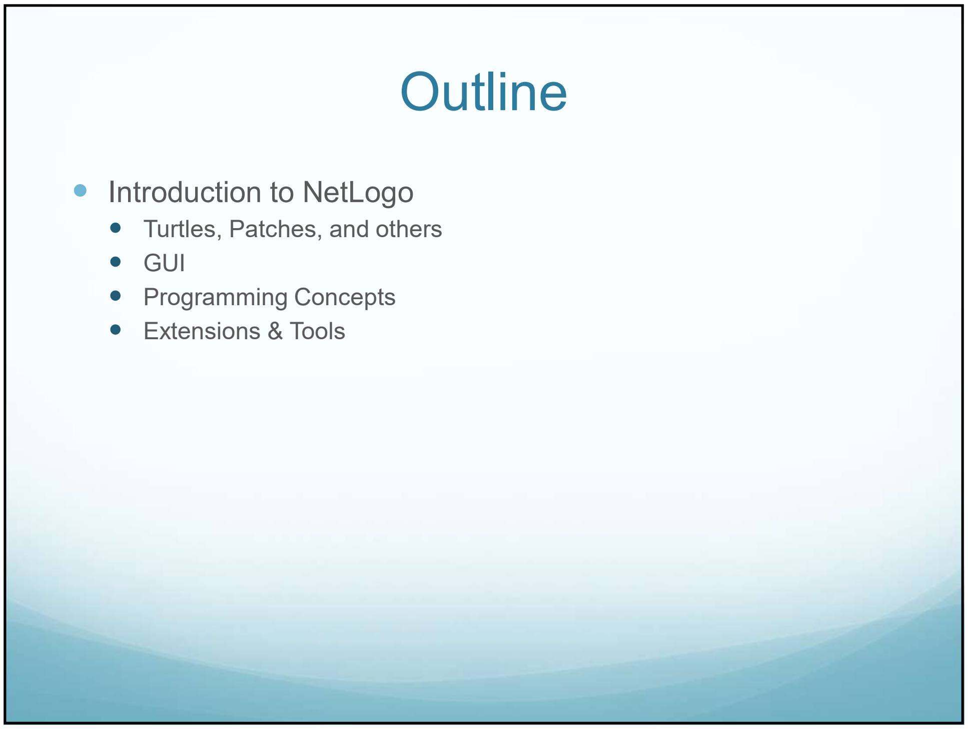


Introduction to NetLogo

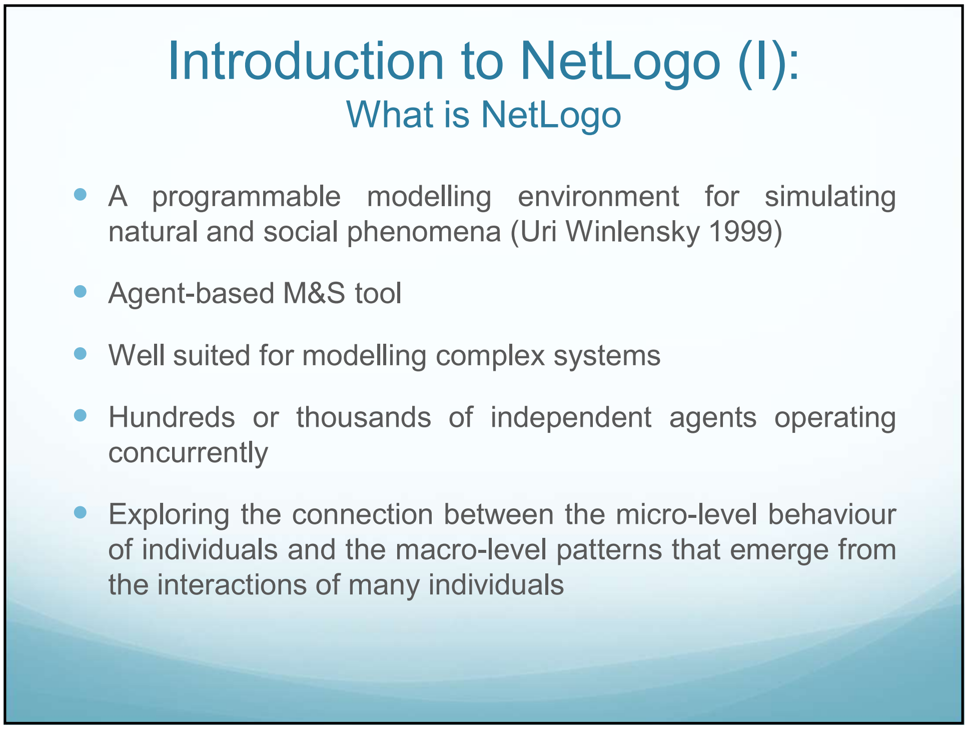
By Z. Kokkinogenis and R. Rossetti (LIACC)

2022-2023



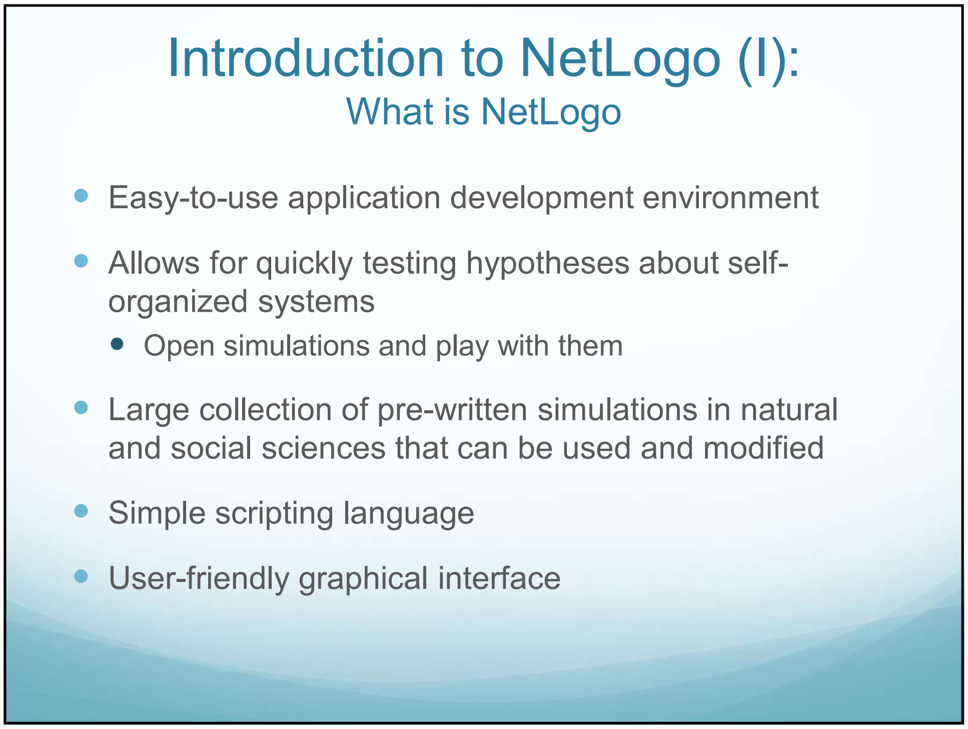
Outline

- Introduction to NetLogo
  - Turtles, Patches, and others
  - GUI
  - Programming Concepts
  - Extensions & Tools



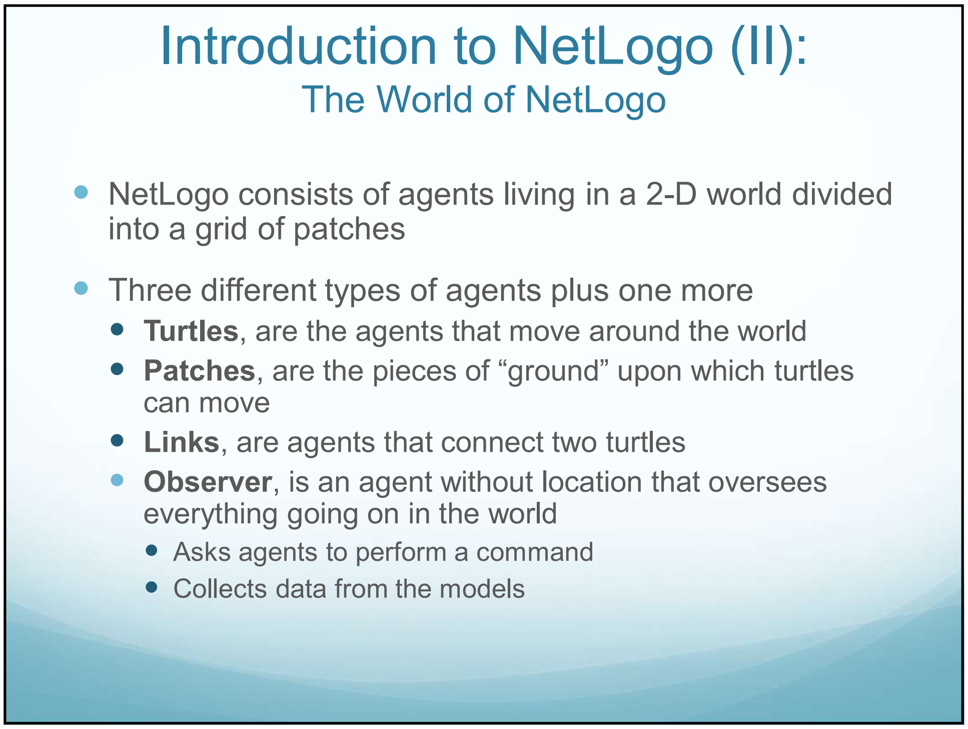
Introduction to NetLogo (I):
What is NetLogo

- A programmable modelling environment for simulating natural and social phenomena (Uri Winlensky 1999)
- Agent-based M&S tool
- Well suited for modelling complex systems
- Hundreds or thousands of independent agents operating concurrently
- Exploring the connection between the micro-level behaviour of individuals and the macro-level patterns that emerge from the interactions of many individuals



