MATHMOD VIENNA

VIENNA CONFERENCE ON MATHEMATICAL MODELLING

An Introduction to NetLogo

given by *Gabriel Wurzer**,

**modified by Todd BenDor, 2016*

www.iemar.tuwien.ac.at

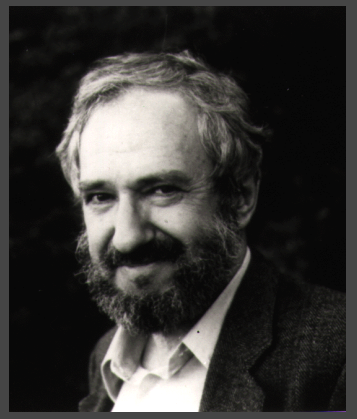
digital architecture and planning (259.1) iemar

institute of architectural sciences
vienna university of technology

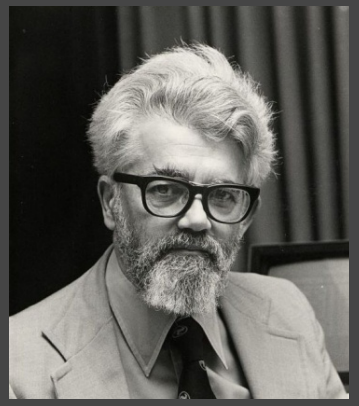
Netlogo

free* agent-based simulation environment
by Uri Wilensky, Northwestern University,
which is
based on programming language
"Logo" by Seymour Papert, MIT
which is
based on programming language
"Lisp" by John McCarthy, Stanford

Wilensky
1999



Papert
1968



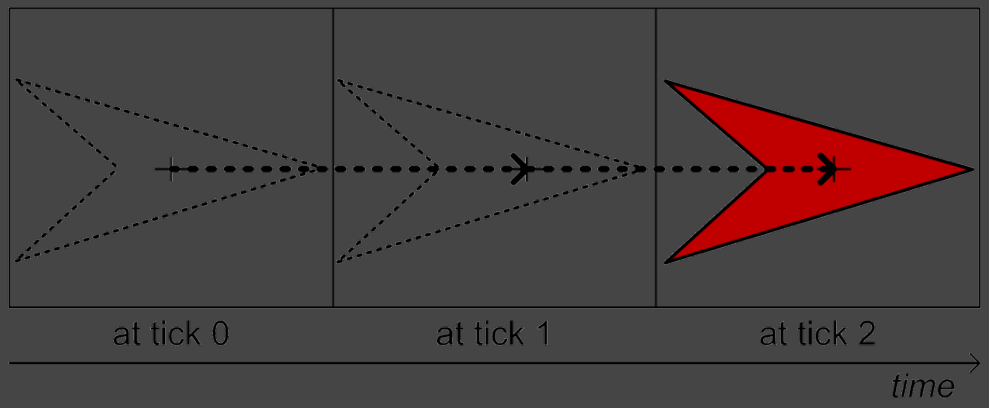
McCarthy
1958

* http://ccl.northwestern.edu/netlogo/

Netlogo is a discrete simulation

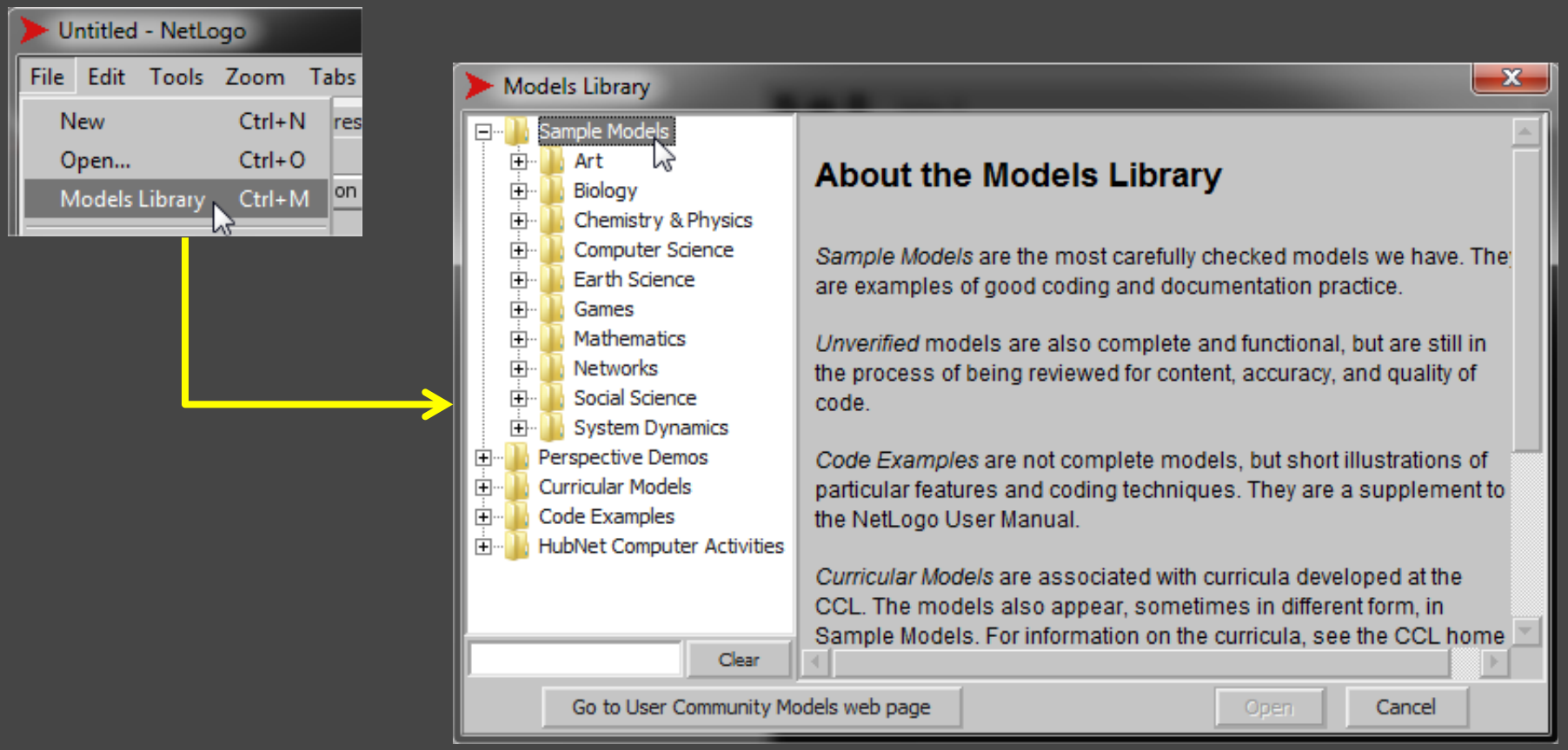
Simulation environment with discretized world ("patches"),

on which agents ("turtles") perform actions in discrete time steps ("ticks")



Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 3, © 2012, Technical University Vienna.

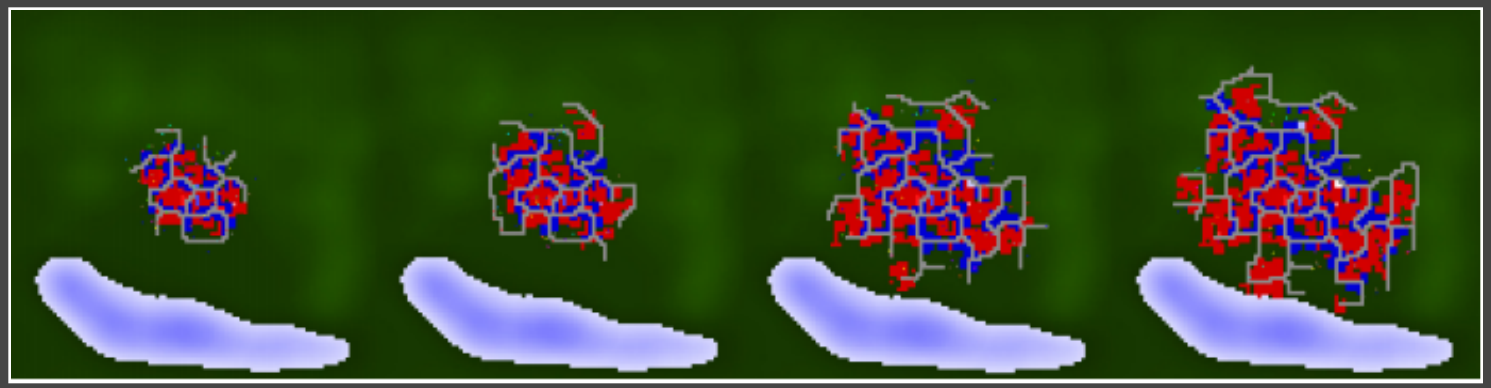
What NetLogo is used for…



Example: urban planning

Procedural City Modeling (Lechner et al. 2003)

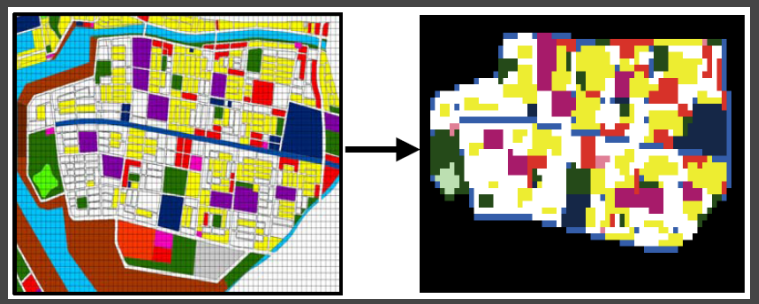
http://ccl.northwestern.edu/papers/ProceduralCityMod.pdf



Picture courtesy of Lechner, Watson, Wilensky and Felsen,

Simulation of Urban Land Development and Land Use (Tsai-chu and Bo-yi 2010)

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5421277



Picture courtesy of Tsai-chu and Bo-yi

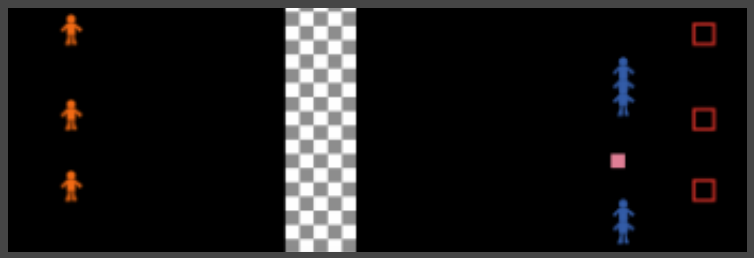
Example: hospital planning

An Agent Based Simulation Tool for Scheduling Emergency Department Physicians (Jones and Evans, 2008)

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656074/

Computer Terminal Placement and Workflow in an Emergency Department: An Agent-based Model (Poynton et al. 2007)

http://tuvalu.santafe.edu/events/workshops/index.php/CSSS_2007_Santa_Fe-Final_Papers



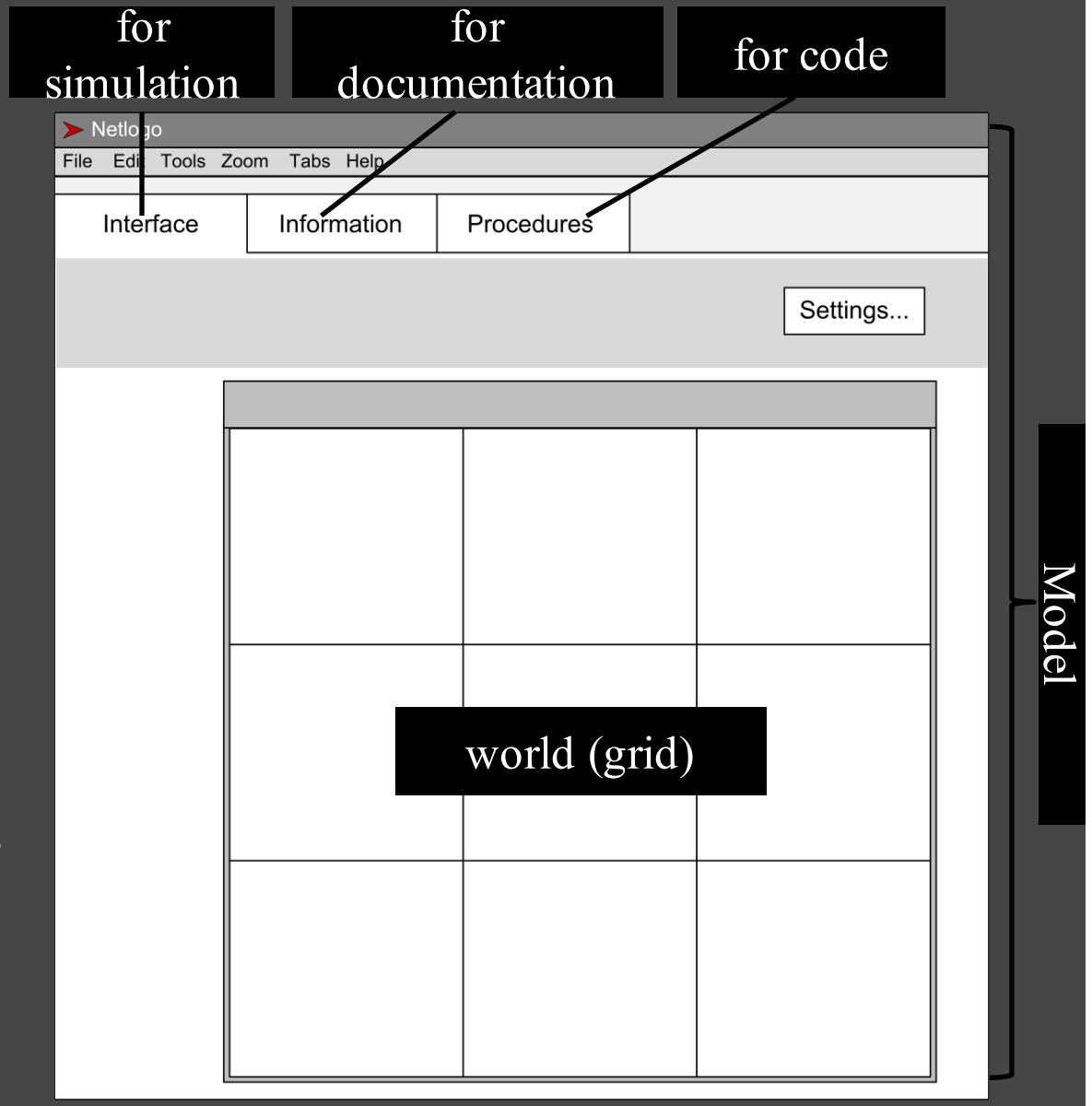
Picture courtesy of Poynton, Shah, BeLue, Mazzotta, Beil and Habibullah