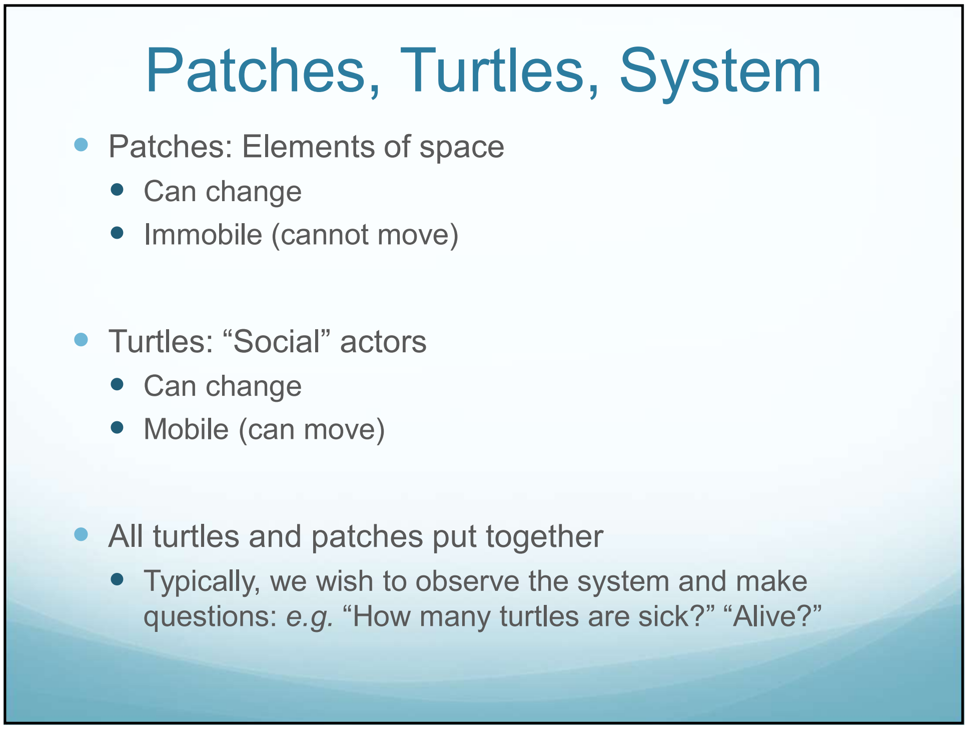
Introduction to NetLogo (I):
What is NetLogo

- Easy-to-use application development environment
- Allows for quickly testing hypotheses about self-organized systems
  - Open simulations and play with them
- Large collection of pre-written simulations in natural and social sciences that can be used and modified
- Simple scripting language
- User-friendly graphical interface



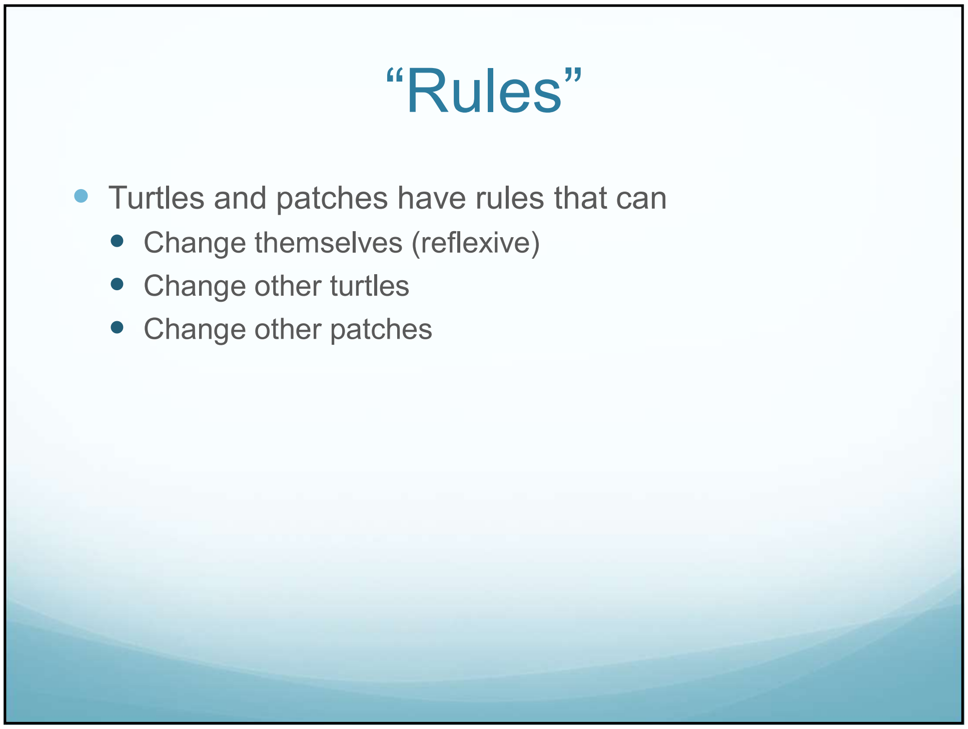
Introduction to NetLogo (II):
The World of NetLogo

- NetLogo consists of agents living in a 2-D world divided into a grid of patches
- Three different types of agents plus one more
  - **Turtles**, are the agents that move around the world
  - **Patches**, are the pieces of “ground” upon which turtles can move
  - **Links**, are agents that connect two turtles
  - **Observer**, is an agent without location that oversees everything going on in the world
    - Asks agents to perform a command
    - Collects data from the models



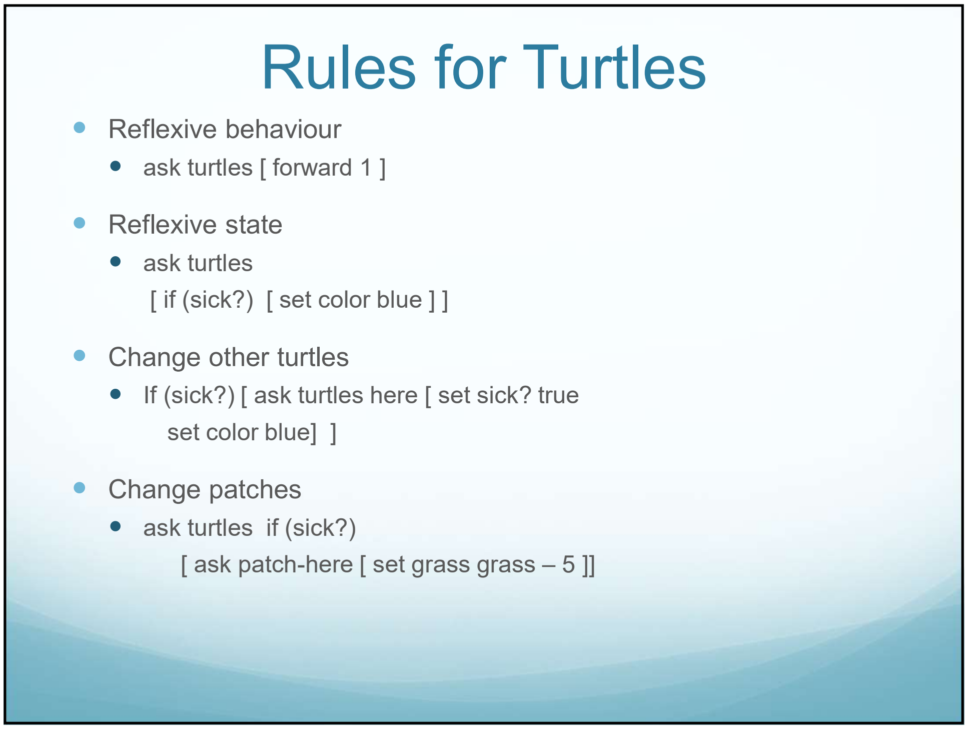
Patches, Turtles, System

- Patches: Elements of space
  - Can change
  - Immobile (cannot move)
- Turtles: “Social” actors
  - Can change
  - Mobile (can move)
- All turtles and patches put together
  - Typically, we wish to observe the system and make questions: *e.g.* “How many turtles are sick?” “Alive?”



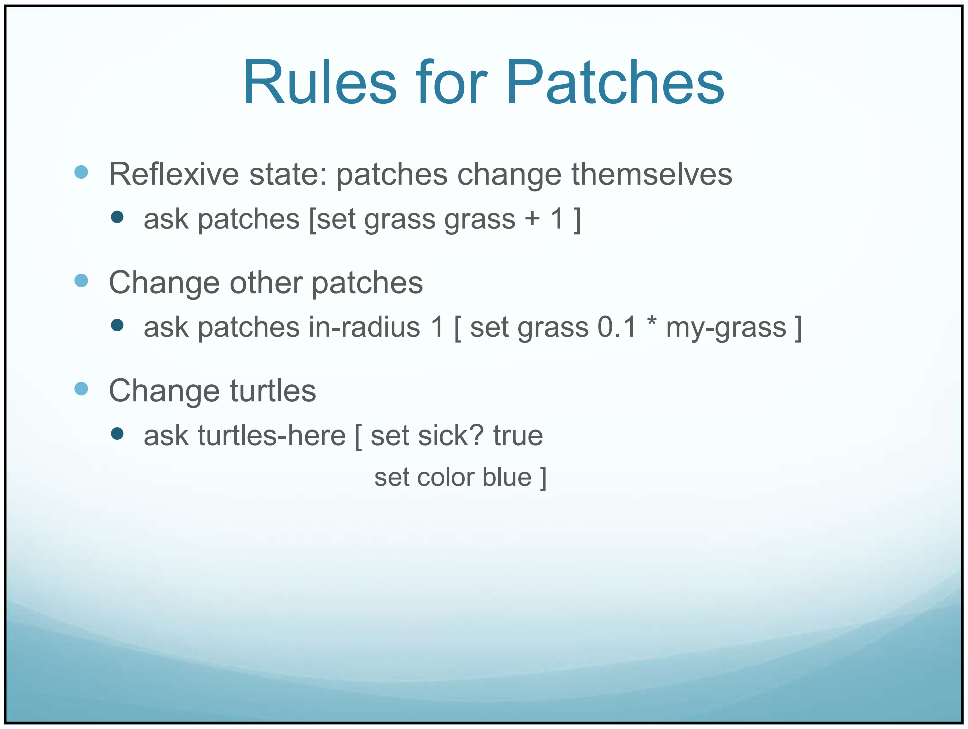
“Rules”

- Turtles and patches have rules that can
  - Change themselves (reflexive)
  - Change other turtles
  - Change other patches