THE NETLOGO ENVIRONMENT

Main screen of a model

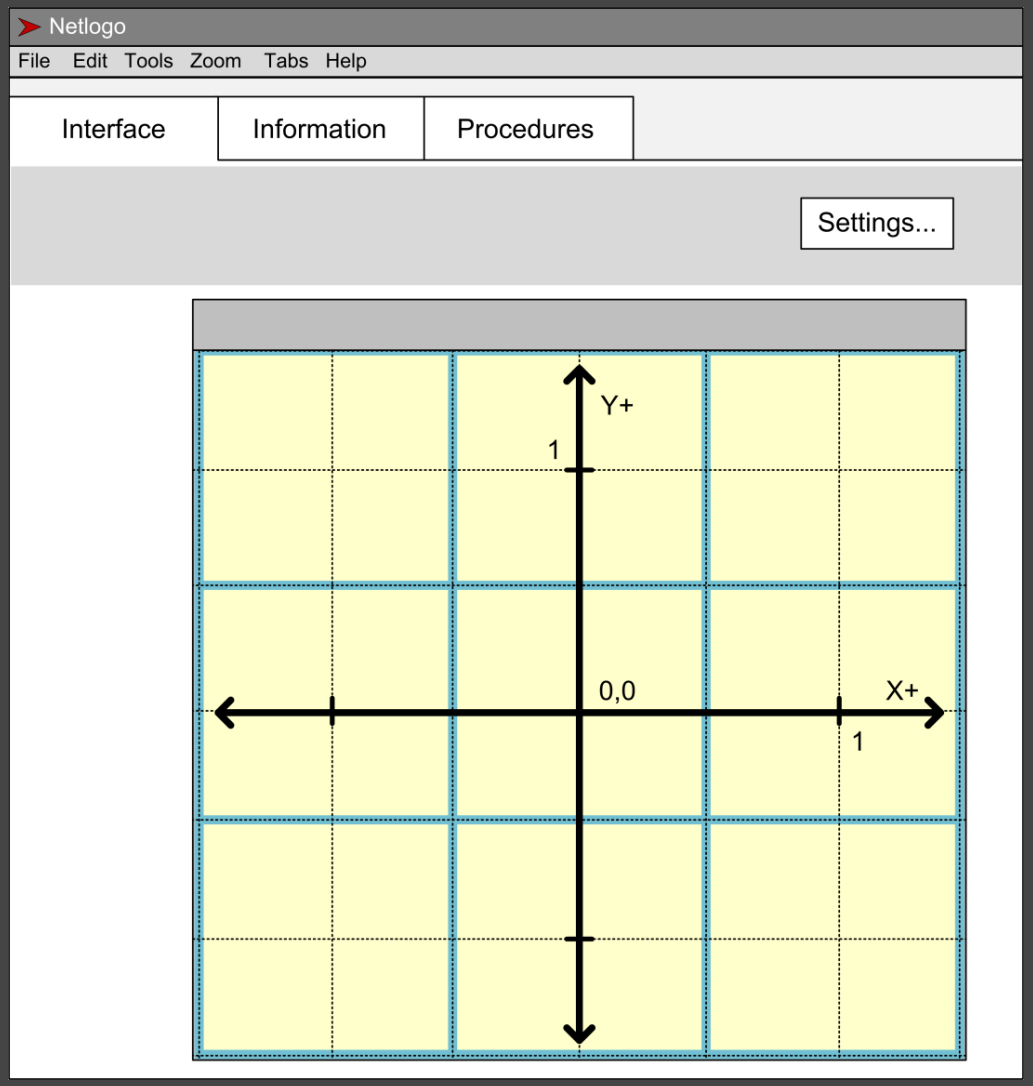
- simulation performed in interface area
- documentation area lists what to do with the model
- programming is done in the procedures area

world discretized into grid, visible in center of screen



Co-ordinate space

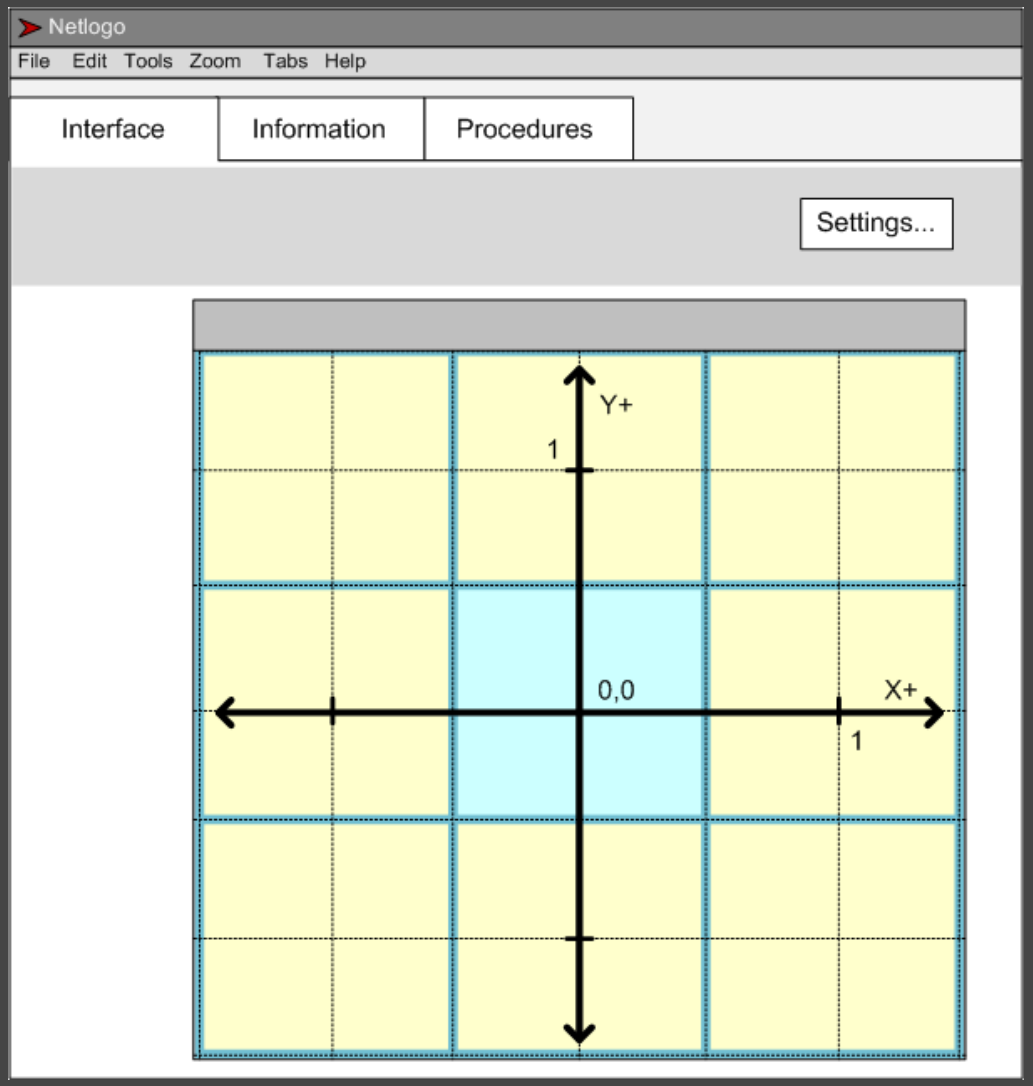
- origin (0,0) in middle of grid
- X+ right, Y+ up
- world composed of grid cells ("patches")
- each patch is identified by the coordinate at its center,



Co-ordinate space

- origin (0,0) in middle of grid
- X+ right, Y+ up
- world composed of grid cells ("patches")
- each patch is identified by the coordinate at its center,

  e.g. patch 0 0 at origin

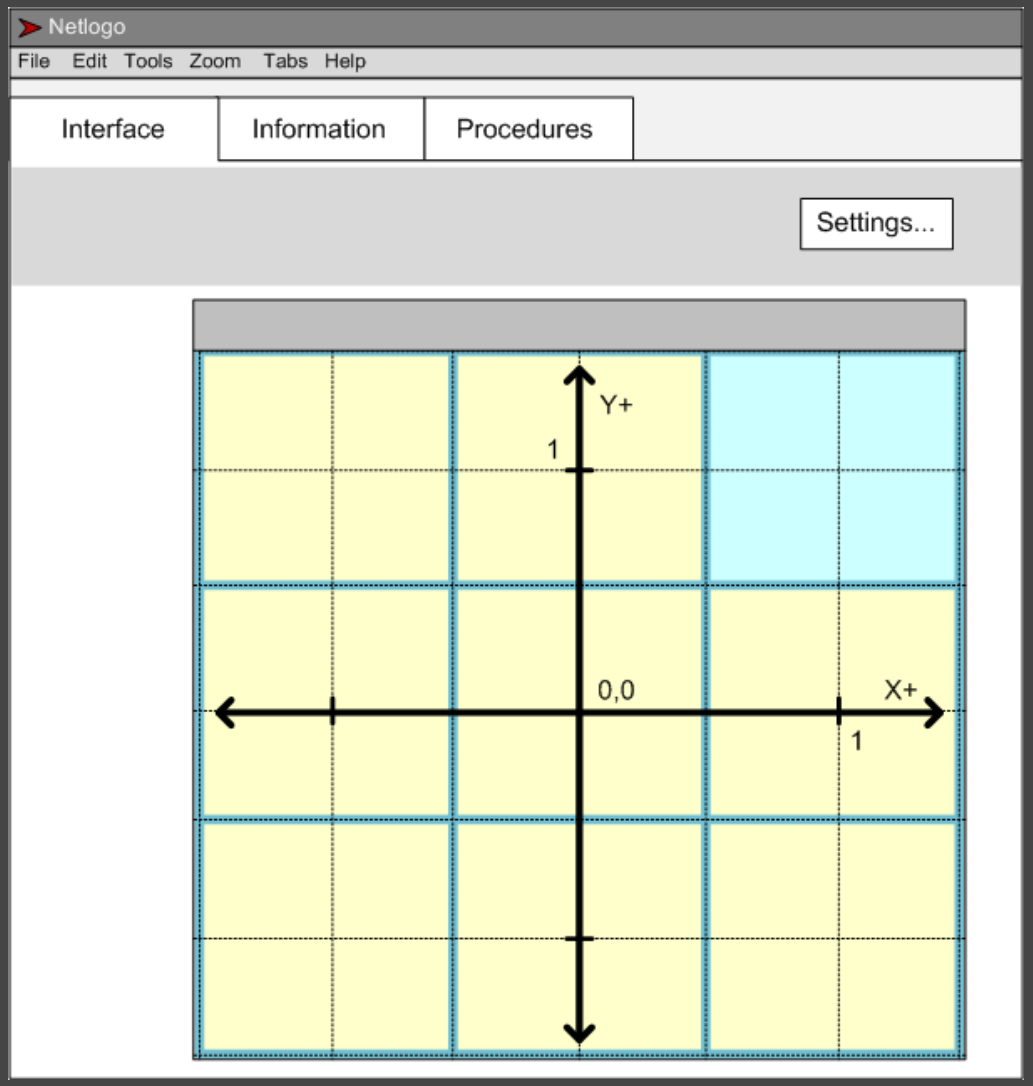


Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 10, © 2012, Technical University Vienna.

Co-ordinate space

- origin (0,0) in middle of grid
- X+ right, Y+ up
- world composed of grid cells ("patches")
- each patch is identified by the coordinate at its center,

  e.g. patch 0 0 at origin
  patch 1 1 elsewhere

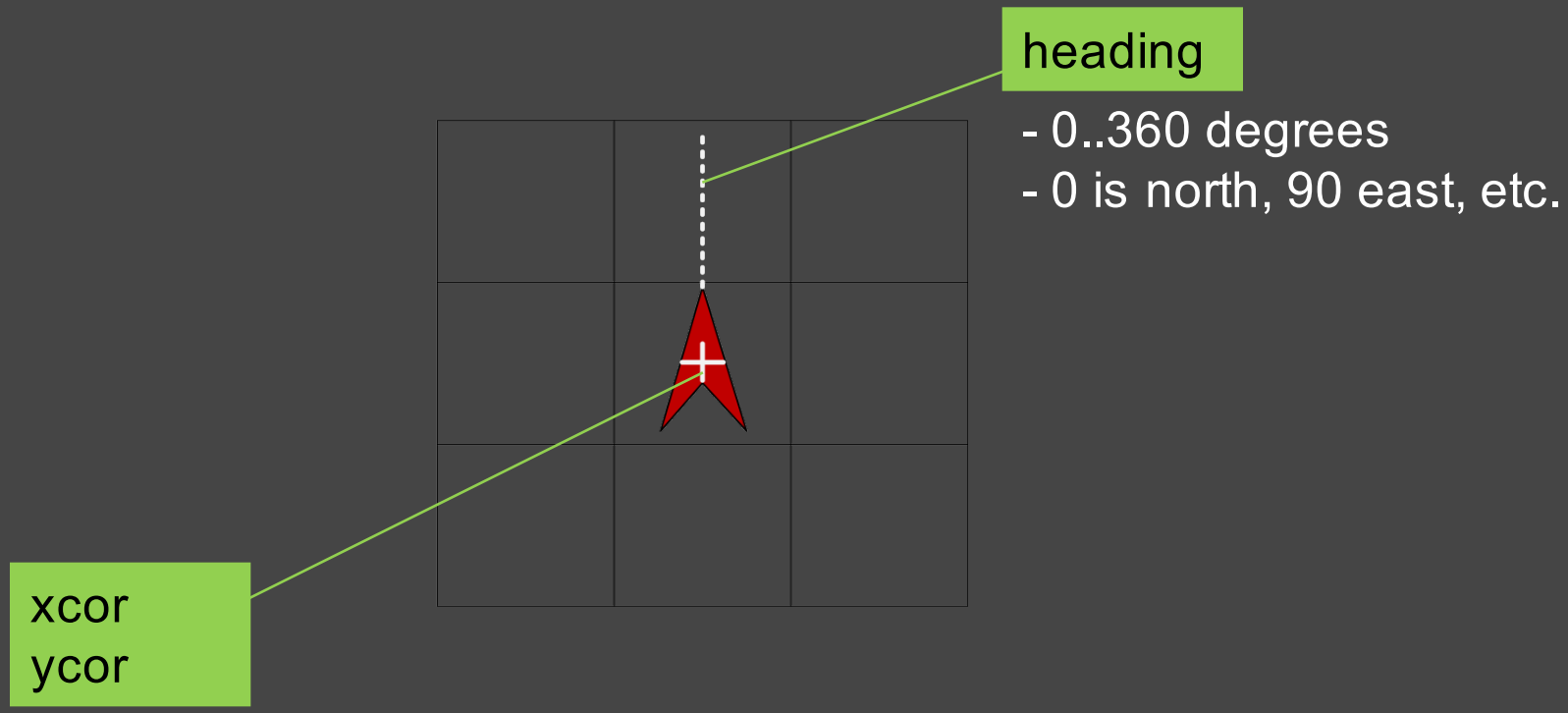


Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 11, © 2012, Technical University Vienna.

PROPERTIES AND ABILITIES OF TURTLES

Turtles are...

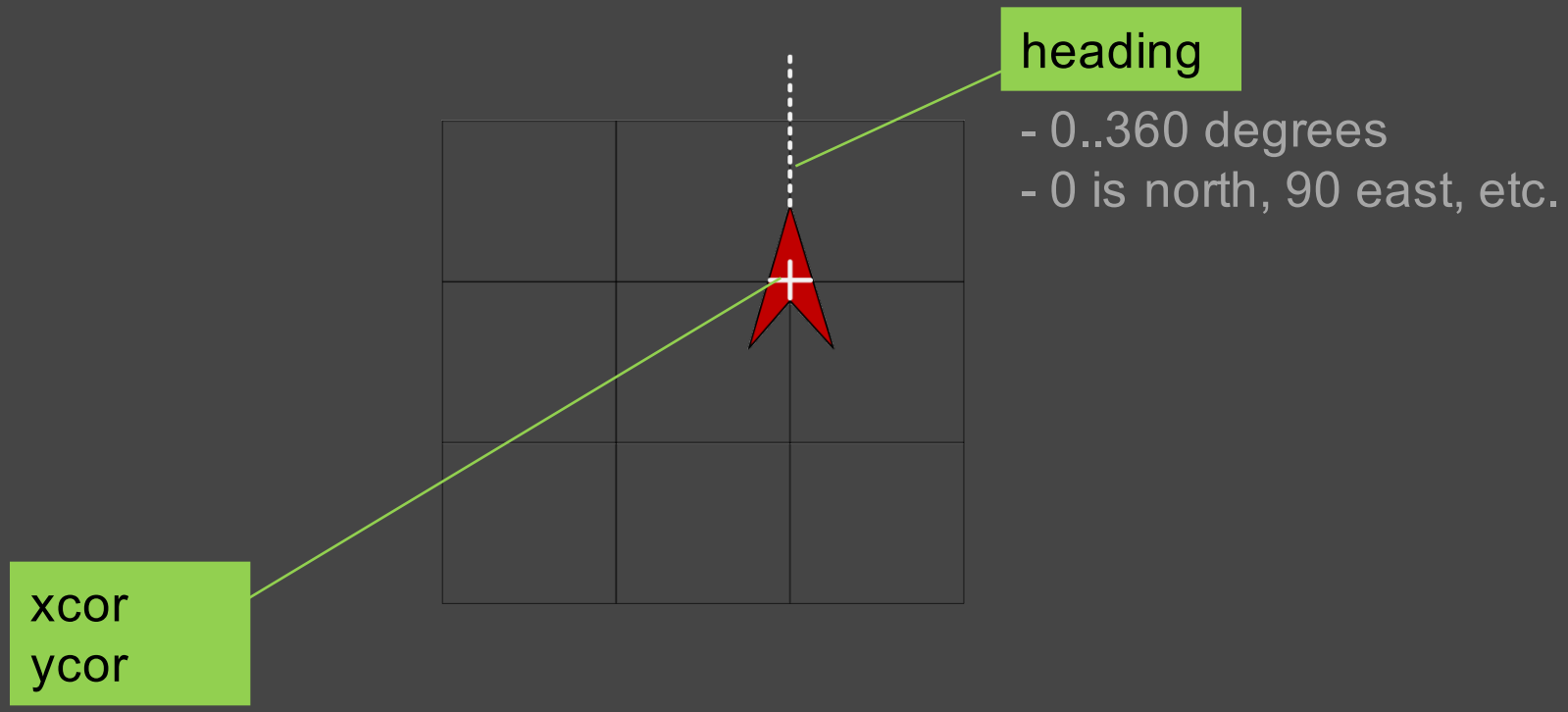
- movable entities within the netlogo world



Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 13, © 2012, Technical University Vienna.