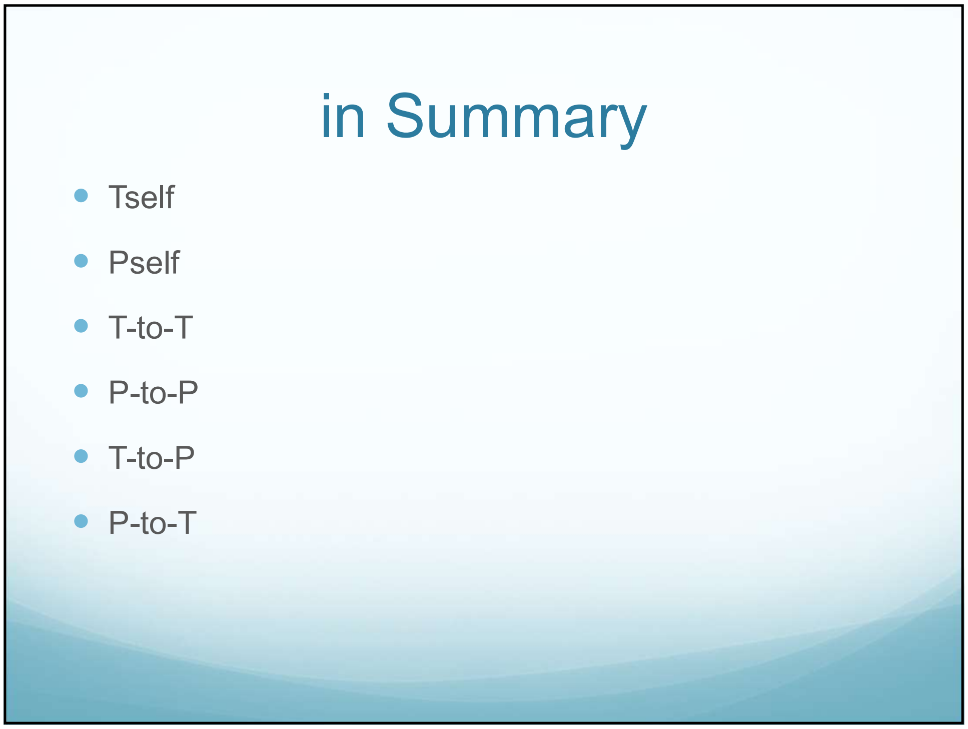
Rules for Turtles

- Reflexive behaviour
  - ask turtles [forward 1]
- Reflexive state
  - ask turtles
    [if (sick?) [set color blue]]
- Change other turtles
  - If (sick?) [ask turtles here [set sick? true
      set color blue]]
- Change patches
  - ask turtles if (sick?)
        [ask patch-here [set grass grass – 5]]



Rules for Patches

- Reflexive state: patches change themselves
  - ask patches [set grass grass + 1]
- Change other patches
  - ask patches in-radius 1 [set grass 0.1 * my-grass]
- Change turtles
  - ask turtles-here [set sick? true
set color blue]

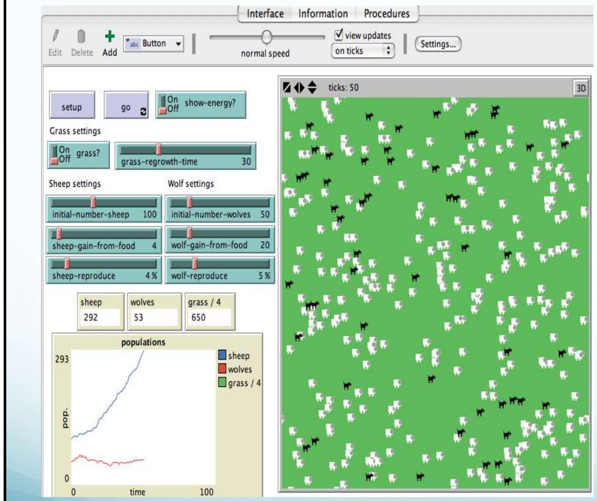


in Summary

- Tself
- Pself
- T-to-T
- P-to-P
- T-to-P
- P-to-T

Introduction to NetLogo (III):

GUI - Controls, Settings, Views



On Off show-energy?

Grass settings

On Off grass?

grass-regrowth-time 30

Sheep settings

initial-number-sheep 100

sheep-gain-from-food 4

sheep-reproduce 4 %

Wolf settings

initial-number-wolves 50

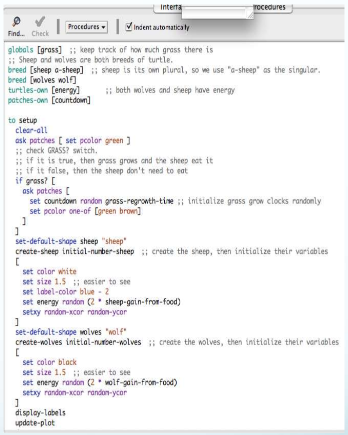
wolf-gain-from-food 20

wolf-reproduce 5 %

sheep 292

wolves 53

grass / 4 650



☑ Indent automatically

```
globals [grass]  ;; keep track of how much grass there is
;; Sheep and wolves are both breeds of turtle.
breed [sheep a-sheep]  ;; sheep is its own plural, so we use "a-sheep" as the singular.
breed [wolves wolf]
turtles-own [energy]       ;; both wolves and sheep have energy
patches-own [countdown]

to setup
  clear-all
  ask patches [ set pcolor green ]
  ;; check GRASS? switch.
  ;; if it is true, then grass grows and the sheep eat it
  ;; if it false, then the sheep don't need to eat
  if grass? [
    ask patches [
      set countdown random grass-regrowth-time ;; initialize grass grow clocks randomly
      set pcolor one-of [green brown]
    ]
  ]
  set-default-shape sheep "sheep"
  create-sheep initial-number-sheep  ;; create the sheep, then initialize their variables
  [
    set color white
    set size 1.5  ;; easier to see
    set label-color blue - 2
    set energy random (2 * sheep-gain-from-food)
    setxy random-xcor random-ycor
  ]
  set-default-shape wolves "wolf"
  create-wolves initial-number-wolves  ;; create the wolves, then initialize their variables
  [
    set color black
    set size 1.5  ;; easier to see
    set energy random (2 * wolf-gain-from-food)
    setxy random-xcor random-ycor
  ]
  display-labels
  update-plot
```

Introduction to NetLogo (III):

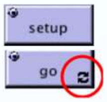
GUI - Controls, Settings, Views

- **controls** (BLUE) - allow to run and control the flow of execution
  - buttons
  - command centre
- **settings** (GREEN) - allow to modify parameters
  - sliders
  - switches
  - choosers
- **views** (BEIGE) - allow to display information
  - monitors
  - plots
  - output text areas
  - graphics window

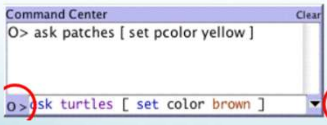
Introduction to NetLogo (III):

GUI - Controls

- Controls - 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 step)
  - ”Forever” buttons repeat the same action

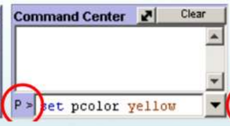


- Command center - asks observer, patches or turtles to execute specific commands during the execution



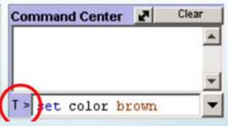
Command Center Clear

```
O> ask patches [ set pcolor yellow ]
O> ask turtles [ set color brown ]
```



Command Center Clear

```
P> set pcolor yellow
```



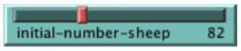
Command Center Clear

```
T> set color brown
```

Introduction to NetLogo (IV):

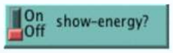
GUI - Settings

- Settings - allow to modify parameters
  - Sliders
  - Switches
- Sliders - adjust a quantity from min to max values by an increment



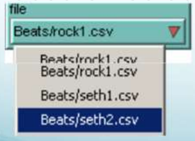
initial-number-sheep 82

- Switches - set a Boolean variable (true/false)



On Off show-energy?

- Choosers - select a value from a list (enumeration)



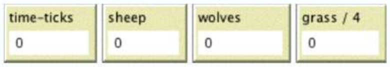
file

Beats/rock1.csv

Beats/rock1.csv
Beats/rock1.csv
Beats/seth1.csv
Beats/seth2.csv

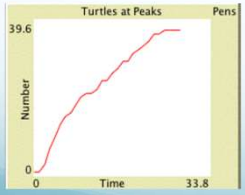
Introduction to NetLogo (V):
GUI - Views

- Views - allow to display information
  - Monitors
  - Plots
  - Graphics window
- Monitors - display the current value of variables



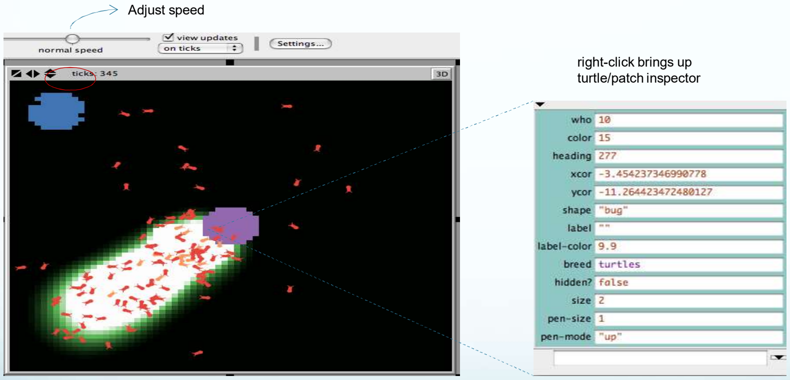
time-ticks	sheep	wolves	grass / 4
0	0	0	0

- Plots - display the history of variables' values over time



Introduction to NetLogo (V):
GUI - Views

- Graphics window - The main view of the 2-D NetLogo world



Adjust speed

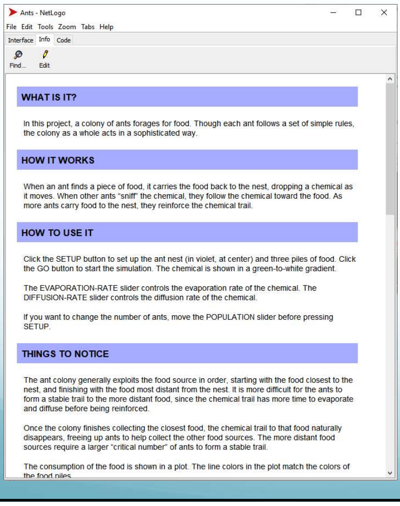
right-click brings up turtle/patch inspector

who	10
color	15
heading	277
xcor	-3.454237346990778
ycor	-11.264423472480127
shape	"bug"
label	""
label-color	9.9
breed	turtles
hidden?	false
size	2
pen-size	1
pen-mode	"up"

Introduction to NetLogo (V):

GUI - Views

- Info and Code windows, to document the model and to write the model code



WHAT IS IT?

In this project, a colony of ants forages for food. Though each ant follows a set of simple rules, the colony as a whole acts in a sophisticated way.

HOW IT WORKS

When an ant finds a piece of food, it carries the food back to the nest, dropping a chemical as it moves. When other ants "sniff" the chemical, they follow the chemical toward the food. As more ants carry food to the nest, they reinforce the chemical trail.

HOW TO USE IT

Click the SETUP button to set up the ant nest (in violet, at center) and three piles of food. Click the GO button to start the simulation. The chemical is shown in a green-to-white gradient.

The EVAPORATION-RATE slider controls the evaporation rate of the chemical. The DIFFUSION-RATE slider controls the diffusion rate of the chemical.

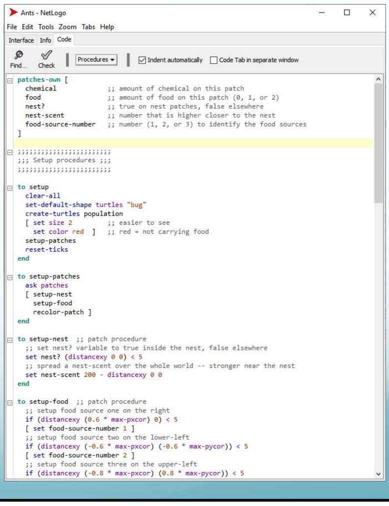
If you want to change the number of ants, move the POPULATION slider before pressing SETUP.

THINGS TO NOTICE

The ant colony generally exploits the food source in order, starting with the food closest to the nest, and finishing with the food most distant from the nest. It is more difficult for the ants to form a stable trail to the more distant food, since the chemical trail has more time to evaporate and diffuse before being reinforced.

Once the colony finishes collecting the closest food, the chemical trail to that food naturally disappears, freeing up ants to help collect the other food sources. The more distant food sources require a larger "critical number" of ants to form a stable trail.

The consumption of the food is shown in a plot. The line colors in the plot match the colors of the food piles



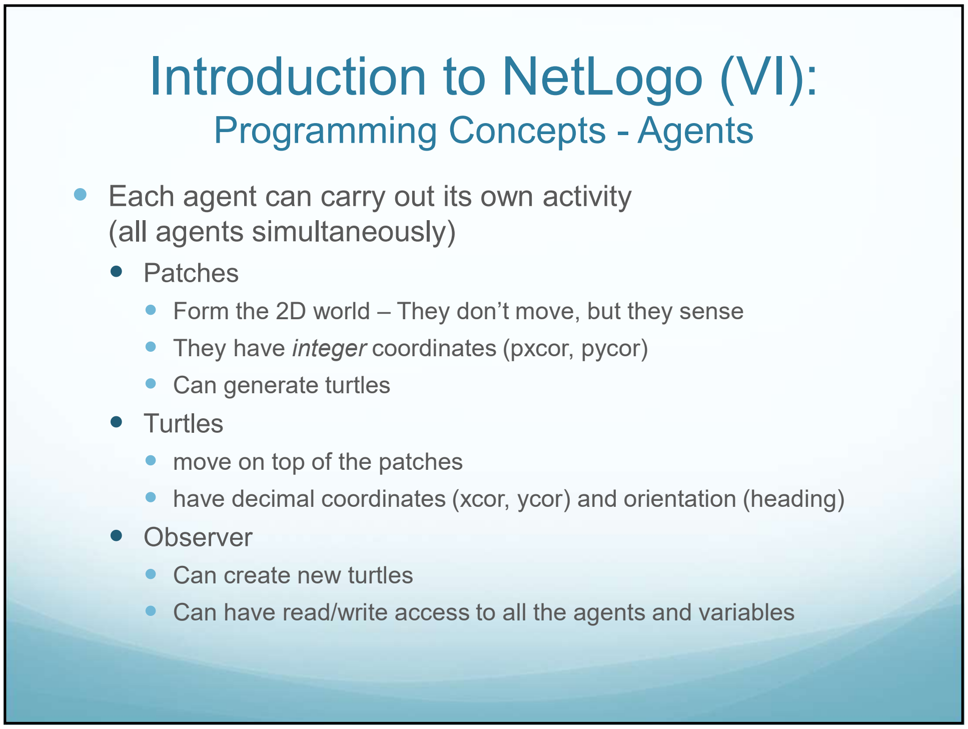
☑ Indent automatically ☐ Code Tab in separate window

```
patches-own [
  chemical             ;; amount of chemical on this patch
  food                 ;; amount of food on this patch (0, 1, or 2)
  nest?                ;; true on nest patches, false elsewhere
  nest-scent           ;; number that is higher closer to the nest
  food-source-number   ;; number (1, 2, or 3) to identify the food sources
]

;;;;;;;;;;;;;;;;;;;;;;;;
;;; Setup procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;

to setup
  clear-all
  set-default-shape turtles "bug"
  create-turtles population
  [ set size 2         ;; easier to see
    set color red  ]   ;; red = not carrying food
  setup-patches
  reset-ticks
end

to setup-patches
  ask patches
  [ setup-nest
    setup-food
    recolor-patch ]
end

to setup-nest  ;; patch procedure
  ;; set nest? variable to true inside the nest, false elsewhere
  set nest? (distancexy 0 0) < 5
  ;; spread a nest-scent over the whole world -- stronger near the nest
  set nest-scent 200 - distancexy 0 0
end

to setup-food  ;; patch procedure
  ;; setup food source one on the right
  if (distancexy (0.6 * max-pxcor) 0) < 5
  [ set food-source-number 1 ]
  ;; setup food source two on the lower-left
  if (distancexy (-0.6 * max-pxcor) (-0.6 * max-pycor)) < 5
  [ set food-source-number 2 ]
  ;; setup food source three on the upper-left
  if (distancexy (-0.8 * max-pxcor) (0.8 * max-pycor)) < 5
```

Introduction to NetLogo (VI):

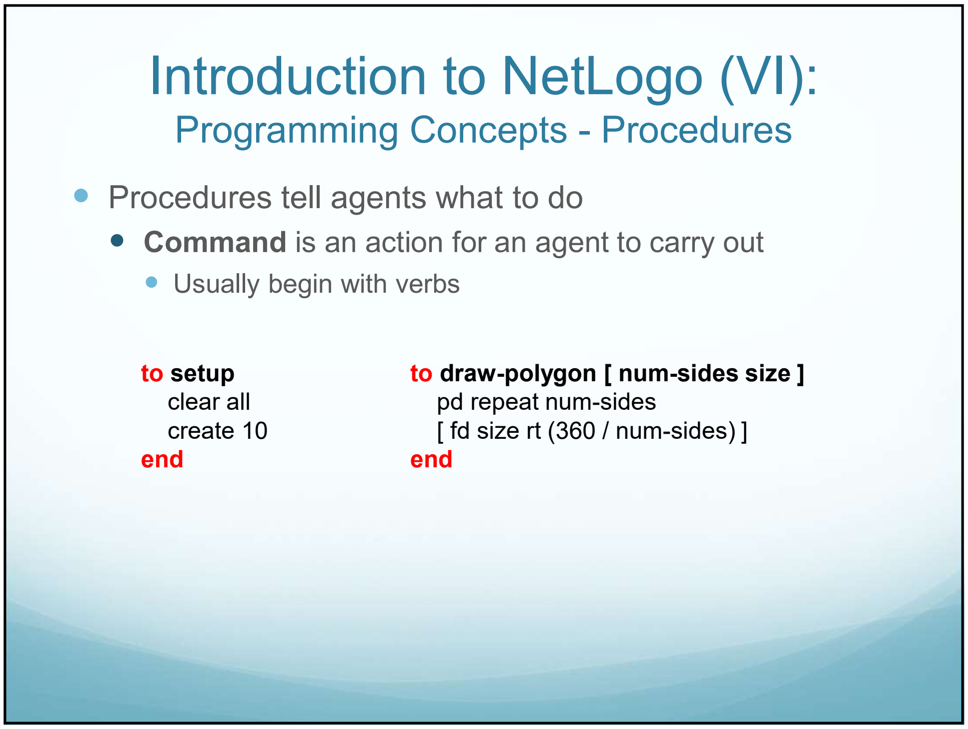
Programming Concepts

- Agents
- Procedures
- Variables
- Ask
- Agentsets
- Breeds
- Synchronization



Introduction to NetLogo (VI):
Programming Concepts - Agents

- Each agent can carry out its own activity
  (all agents simultaneously)
  - Patches
    - Form the 2D world – They don’t move, but they sense
    - They have *integer* coordinates (pxcor, pycor)
    - Can generate turtles
  - Turtles
    - move on top of the patches
    - have decimal coordinates (xcor, ycor) and orientation (heading)
  - Observer
    - Can create new turtles
    - Can have read/write access to all the agents and variables

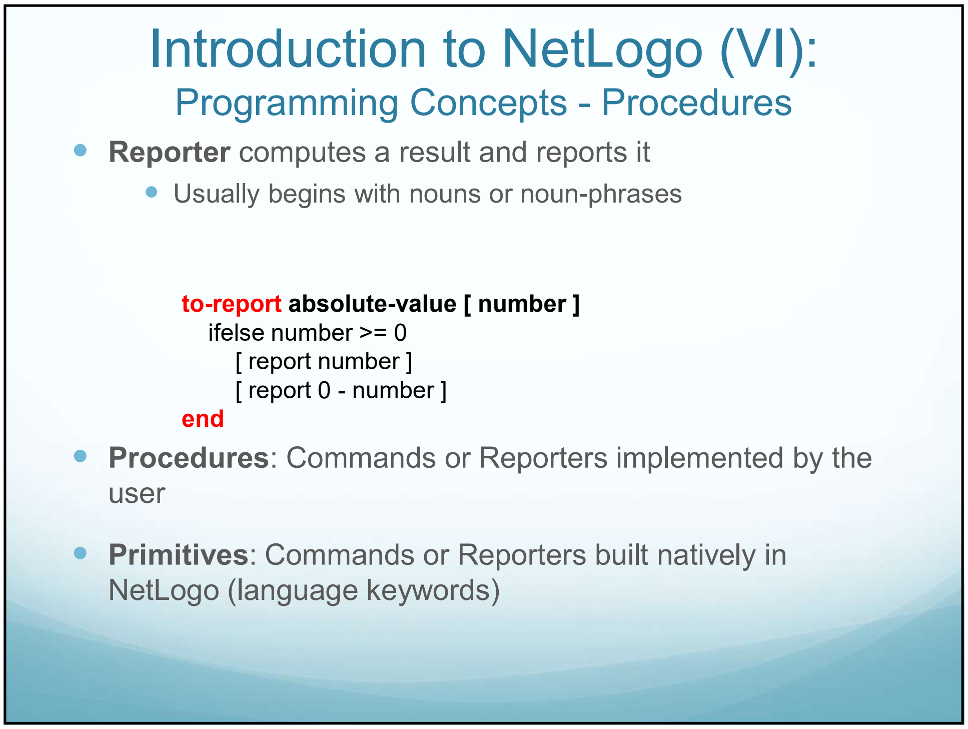


Introduction to NetLogo (VI):
Programming Concepts - Procedures

- Procedures tell agents what to do
  - **Command** is an action for an agent to carry out
    - Usually begin with verbs

```
to setup
   clear all
   create 10
end
```