Turtles are...

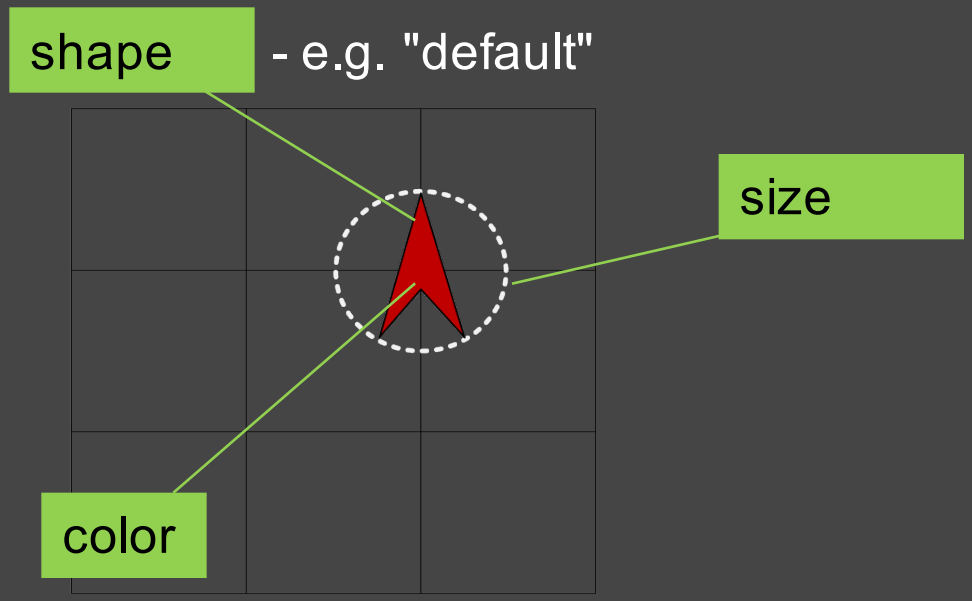
- movable entities within the netlogo world



Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 14, © 2012, Technical University Vienna.

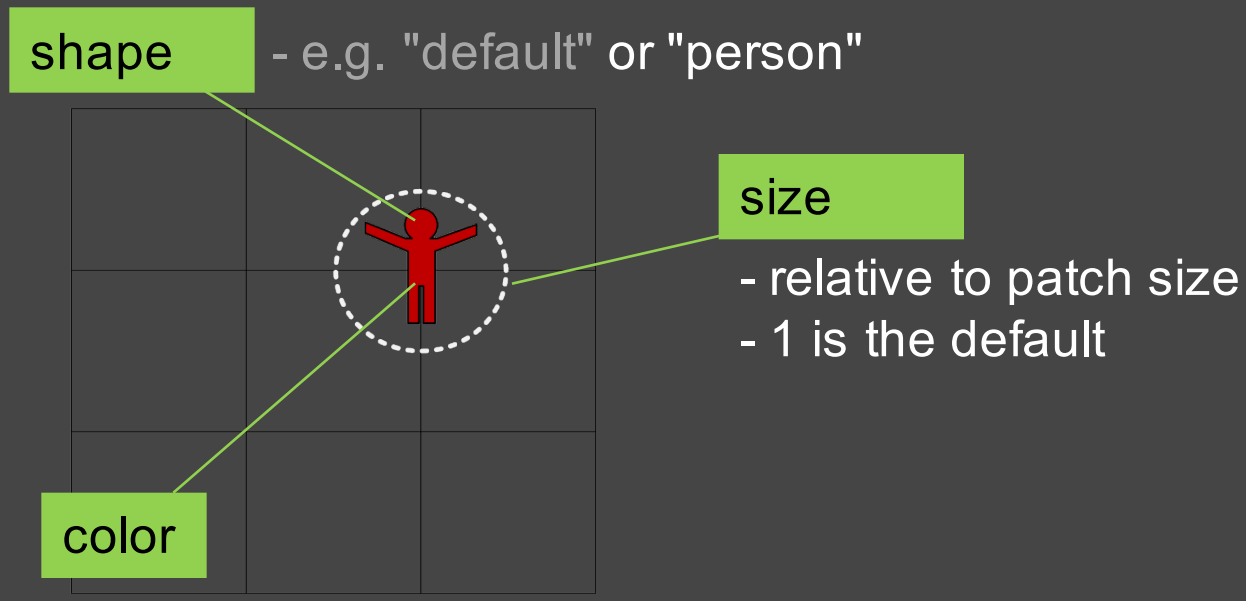
Turtles are...

- taking form (they represent an active, animated entity)



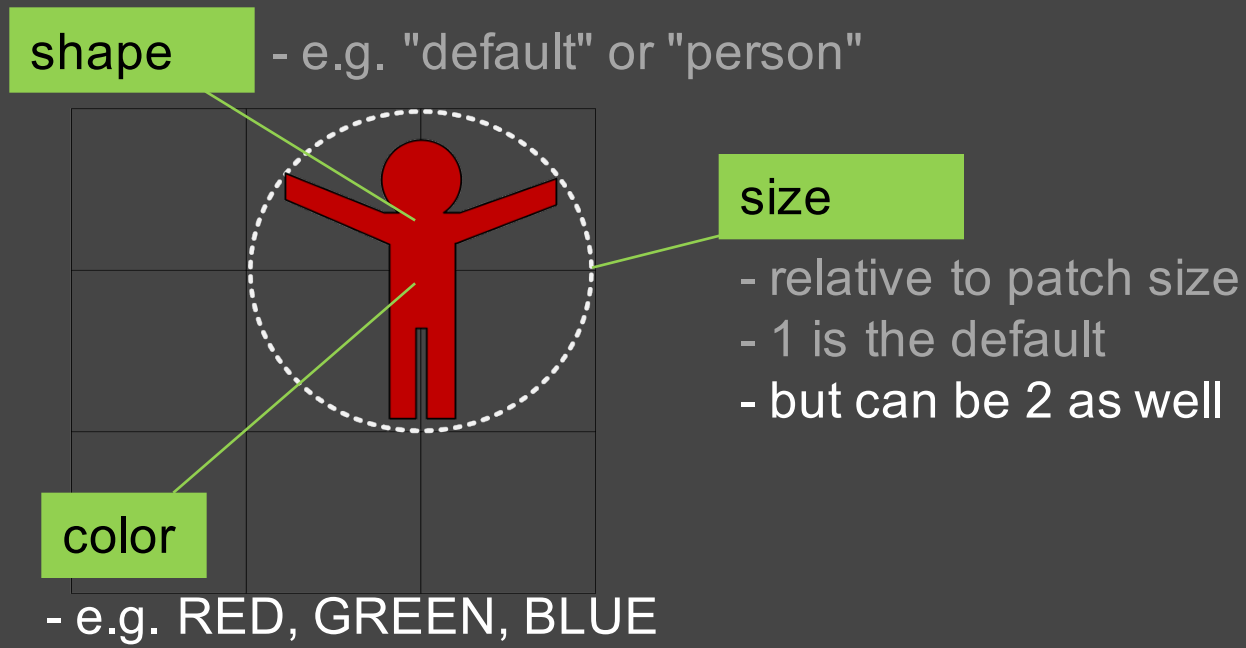
Turtles are...

- taking form (they represent an active, animated entity)



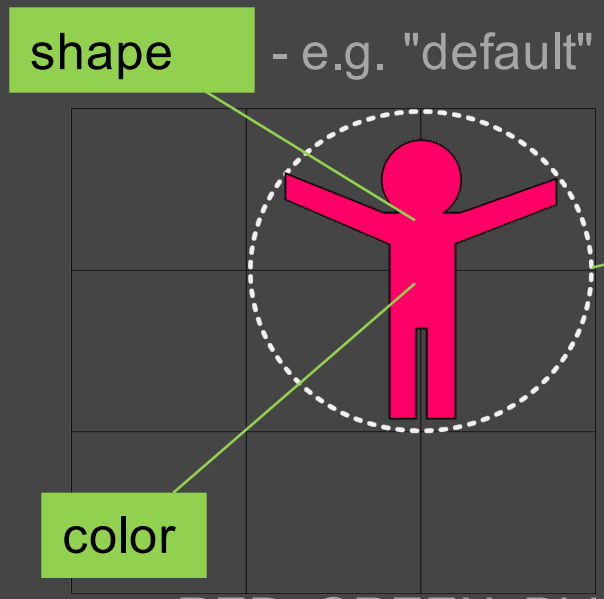
Turtles are...

- taking form (they represent an active, animated entity)



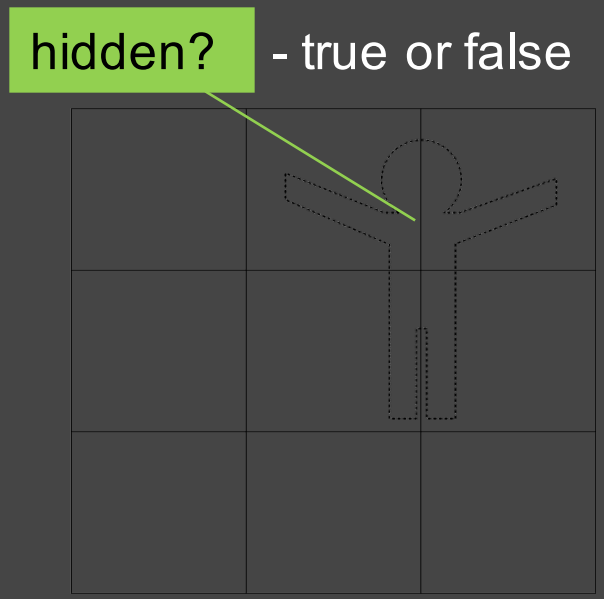
Turtles are...

- taking form (they represent an active, animated entity)



Turtles are...

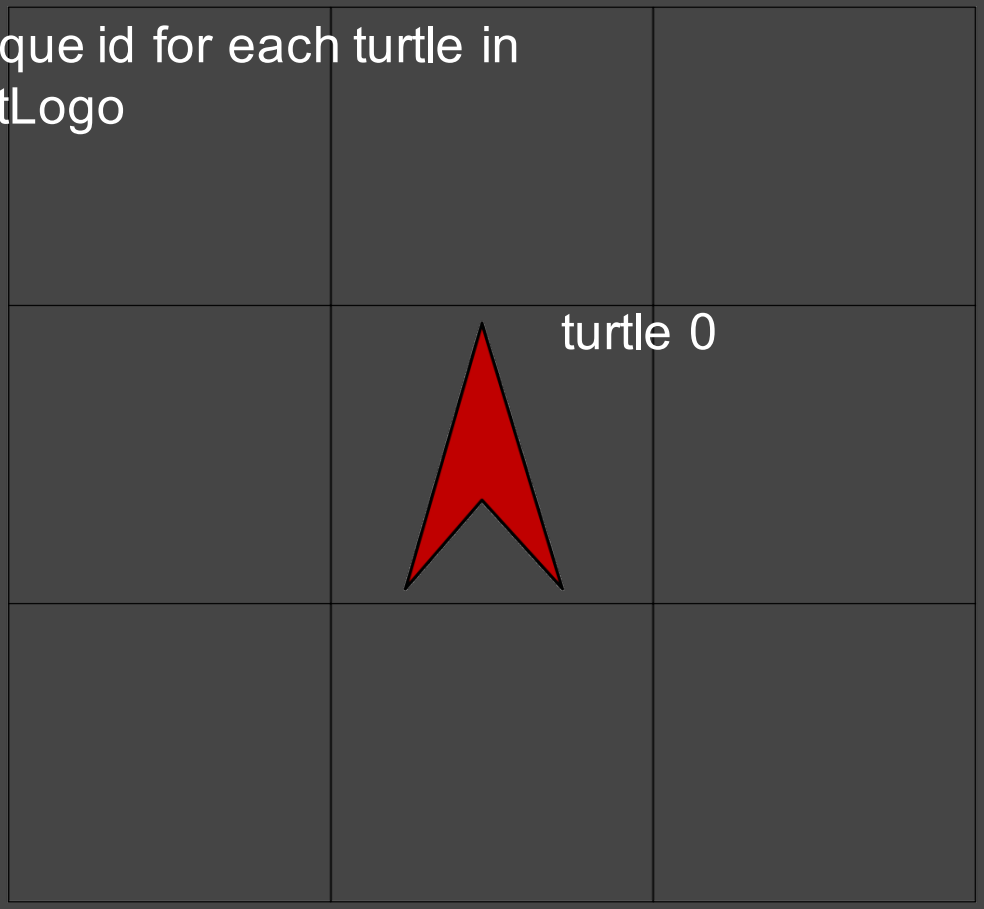
- by default visible, but can be hidden as well



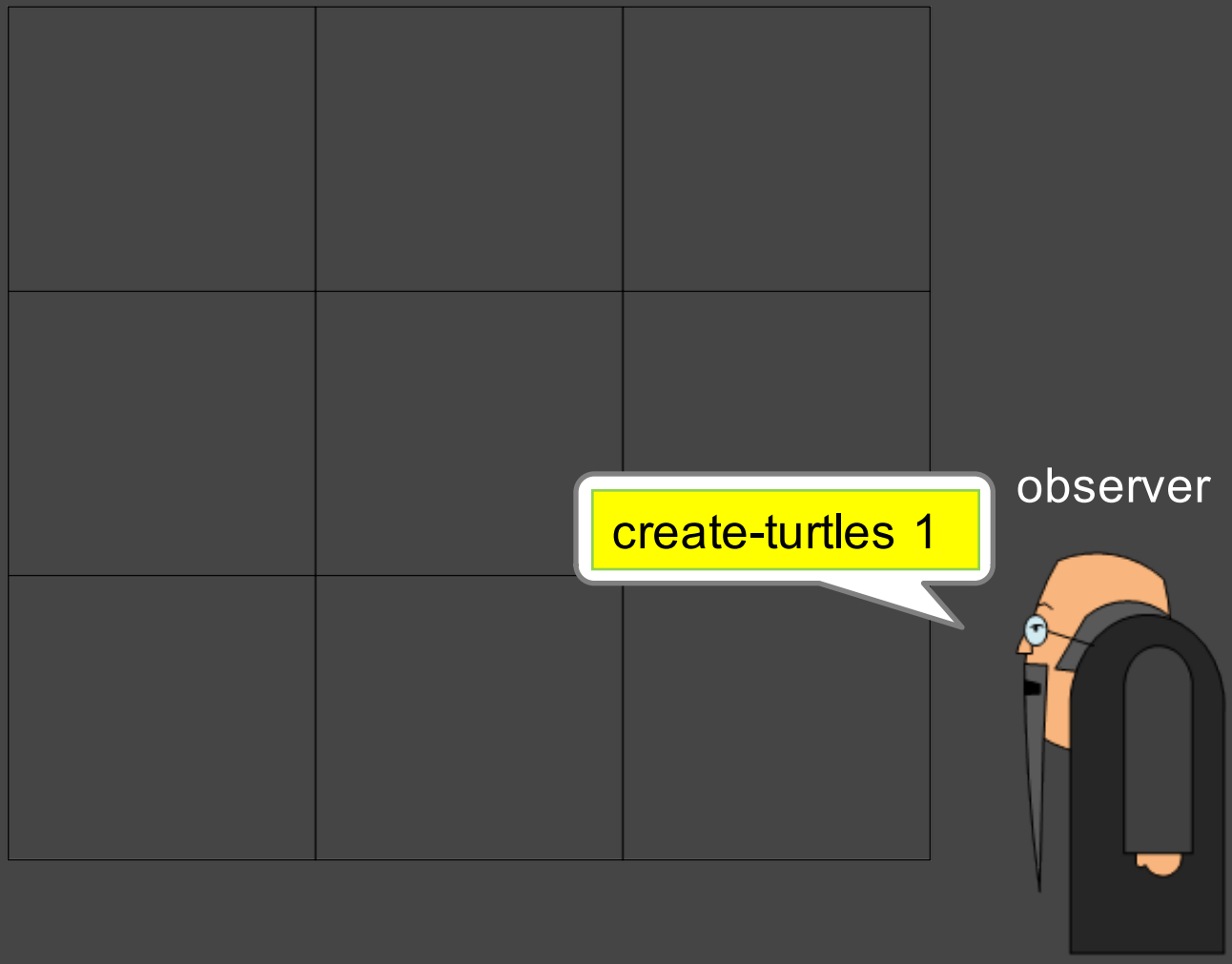
Properties

- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?