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



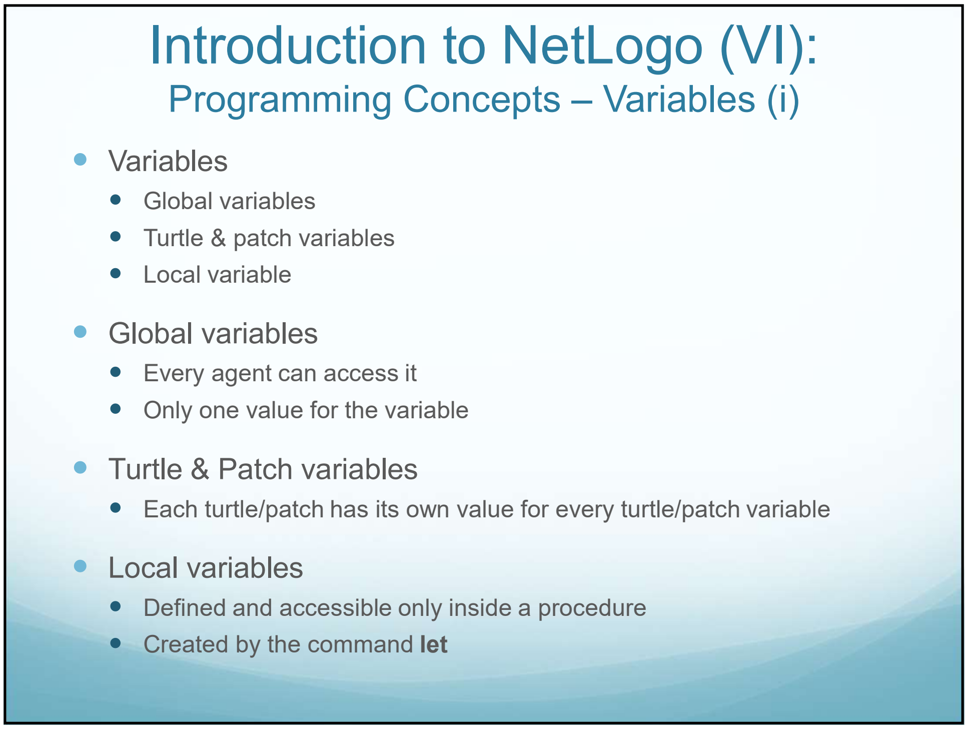
Introduction to NetLogo (VI):

Programming Concepts - Procedures

- **Reporter** computes a result and reports it
  - Usually begins with nouns or noun-phrases

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

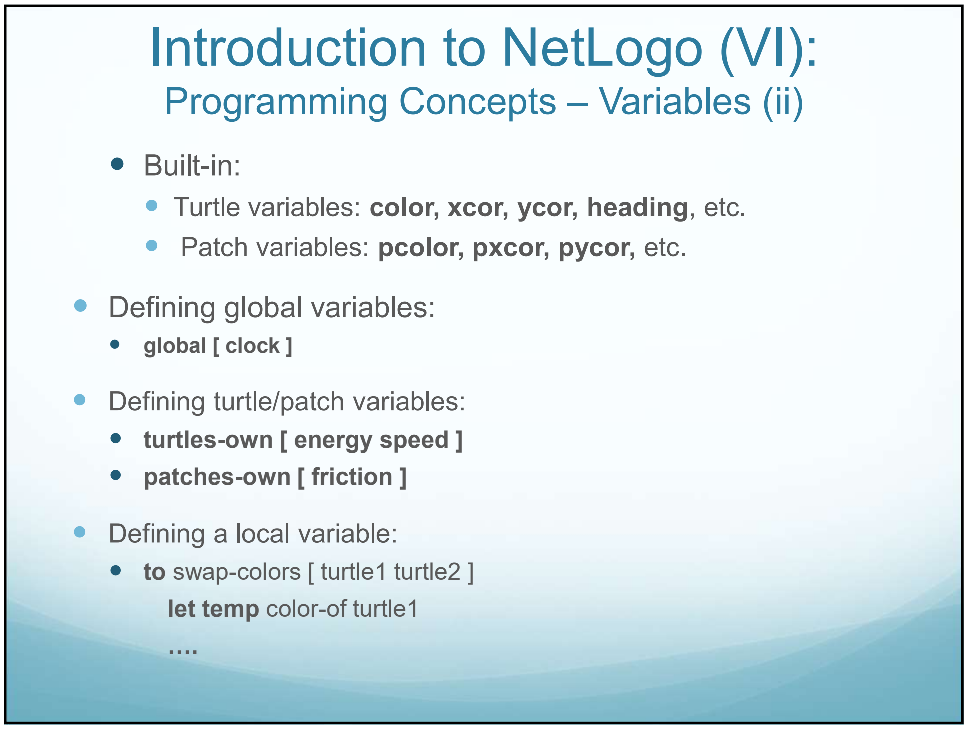
- **Procedures**: Commands or Reporters implemented by the user
- **Primitives**: Commands or Reporters built natively in NetLogo (language keywords)



Introduction to NetLogo (VI):

Programming Concepts – Variables (i)

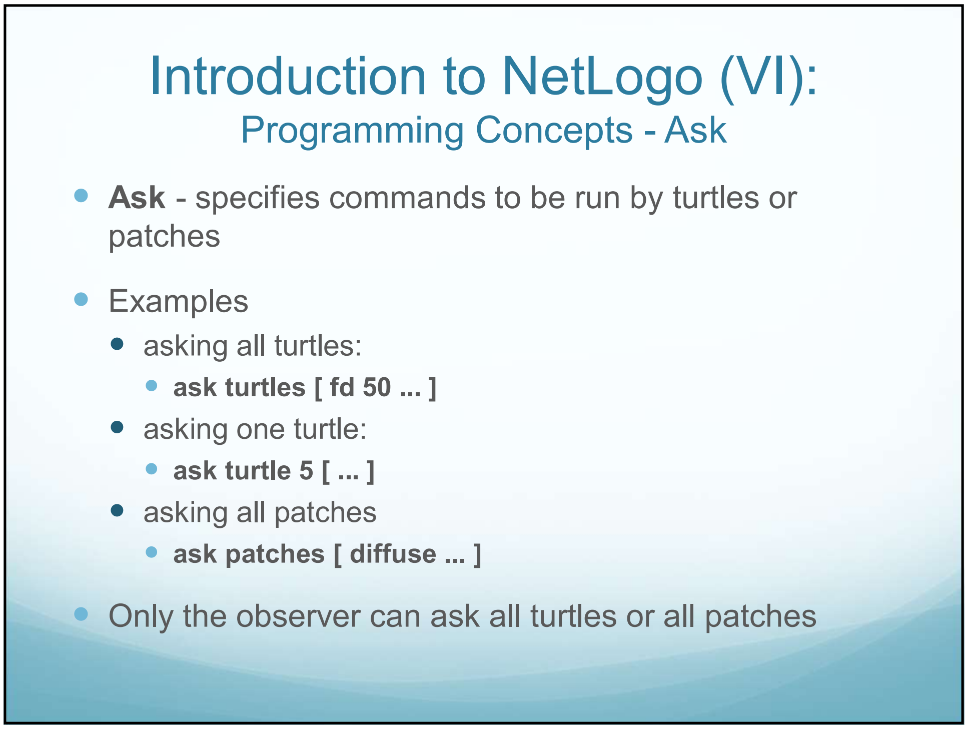
- Variables
  - Global variables
  - Turtle & patch variables
  - Local variable
- Global variables
  - Every agent can access it
  - Only one value for the variable
- Turtle & Patch variables
  - Each turtle/patch has its own value for every turtle/patch variable
- Local variables
  - Defined and accessible only inside a procedure
  - Created by the command **let**



Introduction to NetLogo (VI):

Programming Concepts – Variables (ii)

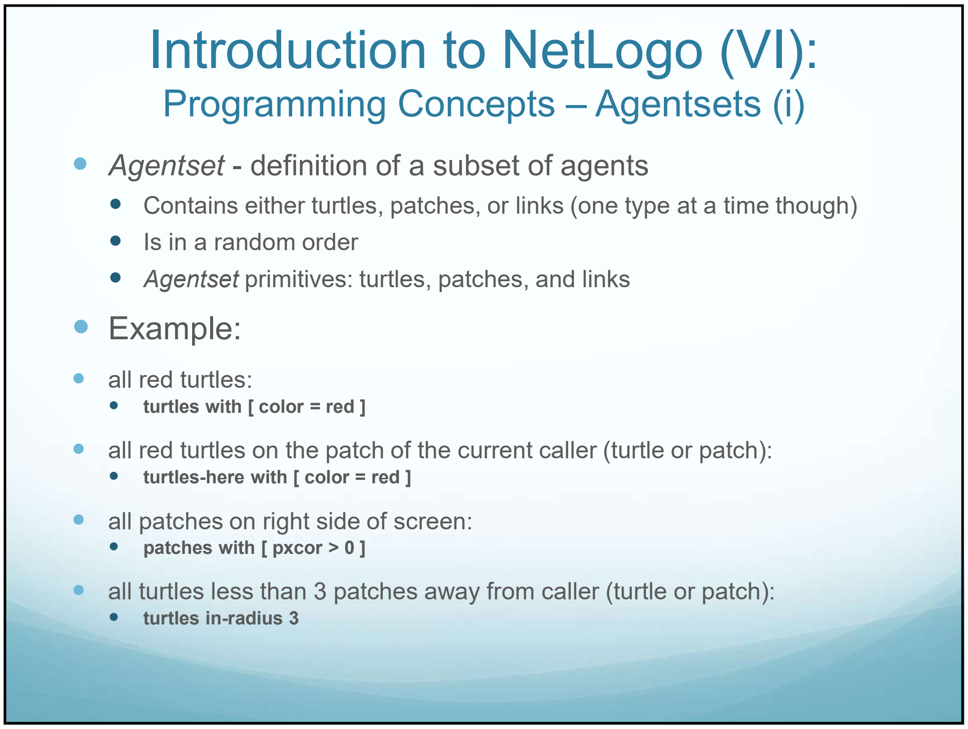
- Built-in:
  - Turtle variables: **color, xcor, ycor, heading**, etc.
  - Patch variables: **pcolor, pxcor, pycor,** etc.
- Defining global variables:
  - **global [clock]**
- Defining turtle/patch variables:
  - **turtles-own [energy speed]**
  - **patches-own [friction]**
- Defining a local variable:
  - **to** swap-colors [turtle1 turtle2]
    let temp color-of turtle1
    ….