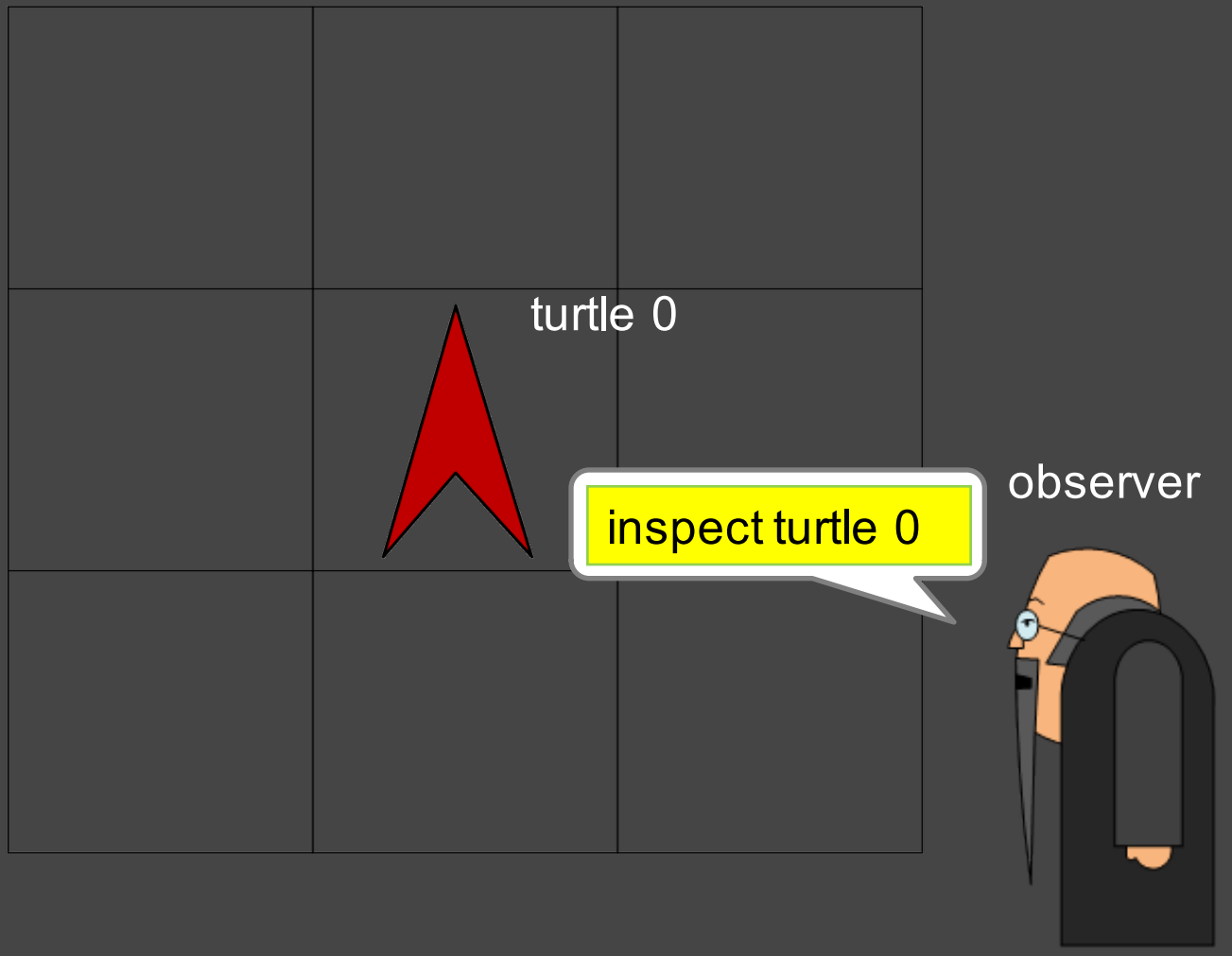
- unique id for each turtle in NetLogo



Commands

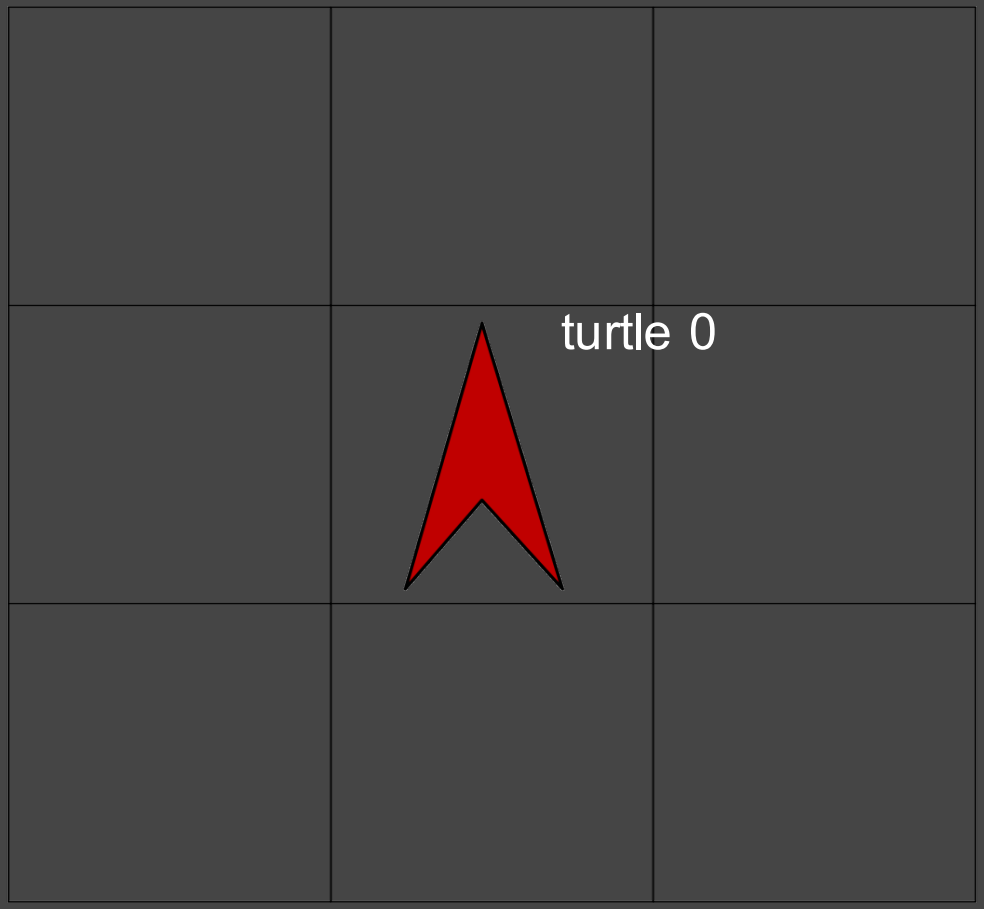


Commands



Commands

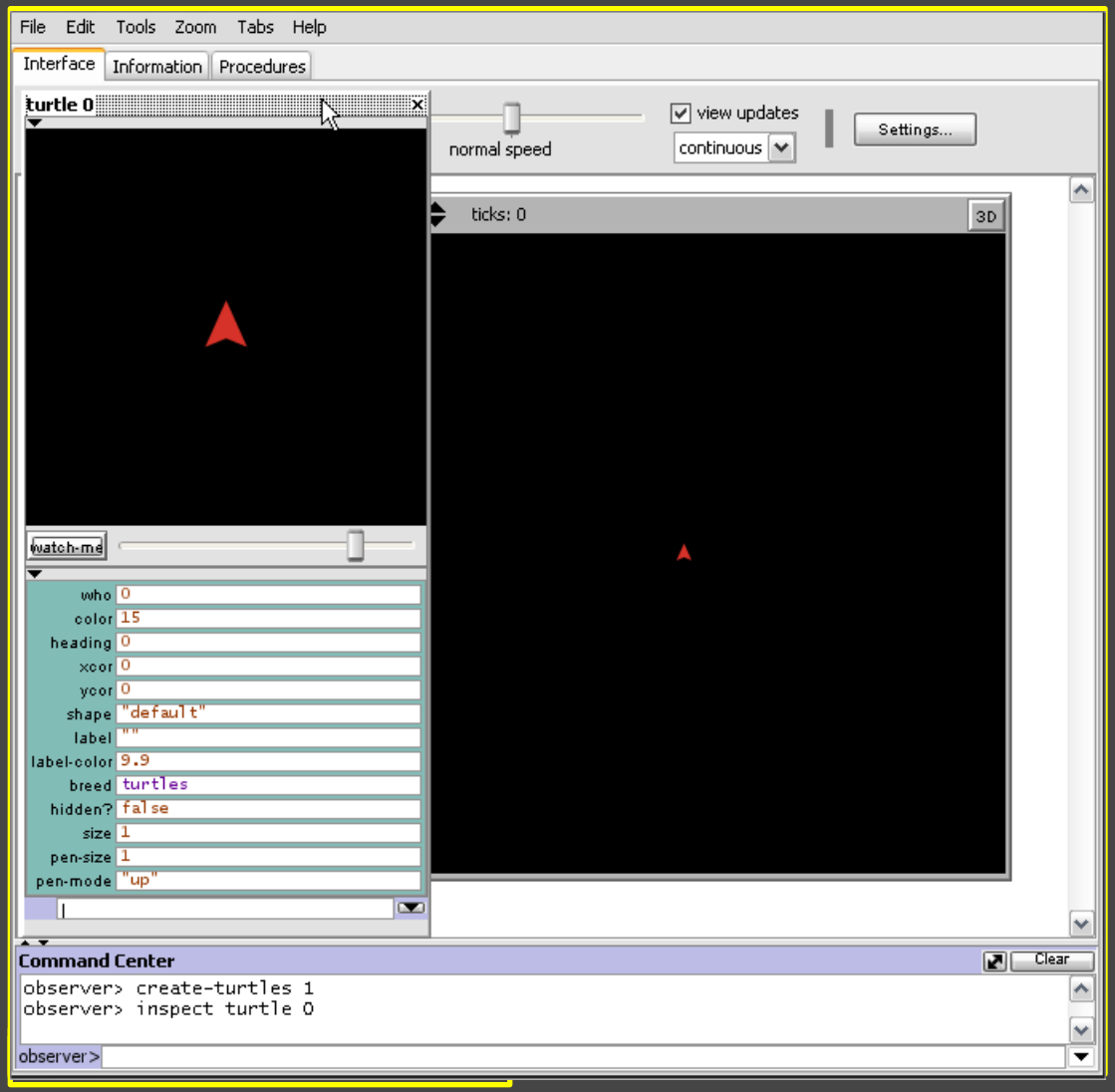
- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?



observer

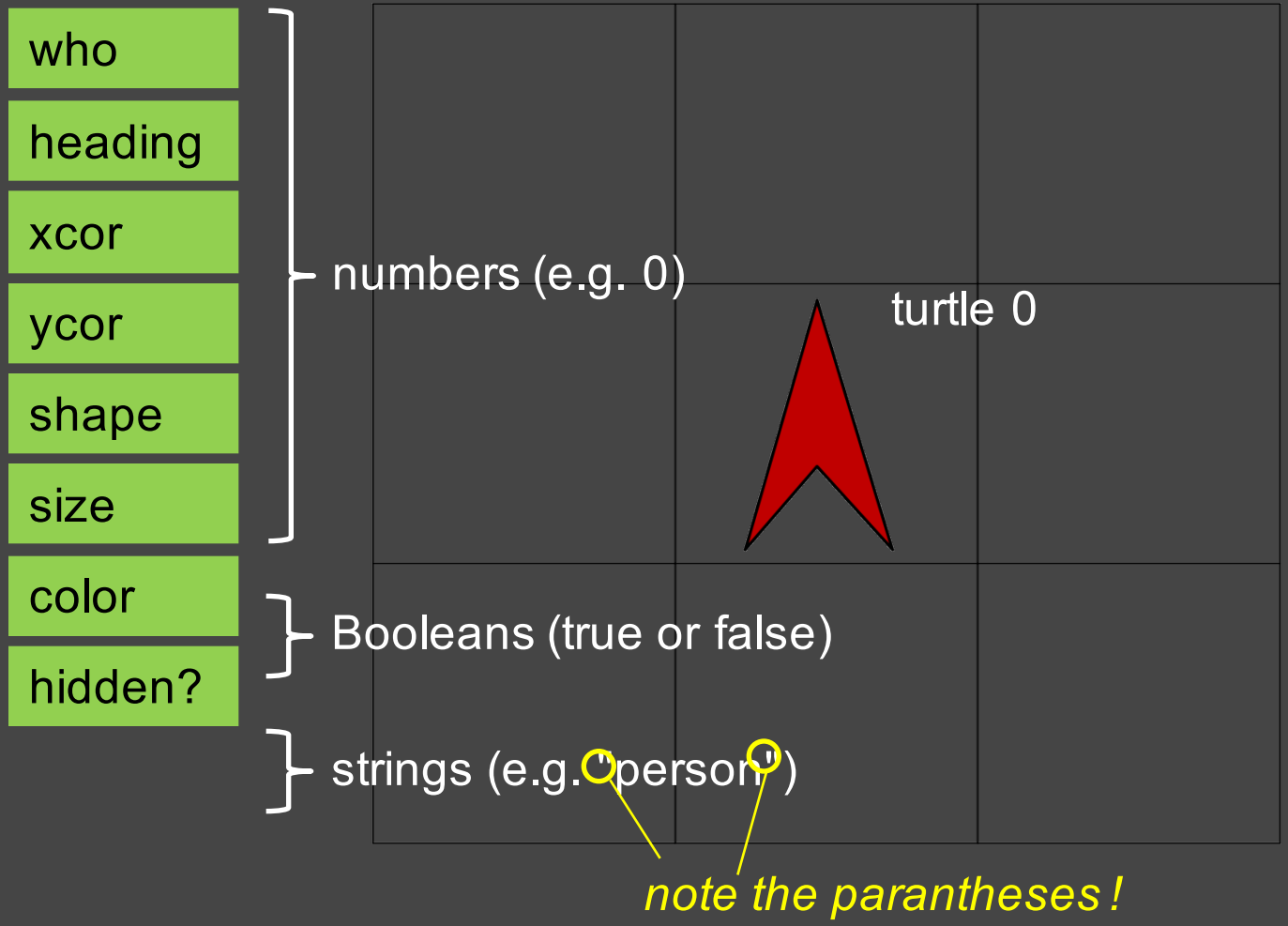
Your turn...

1. Start NetLogo
2. In the observer> input box, enter
   create-turtles 1
3. in the same location, enter
   inspect turtle 0
4. enter RED as color,
   0 as heading
   1 as xcor
   1 as ycor
   "person" as shape



Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 24, © 2012, Technical University Vienna.

A closer look at the inspected properties...