Introduction to NetLogo (VI):

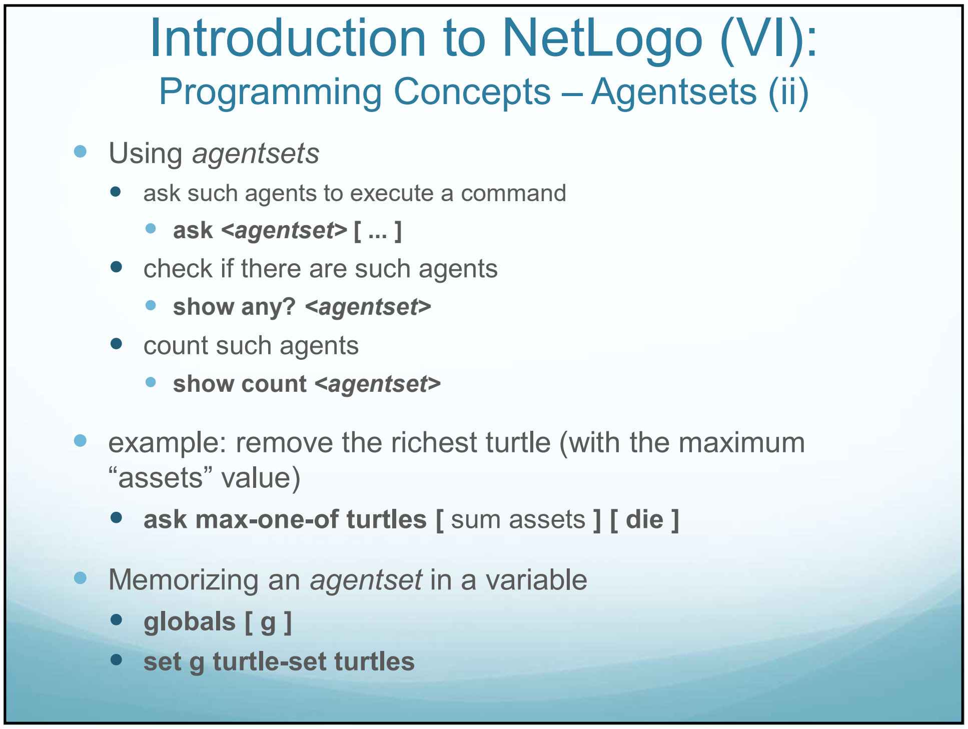
Programming Concepts - Ask

- **Ask** - specifies commands to be run by turtles or patches
- Examples
  - asking all turtles:
    - **ask turtles [fd 50 ...]**
  - asking one turtle:
    - **ask turtle 5 [...]**
  - asking all patches
    - **ask patches [diffuse ...]**
- Only the observer can ask all turtles or all patches



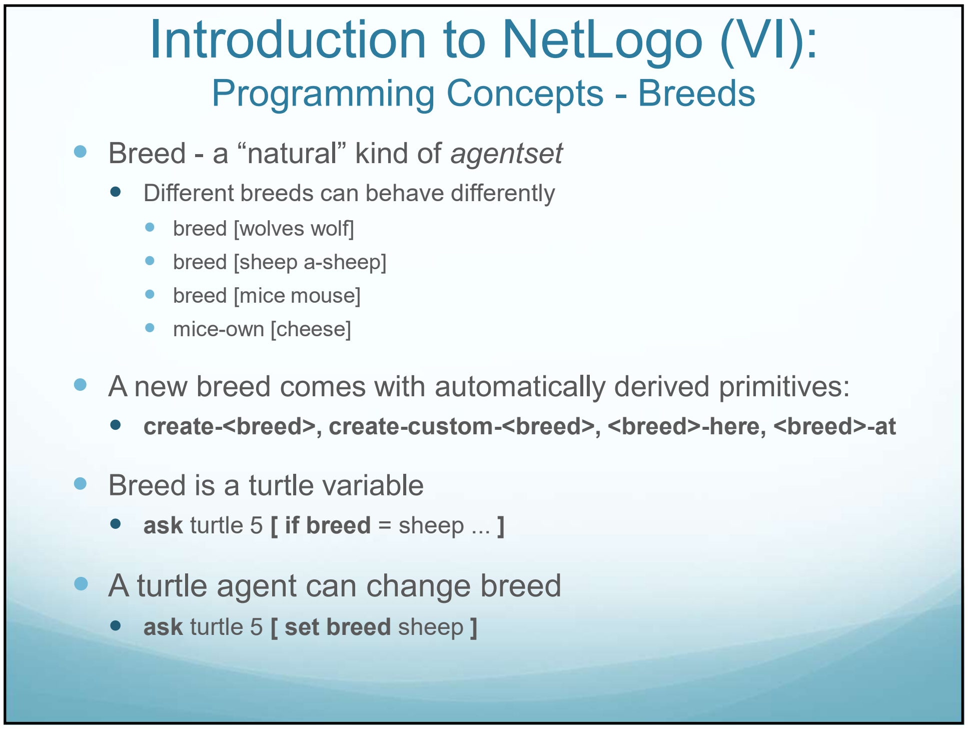
Introduction to NetLogo (VI):
Programming Concepts – Agentsets (i)

- *Agentset* - definition of a subset of agents
  - Contains either turtles, patches, or links (one type at a time though)
  - Is in a random order
  - *Agentset* primitives: turtles, patches, and links
- Example:
- 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**



Introduction to NetLogo (VI):
Programming Concepts – Agentsets (ii)

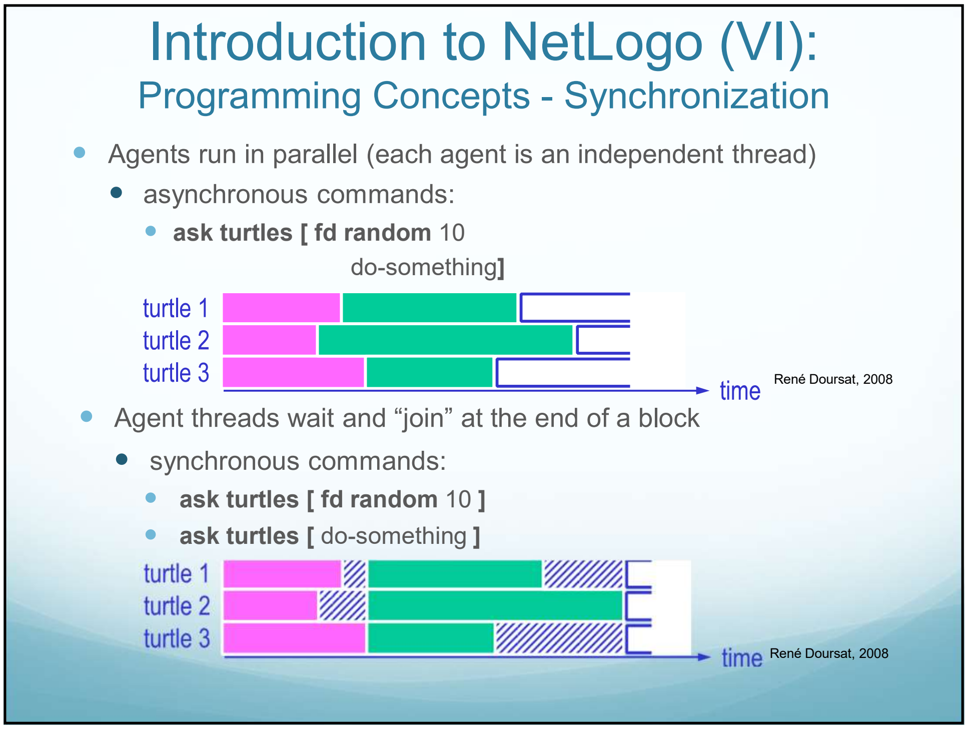
- Using *agentsets*
  - ask such agents to execute a command
    - **ask <*agentset*> […]**
  - check if there are such agents
    - **show any? <*agentset*>**
  - count such agents
    - **show count <*agentset*>**
- example: remove the richest turtle (with the maximum “assets” value)
  - **ask max-one-of turtles [** sum assets **] [die]**
- Memorizing an *agentset* in a variable
  - **globals [g]**
  - **set g turtle-set turtles**



Introduction to NetLogo (VI):

Programming Concepts - Breeds

- Breed - a “natural” kind of *agentset*
  - Different breeds can behave differently
    - breed [wolves wolf]
    - breed [sheep a-sheep]
    - breed [mice mouse]
    - mice-own [cheese]
- A new breed comes with automatically derived primitives:
  - **create-<breed>, create-custom-<breed>, <breed>-here, <breed>-at**
- Breed is a turtle variable
  - **ask** turtle 5 **[if breed** = sheep ... **]**
- A turtle agent can change breed
  - **ask** turtle 5 **[set breed** sheep **]**



Introduction to NetLogo (VI):

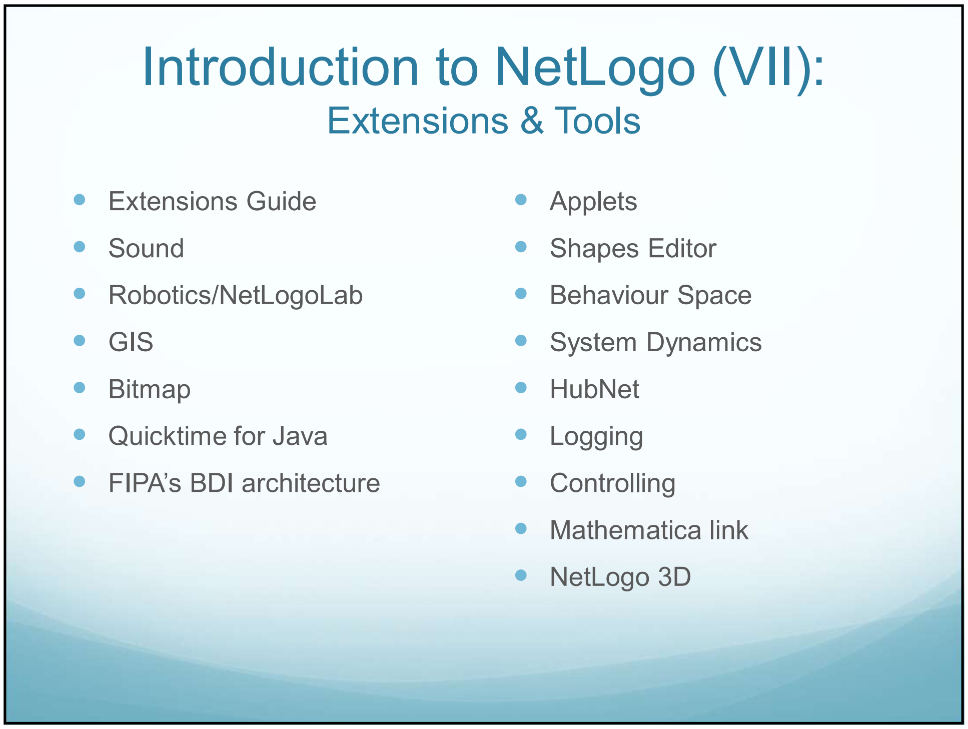
Programming Concepts - Synchronization

- Agents run in parallel (each agent is an independent thread)
  - asynchronous commands:
    - **ask turtles [fd random** 10
      do-something**]**

René Doursat, 2008

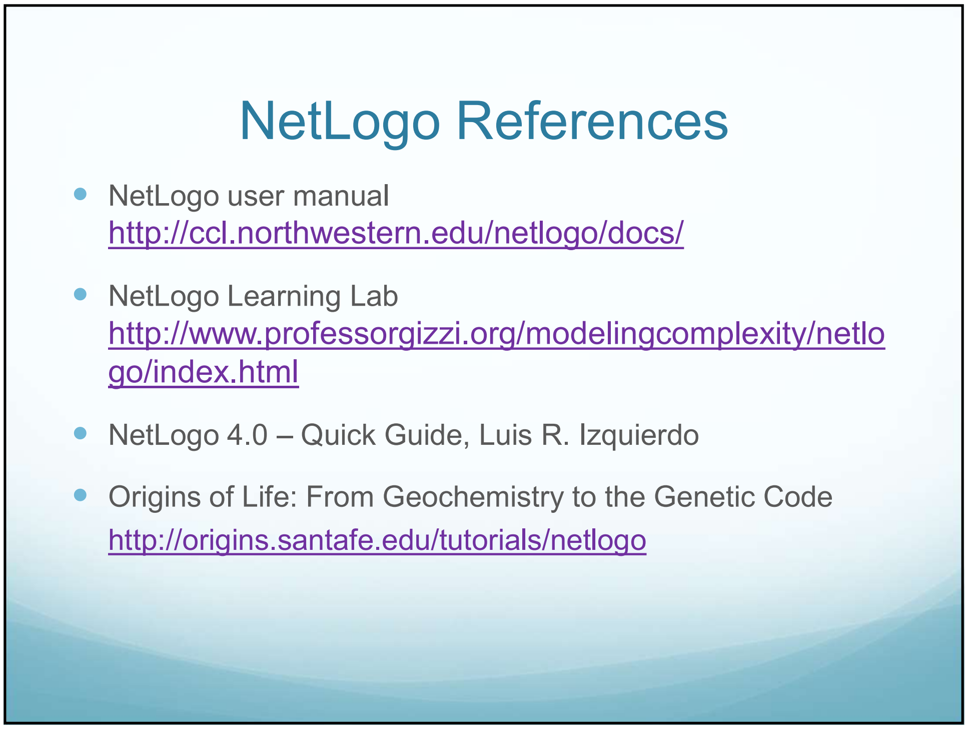
- Agent threads wait and “join” at the end of a block
  - synchronous commands:
    - **ask turtles [fd random** 10 **]**
    - **ask turtles [** do-something **]**

René Doursat, 2008



Introduction to NetLogo (VII):
Extensions & Tools

- Extensions Guide
- Sound
- Robotics/NetLogoLab
- GIS
- Bitmap
- Quicktime for Java
- FIPA’s BDI architecture
- Applets
- Shapes Editor
- Behaviour Space
- System Dynamics
- HubNet
- Logging
- Controlling
- Mathematica link
- NetLogo 3D



NetLogo References

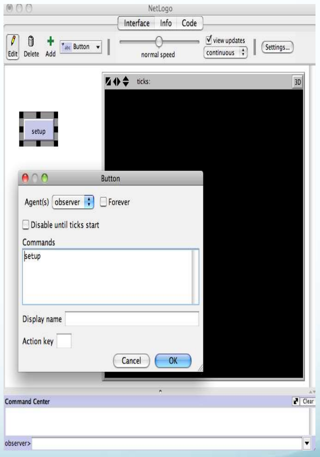
- NetLogo user manual
  http://ccl.northwestern.edu/netlogo/docs/
- NetLogo Learning Lab
  http://www.professorgizzi.org/modelingcomplexity/netlogo/index.html
- NetLogo 4.0 – Quick Guide, Luis R. Izquierdo
- Origins of Life: From Geochemistry to the Genetic Code
  http://origins.santafe.edu/tutorials/netlogo

A simple tutorial

- Create via “File/New”, a new NetLogo program
- Save it, via “File/Save as” with the name *MushroomHunt.nlogo*
- From the “Settings” button
  - view of the World’s geometry
- To initialize the World and run the model
  - setup procedure
  - go procedure

1

- “Interface” tab -> “Button”
- create *setup* button
- similarly create a *go* button



☑ view updates
continuous

Button

Agent(s) observer □ Forever

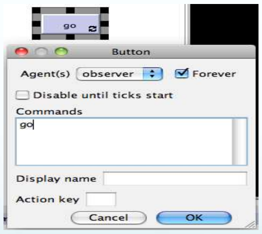
□ Disable until ticks start

Commands

```
setup
```

Display name

Action key



Button

Agent(s) observer ☑ Forever

□ Disable until ticks start

Commands

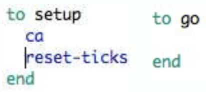
```
go
```

Display name

Action key

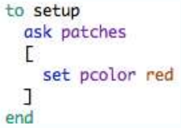
2

- In “Code” tab
  - Create the skeleton of setup & go



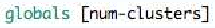
```
to setup          to go
  ca
  reset-ticks     end
end
```

  - Change setup to



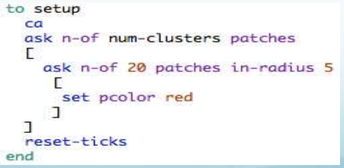
```
to setup
  ask patches
  [
    set pcolor red
  ]
end
```

  - Create the clusters of mushrooms (patches).
    - The cluster can be a model parameter
    - Define a global variable *num-clusters*



```
globals [num-clusters]
```

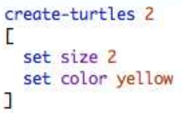
  - Modify the setup to turn in red randomly a “num-cluster” patches



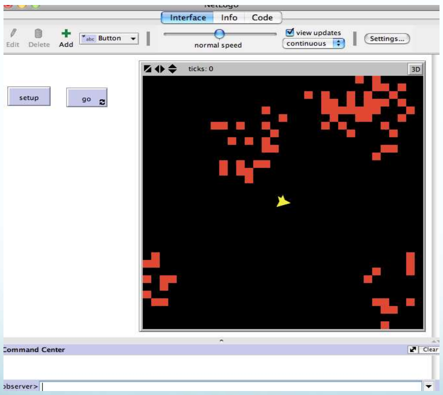
```
to setup
  ca
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
    [
      set pcolor red
    ]
  ]
  reset-ticks
end
```

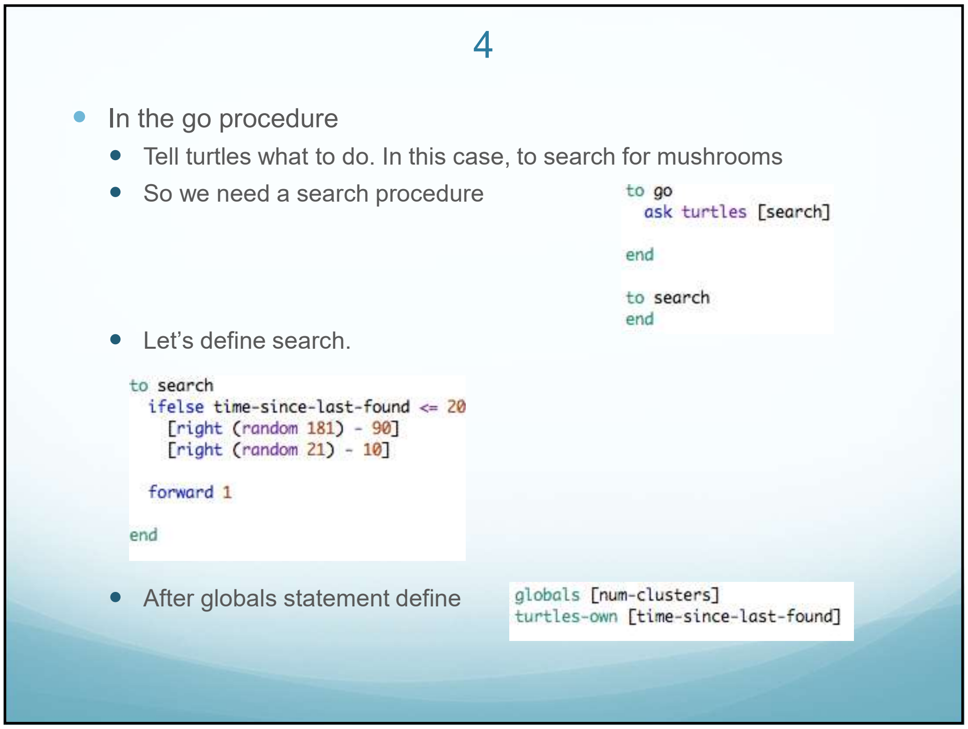
3

- create the turtles
  - use the primitive create-turtles



```
create-turtles 2
[
  set size 2
  set color yellow
]
```





4

- In the go procedure
  - Tell turtles what to do. In this case, to search for mushrooms
  - So we need a search procedure