Data types

- Numbers, Booleans and strings are **data types**
- Each data type has its own syntax (e.g. "xyz" for strings)
- Each data type has its own benefits
  - numbers are made for calculations (+, -, /, *, sin, cos, etc.)
  - Booleans are made for conditions (if hidden? ...)
  - strings are made for supplying names (e.g. use the "default" shape)

In detail…

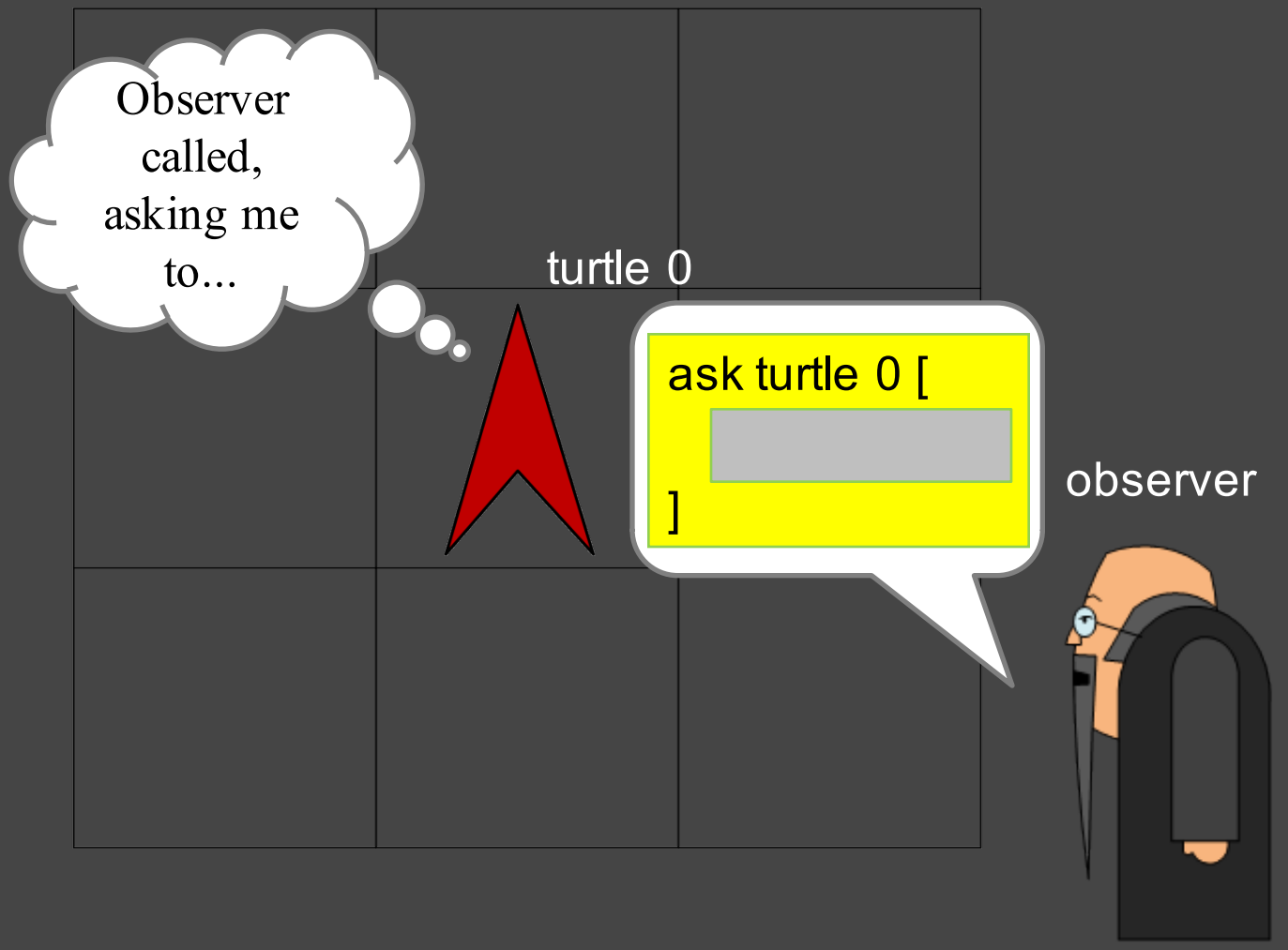
numbers…
ordinal type (1,2,3)
comparison: (1<2)
operators: +,-,*,/

Booleans…
truth type (true, false)
comparison: (true != false)
operators: and, or, not

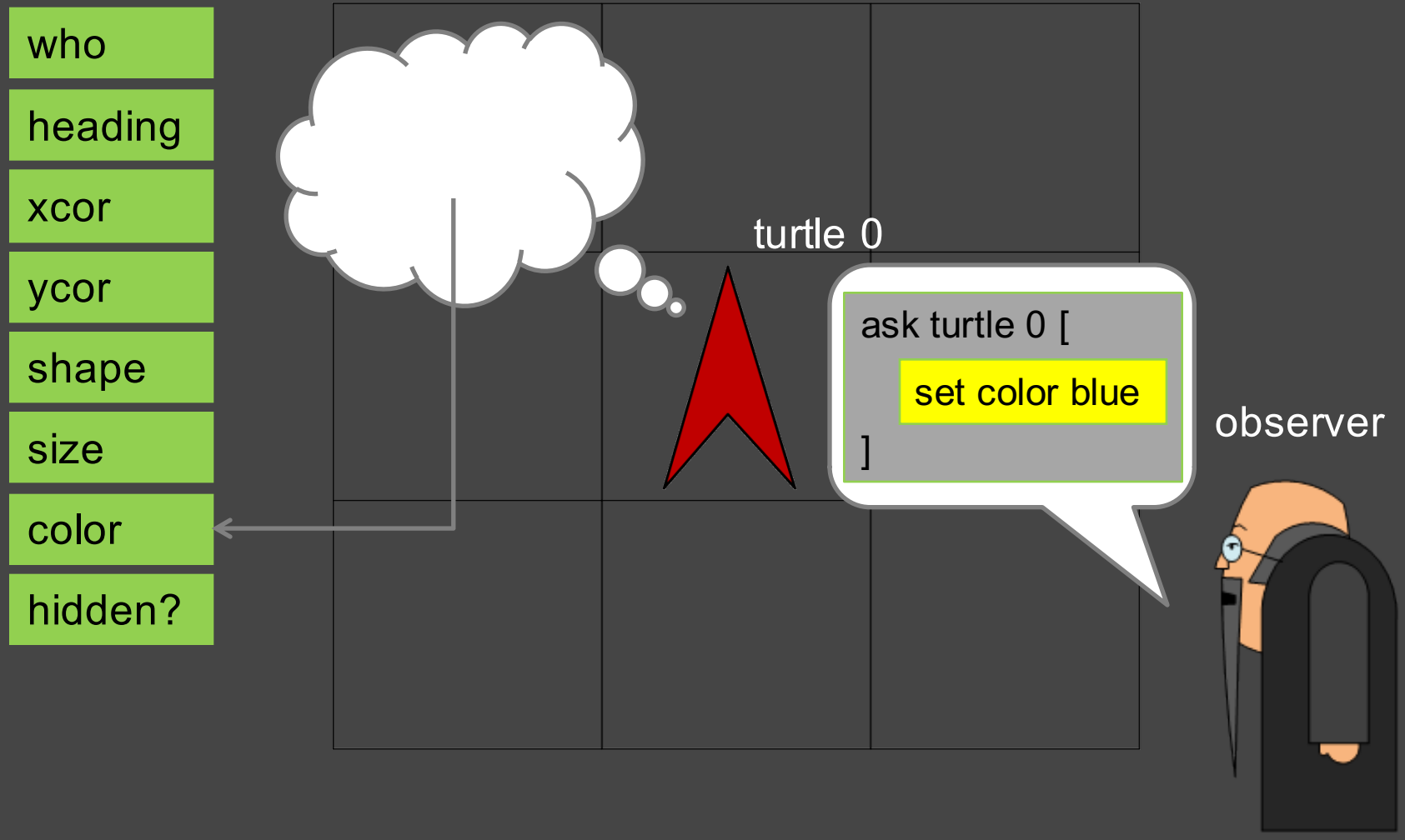
strings…
character chains ("abra")
comparison "test" != "abra"
operators:
concatenation ("abra" + "cadabra" = "abracadabra")
slicing ("abracadabra"[4:6] = "cad")
…

The ask command

- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?

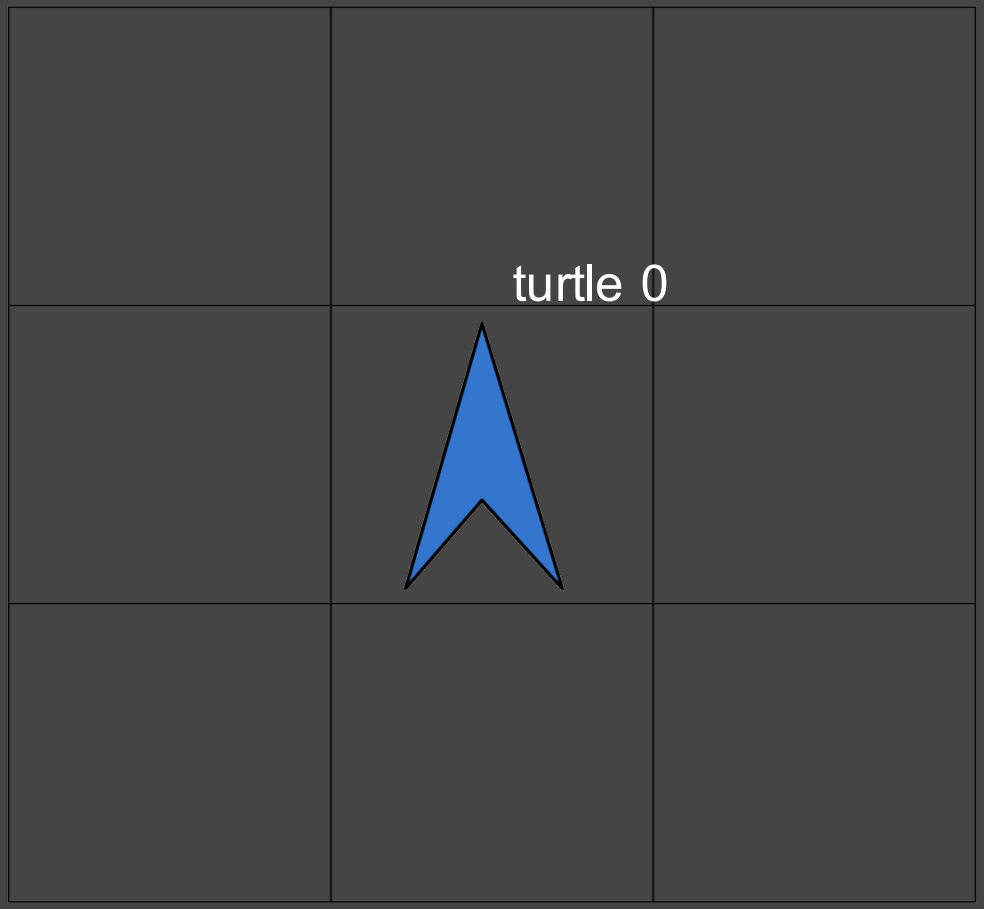


The set command



The set command

- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?



Ask explained

- The ask command calls a set of turtles or patches, passing commands to them
- These commands are supplied in brackes,
  i.e.

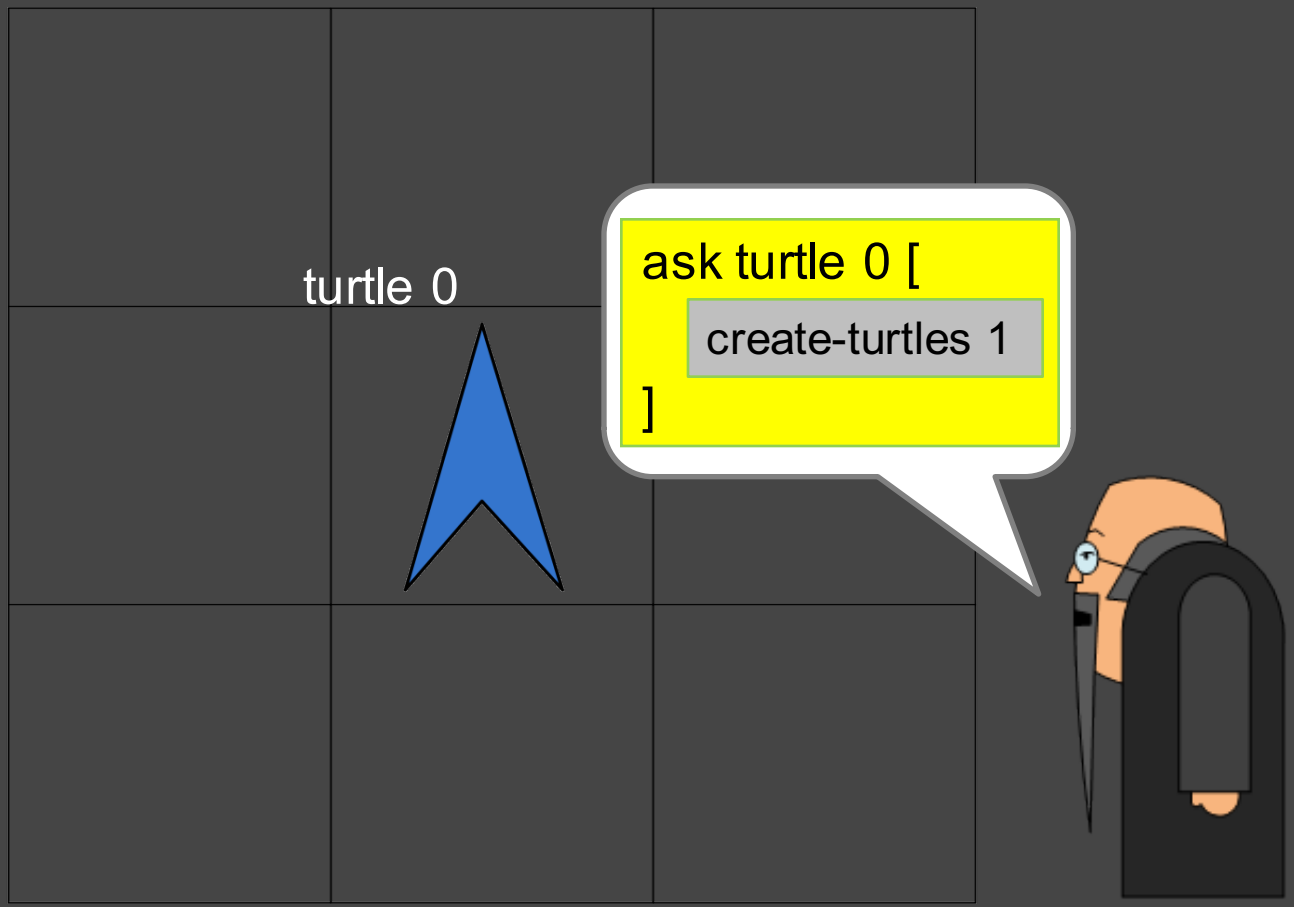
```
ask somebody [
   do this
   do that
]
```

- The commands are executed by the called turtle or patch, and influence its properties

Context

Because observer, turtles and patches are inherently different, only commands that the called entity understands can be issued

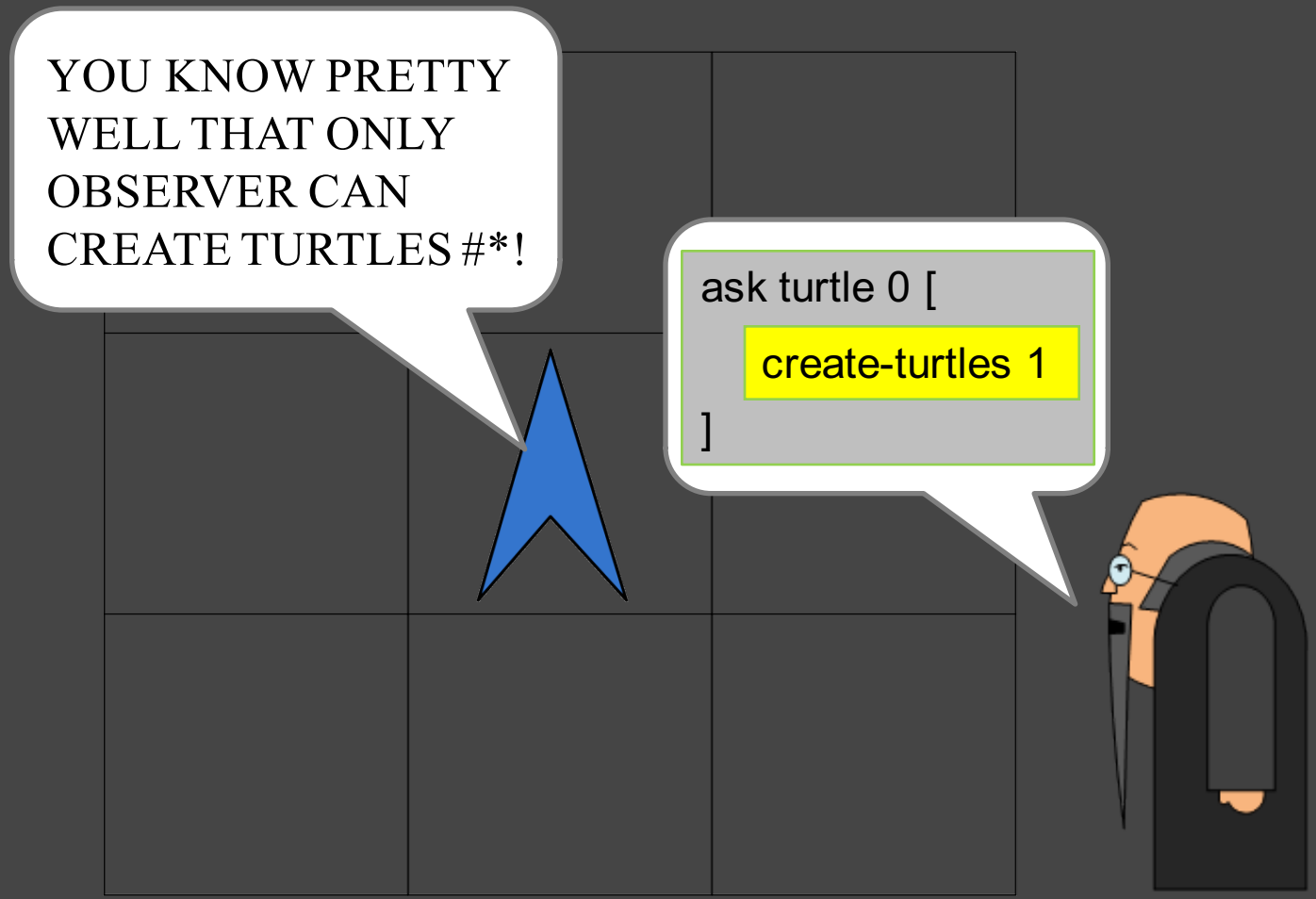
- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?



Context

Because observer, turtles and patches are inherently different, only commands that the called entity understands can be issued

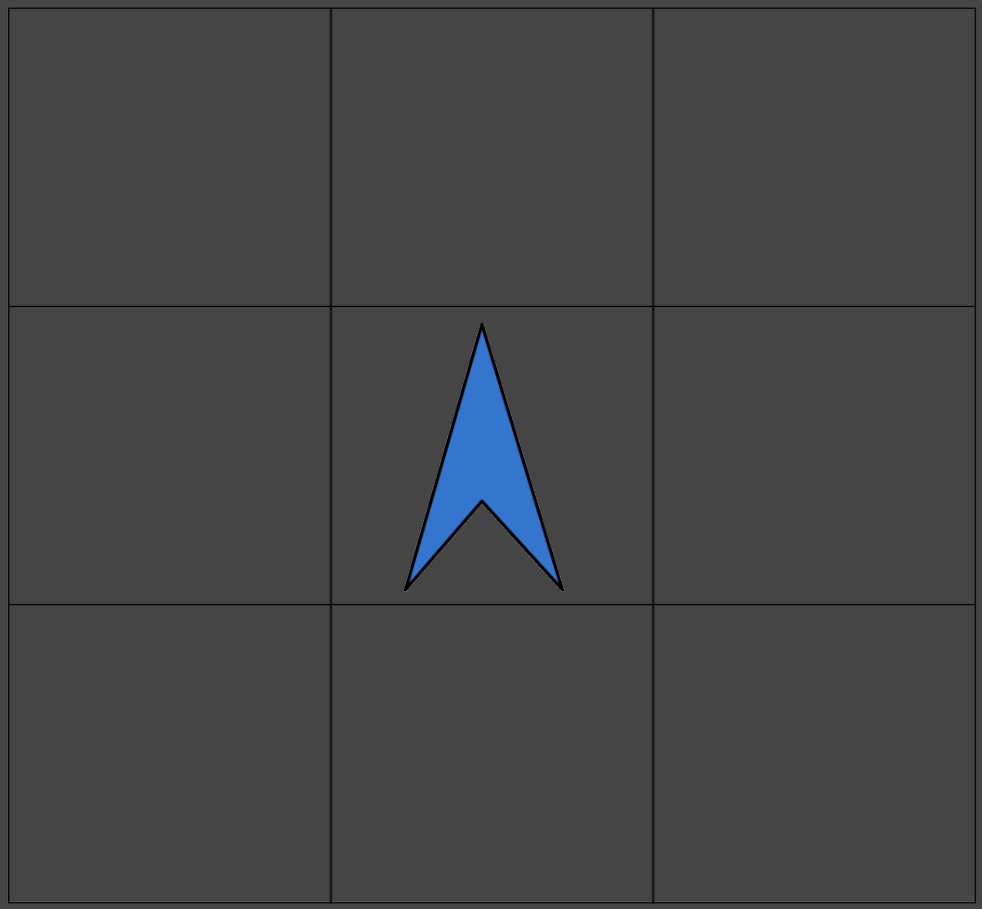
- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?

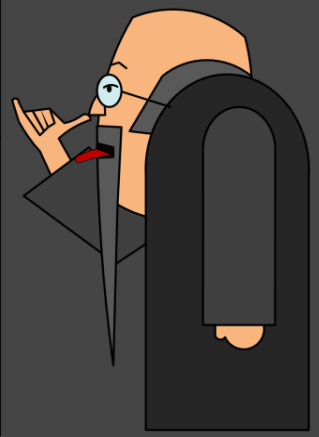


Context

Because observer, turtles and patches are inherently different, only commands that the called entity understands can be issued

- who
- heading
- xcor
- ycor
- shape
- size
- color
- hidden?





Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 33, © 2012, Technical University Vienna.

Commands for turtles

set *property value*

sets a property to a specified value

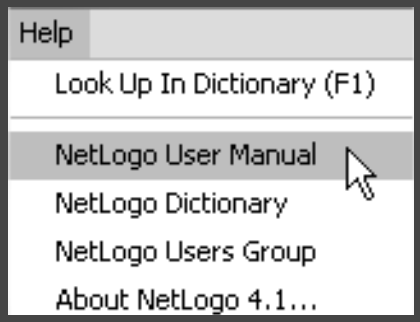
forward *patch-units*, **back** *patch-units*

moves a turtle in the current direction

left *degrees*, **right** *degrees*

alters the heading of a turtle

...and every other command listed in the Netlogo Dictionary under "Turtle-related" (see: Menu - Help – NetLogo Dictionary)



Have you seen it?

The NetLogo Dictionary is NetLogo's central source for help.

http://ccl.northwestern.edu/netlogo/docs/

Assignment on your own

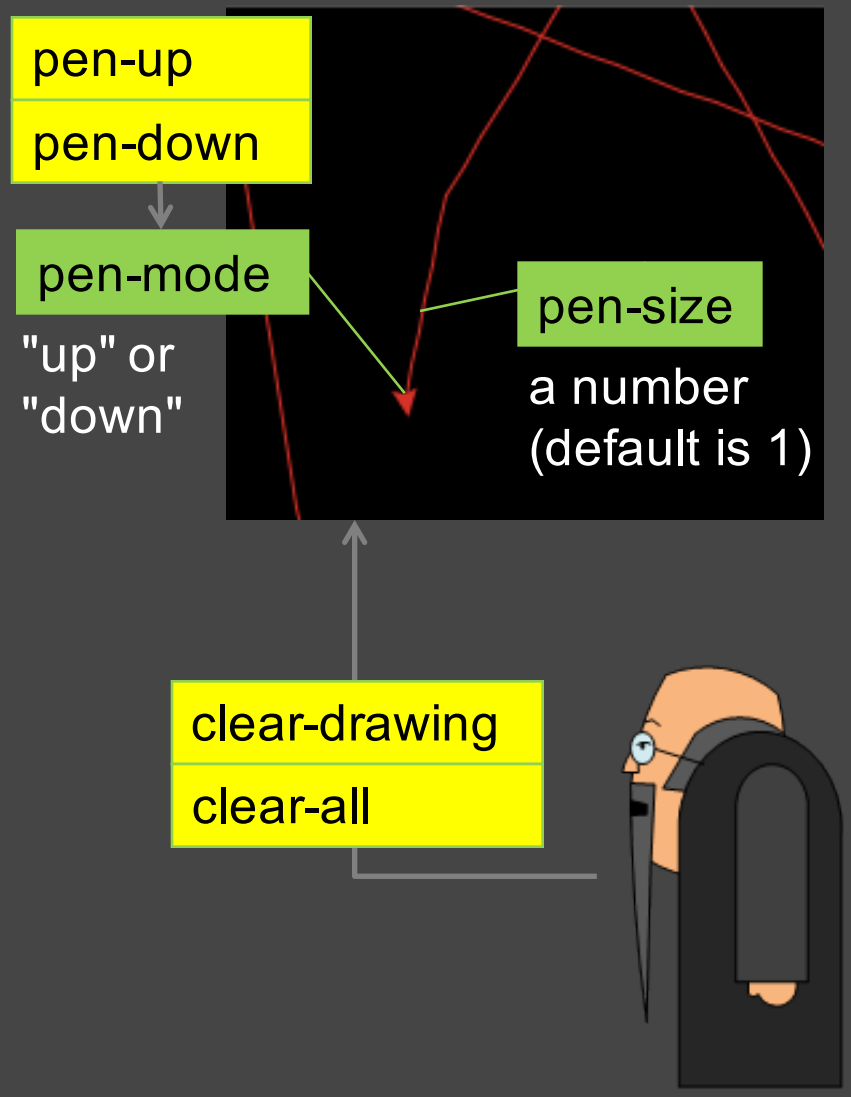
1. let observer ask turtle 0
   - to set its property "color" to yellow
   - to issue the following commands:
     forward 1 left 45 forward 1
2. see for yourself what happens when you run:
   ask turtle 0 [create-turtles 1]
3. look inside the NetLogo Dictionary and find the meaning of the following commands:

   - pen-up, pen-down

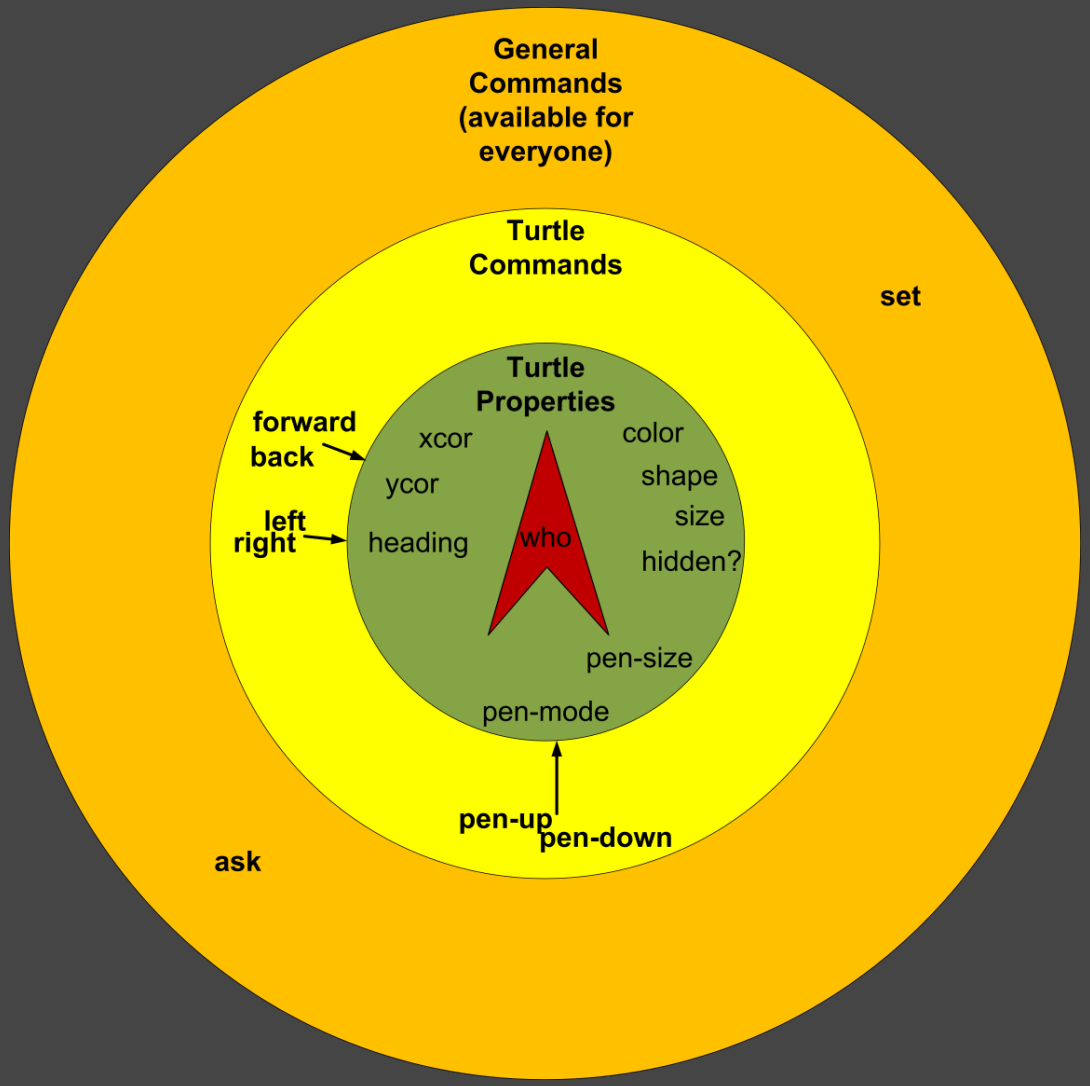
   then, experiment with these using forward, left and right as additional commands!