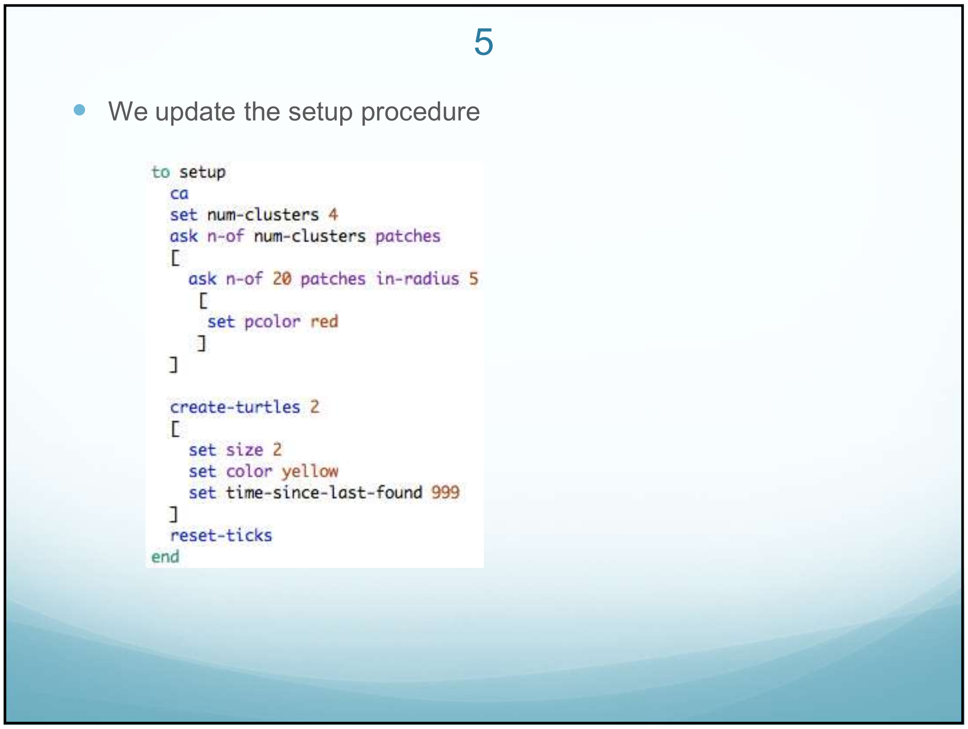
```
to go
  ask turtles [search]

end

to search
end
```

  - Let's define search.

```
to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

end
```

  - After globals statement define

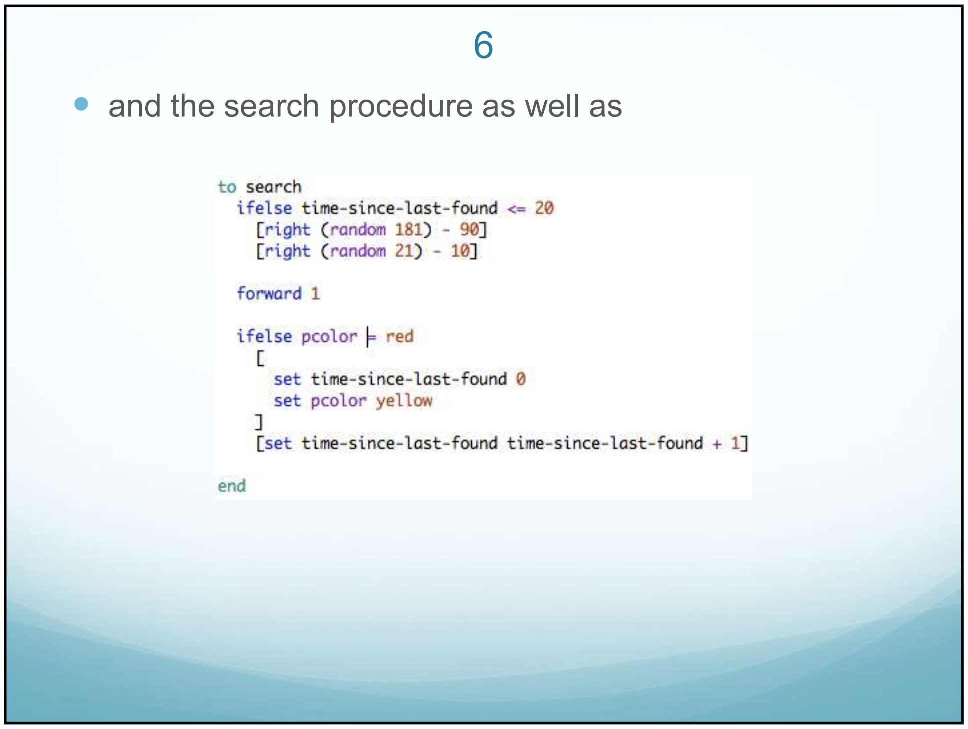
```
globals [num-clusters]
turtles-own [time-since-last-found]
```



5

- We update the setup procedure

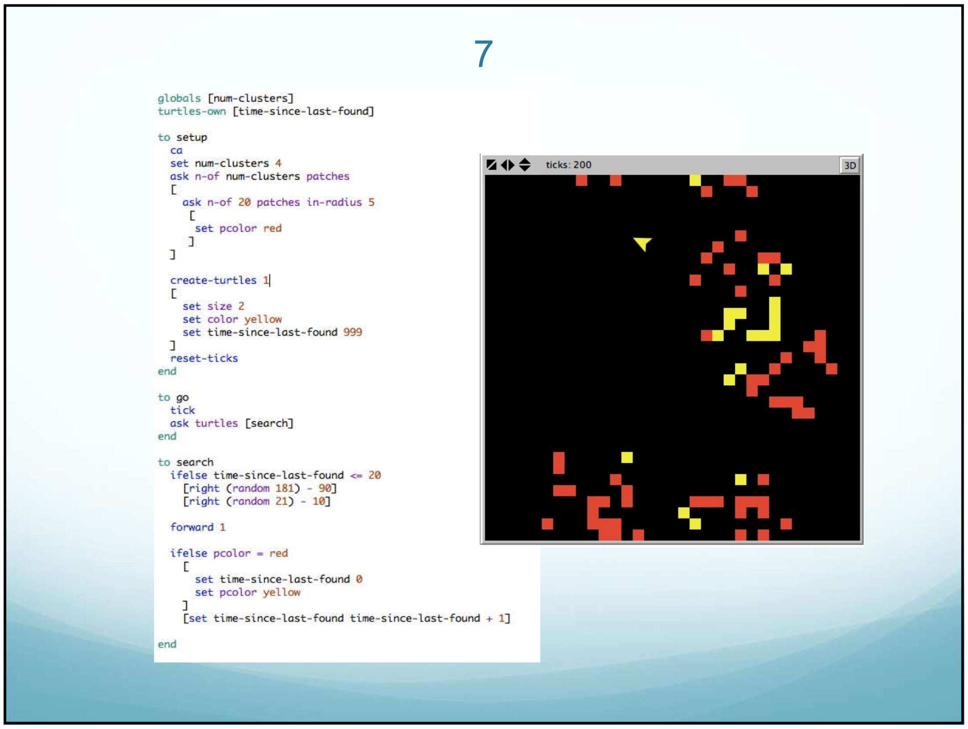
```
to setup
  ca
  set num-clusters 4
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
     [
      set pcolor red
     ]
  ]

  create-turtles 2
  [
    set size 2
    set color yellow
    set time-since-last-found 999
  ]
  reset-ticks
end
```



6

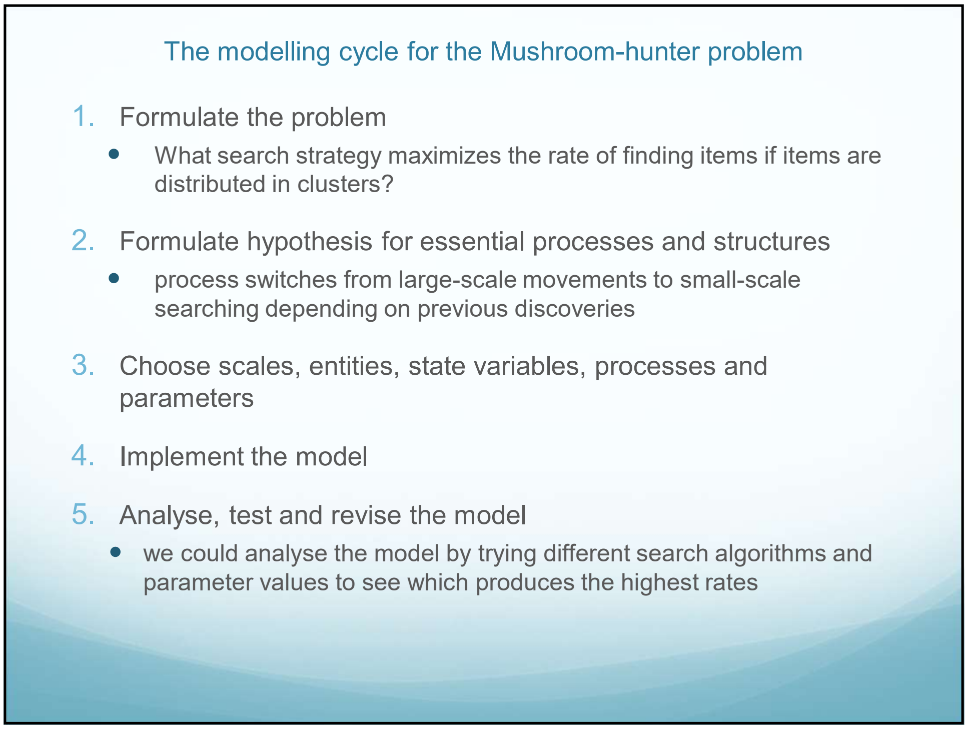
- and the search procedure as well as

```
to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

  ifelse pcolor = red
    [
      set time-since-last-found 0
      set pcolor yellow
    ]
    [set time-since-last-found time-since-last-found + 1]

end
```



7

```
globals [num-clusters]
turtles-own [time-since-last-found]

to setup
  ca
  set num-clusters 4
  ask n-of num-clusters patches
  [
    ask n-of 20 patches in-radius 5
     [
      set pcolor red
     ]
  ]

  create-turtles 1
  [
    set size 2
    set color yellow
    set time-since-last-found 999
  ]
  reset-ticks
end

to go
  tick
  ask turtles [search]
end

to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]

  forward 1

  ifelse pcolor = red
    [
      set time-since-last-found 0
      set pcolor yellow
    ]
    [set time-since-last-found time-since-last-found + 1]

end
```



The modelling cycle for the Mushroom-hunter problem

1. Formulate the problem
   - What search strategy maximizes the rate of finding items if items are distributed in clusters?
2. Formulate hypothesis for essential processes and structures
   - process switches from large-scale movements to small-scale searching depending on previous discoveries
3. Choose scales, entities, state variables, processes and parameters
4. Implement the model
5. Analyse, test and revise the model
   - we could analyse the model by trying different search algorithms and parameter values to see which produces the highest rates