Results (Probably)

- pen-down and pen-up change the state of a property named "pen-mode"
- color of track equals color of turtle
- thickness of track can be set using the property "pen-size" (also found in a turtle)
- observer may erase the tracks by using the command "clear-drawing" or everything including turtles with "clear-all"



Summing up

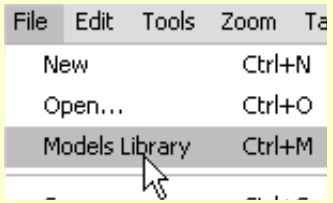

General Commands (available for everyone)
Turtle Commands
Turtle Properties
forward
back
left
right
xcor
ycor
heading
who
color
shape
size
hidden?
pen-size
pen-mode
pen-up
pen-down
set
ask

WRITING PROGRAMS

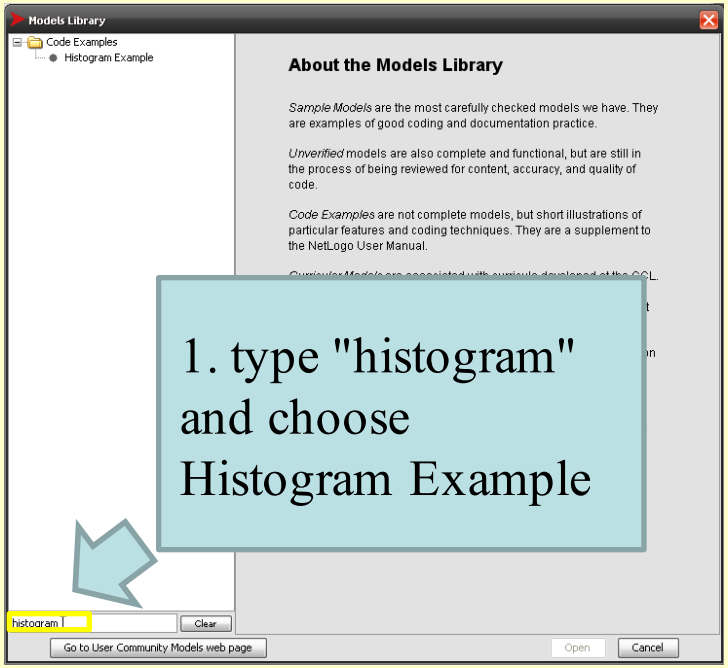
General NetLogo program layout

1. set up the program (once), e.g.
   - clear everything,
   - set the environment
   - create agents
2. simulation loop (called repeatedly)
   - simulate world
     e.g. grain growth on patches
   - simulate agent behaviour
     e.g. movement, interaction
   - update charts & plots

Example from Models Library

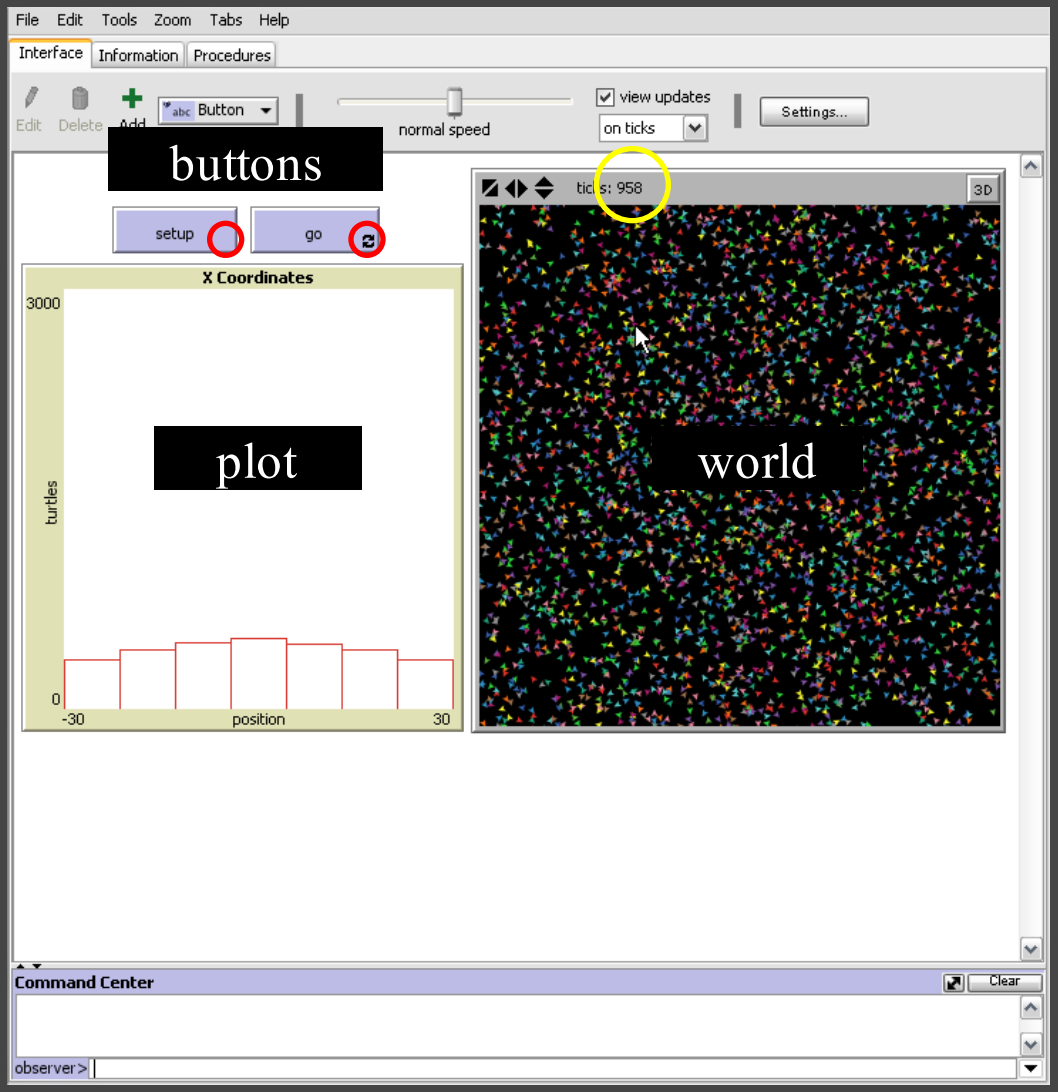


1. open the "Models Library"



1. type "histogram" and choose Histogram Example

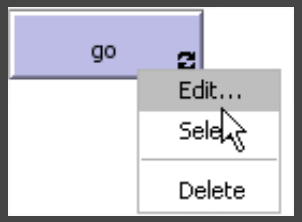
Exploring the "Histogram Example"



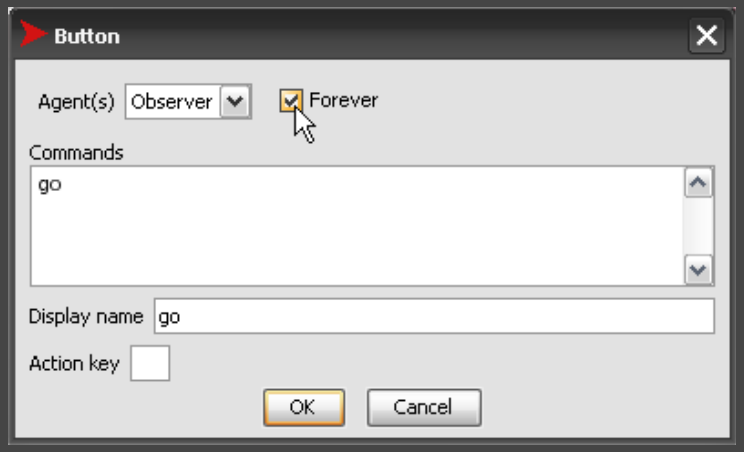
- the two buttons "setup" and "go" are used to interact with the model
- setup clears and fills the world,
- go simulates and generates the histogram (repeatedly, in time steps – "ticks")

➔ click setup, then go

Forever or not forever



0. (Unpress the "go" button)
1. Right-click on the go button
2. Select Edit...

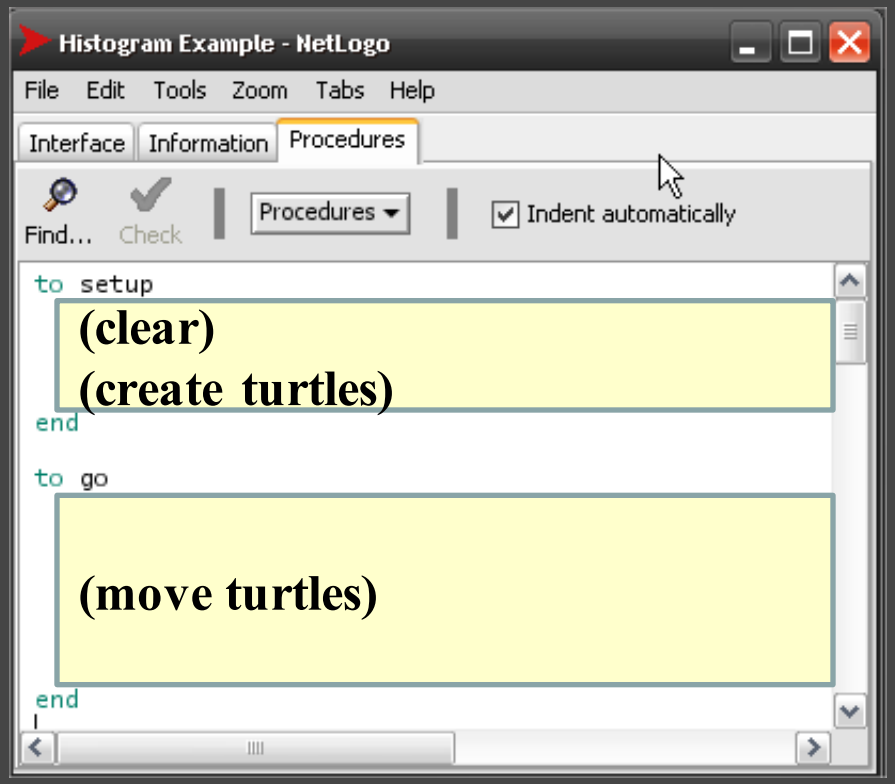


3. A dialog appears
4. Unclick "Forever"
5. Choose "OK"

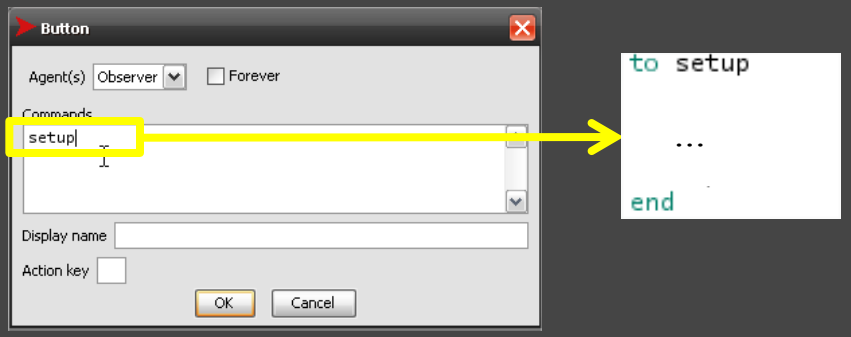
➔ click on "setup", then "go"

- Buttons are used to call setup and simulation routine
- Choosing "Forever" on a button will repeatedly call it, therefore establishing a loop

Behind the scenes

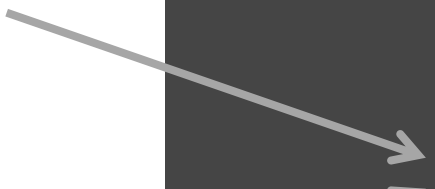


1. Go to the procedures tab
2. observe the two routines "to setup" and "to go", that contain the actual code that is performed for setting up and performing a sim step
3. note how these routines are called from the buttons:



Procedures

```
to setup
   ...commands...
end

to go
   ...commands...
end
```



A set of commands is structured into logical units called

Procedures

that always have the syntax

```
to name of procedure
   commands
end
```

Note that the commands in a are the same as were entered in command center (observer>)

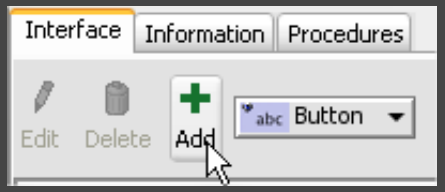
Writing the first program

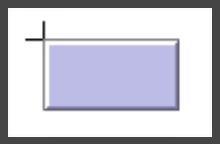
1. Choose File – New
2. Go to Procedures tab
3. Type the following code:

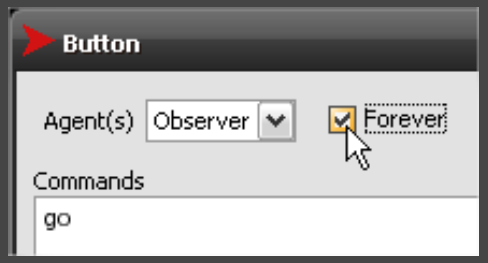
```
to setup                ;start of "setup"
  clear-all             ;clear world
  create-turtles 1      ;create turtle
end                     ;end of "setup"

to go                   ;start of "go"
  ask turtle 0 [        ;ask the turtle
    forward 1           ;move fwd 1 unit
  ]                     ;end of ask
end                     ;end of "go"
```

- These are comments, starting with ’
- Comments are ignored by NetLogo, but may help in understanding your code
- A common approach is "*comment before code*"

Connecting the program to the user interface







1. On the Interface tab, choose Add ("Button" must be selected in the neighboring dropdown)
2. Click anywhere within the white space to insert a button
3. A dialog appears
4. Enter "setup" in the Commands textfield and hit the "OK" button
5. Insert another button (using the same steps), enter "go" in Commands and enable "Forever", then choose "OK".

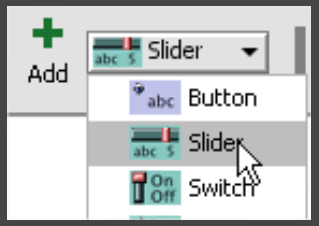
➔ press the "setup" button, then "go"

Introducing many turtles (battle plan)

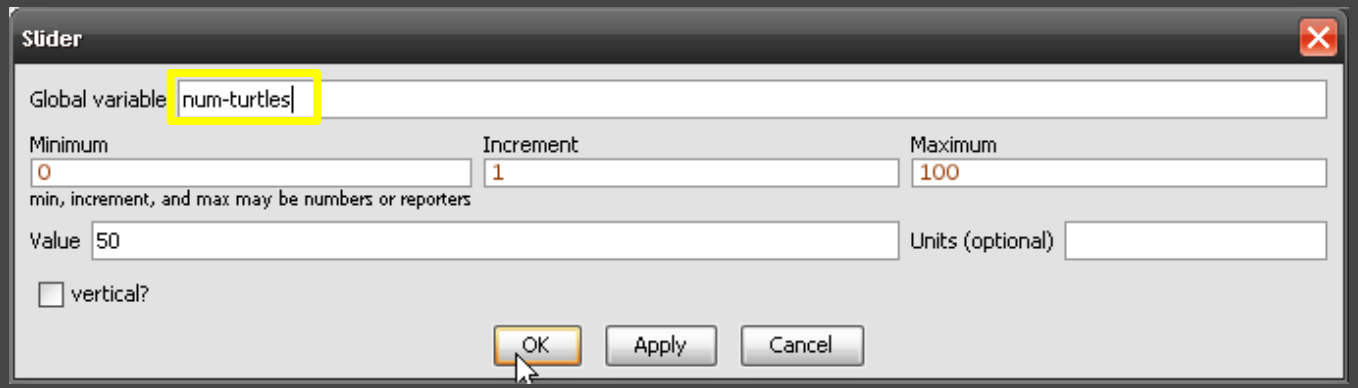
The presented program is now extended in order to create a whole population of turtles:

- introduce a slider named "num-turtles" which sets the number of turtles to create
- use this value in setup
- get hold of all turtles and tell them to set their heading, color and shape to a defined value
- furthermore, distribute all turtles over the available world

Adding a slider



1. In the Interface tab, click on the dropdown where "Button" is shown, in order to expose all available interface components.
2. Choose "Slider"
3. Click Add and click within the white space in order to add the slider
4. In the appearing dialog, add "num-turtles" in the "Global variable" textfield:



Creating num-turtles

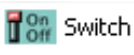
1. change setup as given below

```
to setup
  clear-all
  create-turtles num-turtles
end
```

2. run the changed procedure
   using the "setup" button
3. take note of the dozens of turtles created using inspect
   (right mouse click on the turtles)

Global variables

- The value of the slider "num-turtles" is available within the code by giving its name. We call this a **global variable**.
- There a four options for defining global variables which are controllable via the user interface:

			
Produces a number within the range set in the "Slider" dialog	Produces a Boolean (true or false)	Can produce any data type, depending on the list of values entered in the "Chooser" dialog	Can produce a number, string or color, depending on the set data type given in the "Input" dialog

Asking all turtles

- If all turtles should be asked, the term **ask turtles** is used:

```
to setup
  clear-all
  create-turtles num-turtles
  ask turtles [
    set color RED
    set shape "person"
    set heading 0
  ]
end
```

- The contained commands are in line with the battle plan for extending the program, i.e. to give the turtles common form

Why we need reporters

- As can be seen, the turtles all stick to the origin (0, 0)
- What is needed is a piece of code that puts each turtle in a random location of the world, something like:

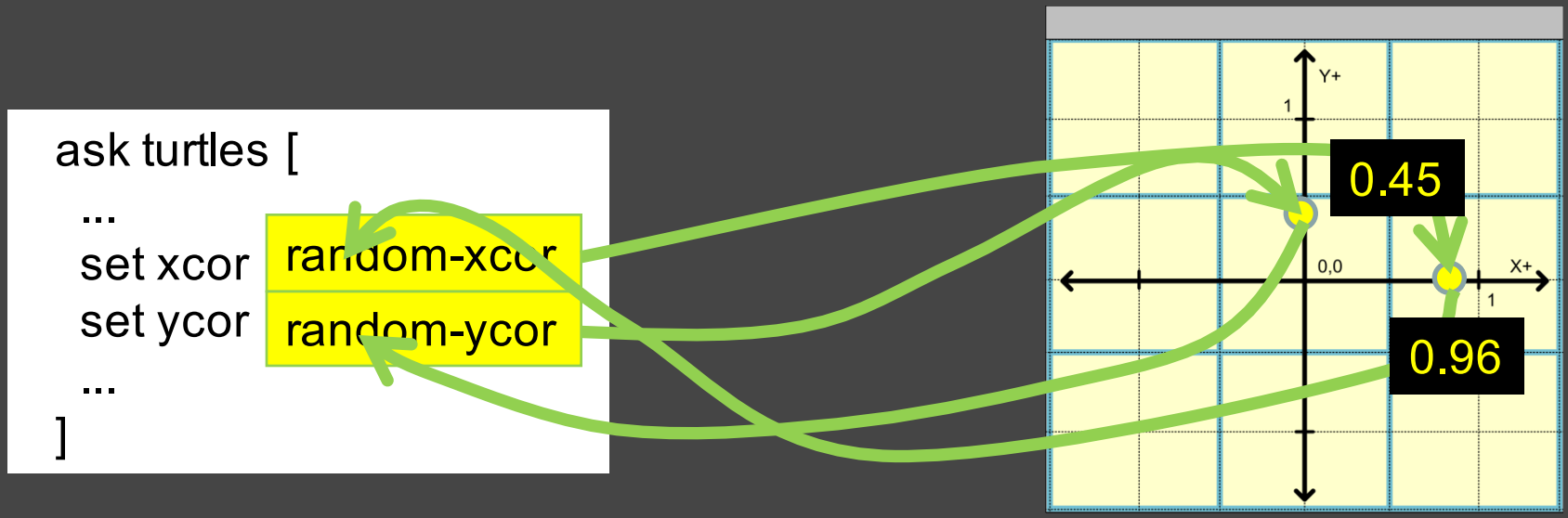
```
ask turtles [
  ...
  set xcor to some random x-coordinate
  set ycor to some random y-coordinate
  ...
]
```

- There are two commands which are made for this purpose, which we will be dealing with in due course:

random-xcor and random-ycor

Introducing reporters

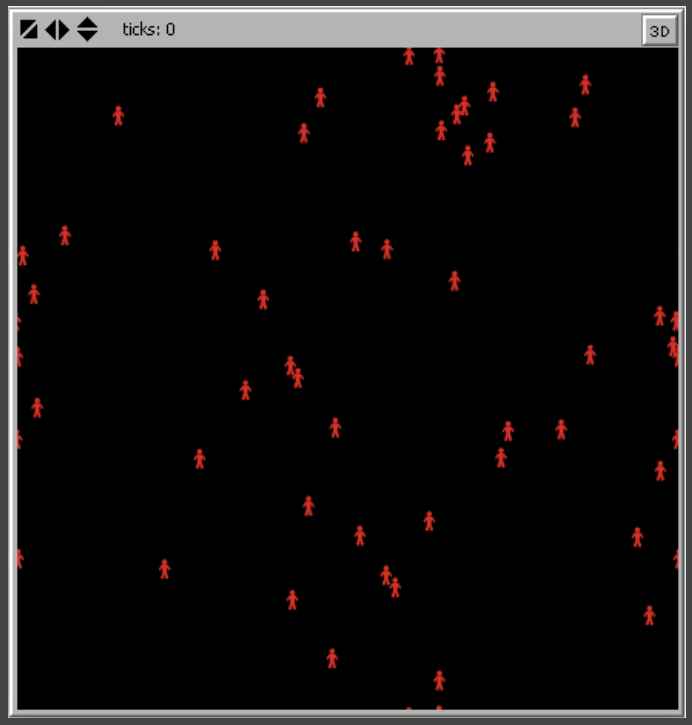
- The presented commands are called **reporters**.
- They compute a value (in this case: a number representing a random co-ordinate) and pass it to the caller:



Another run

Try out the new setup code:

```
to setup
  clear-all
  create-turtles num-turtles
  ask turtles [
    set color RED
    set shape "person"
    set heading 0
    set xcor random-xcor
    set ycor random-ycor
  ]
end
```

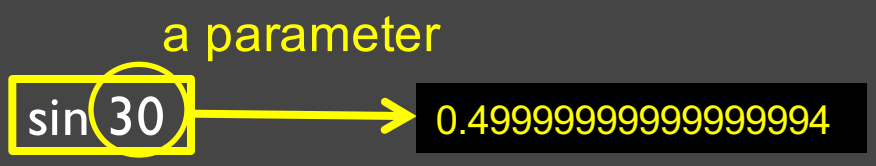


Hint: you can also execute a procedure by typing its name into the command center

Comparing reporters to global variables

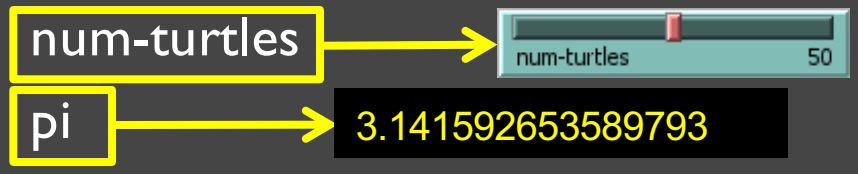
reporter

- can be used as placeholder for a concrete value
- value is produced by computation (e.g. generate random position)
- may need to pass **parameters** needed for computation, e.g.:



global variable

- can be used as placeholder for a concrete value
- value is produced by the user (e.g. by adjusting a slider) or is predefined (see further down)
- is "just a value" you can refer to, e.g.:



Wurzer: "Mathmod 2012 Netlogo Tutorial", Lecture Notes, Slide 55, © 2012, Technical University Vienna.

Asking immediately at creation time (Hint)

Instead of creating and then asking the turtles....

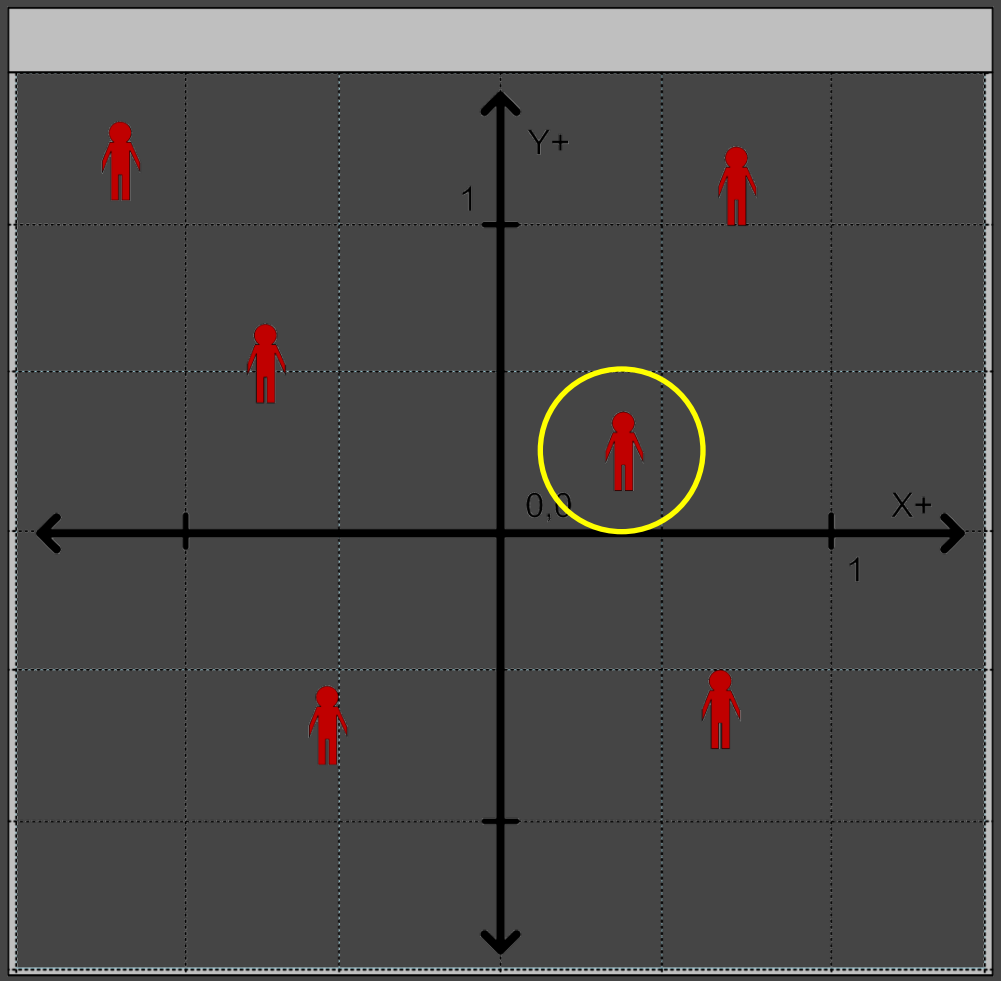
```
to setup
  clear-all
  create-turtles num-turtles
  ask turtles [
    set color RED
    set shape "person"
    set heading 0
    set xcor random-xcor
    set ycor random-ycor
  ]
end
```

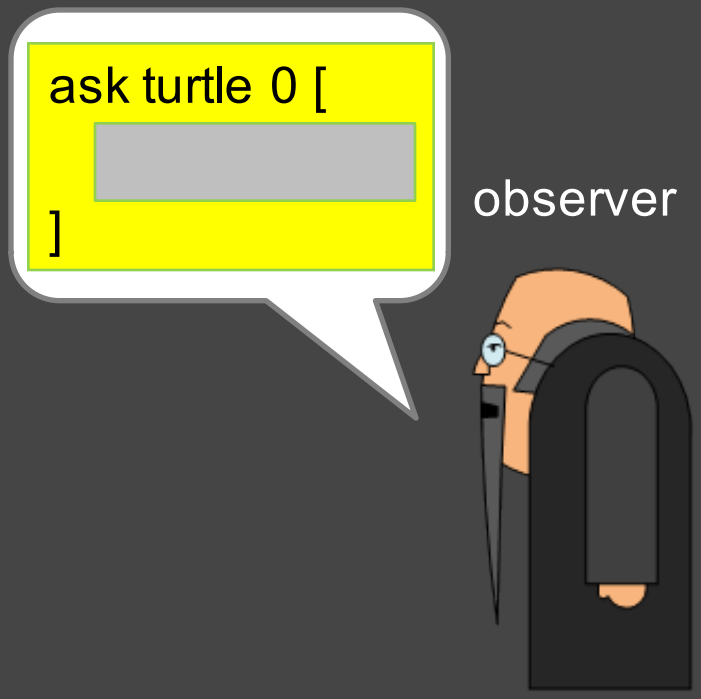
You may also use the more convenient form....

```
to setup
  clear-all
  create-turtles num-turtles [
    ; and tell them immediately to...
    set color RED
    set shape "person"
    set heading 0
    set xcor random-xcor
    set ycor random-ycor
  ]
end
```

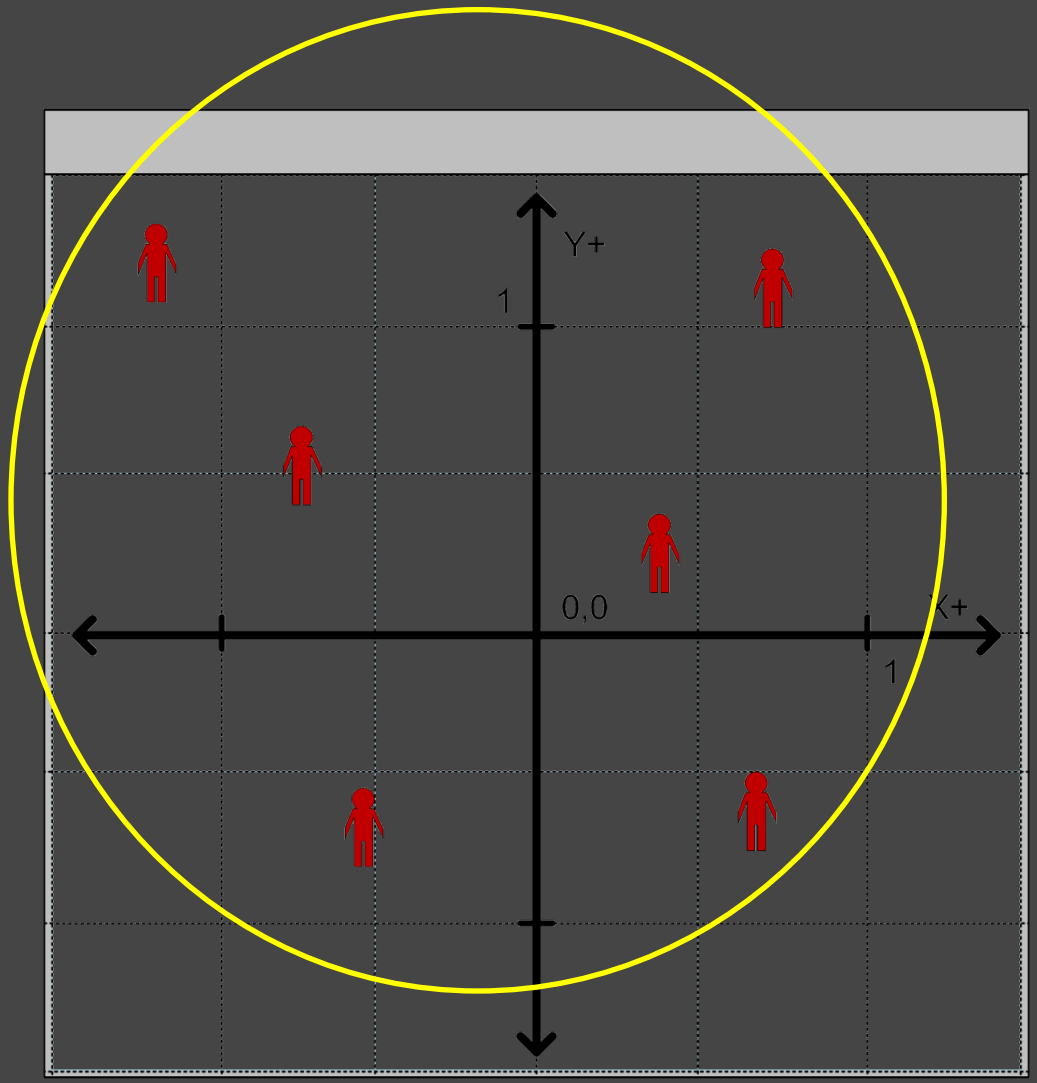
ADRESSING SETS OF TURTLES

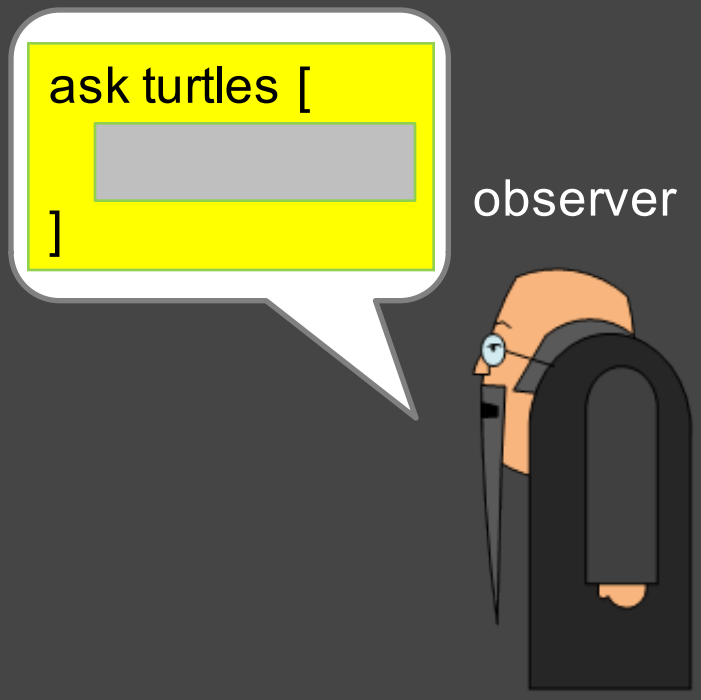
So far...



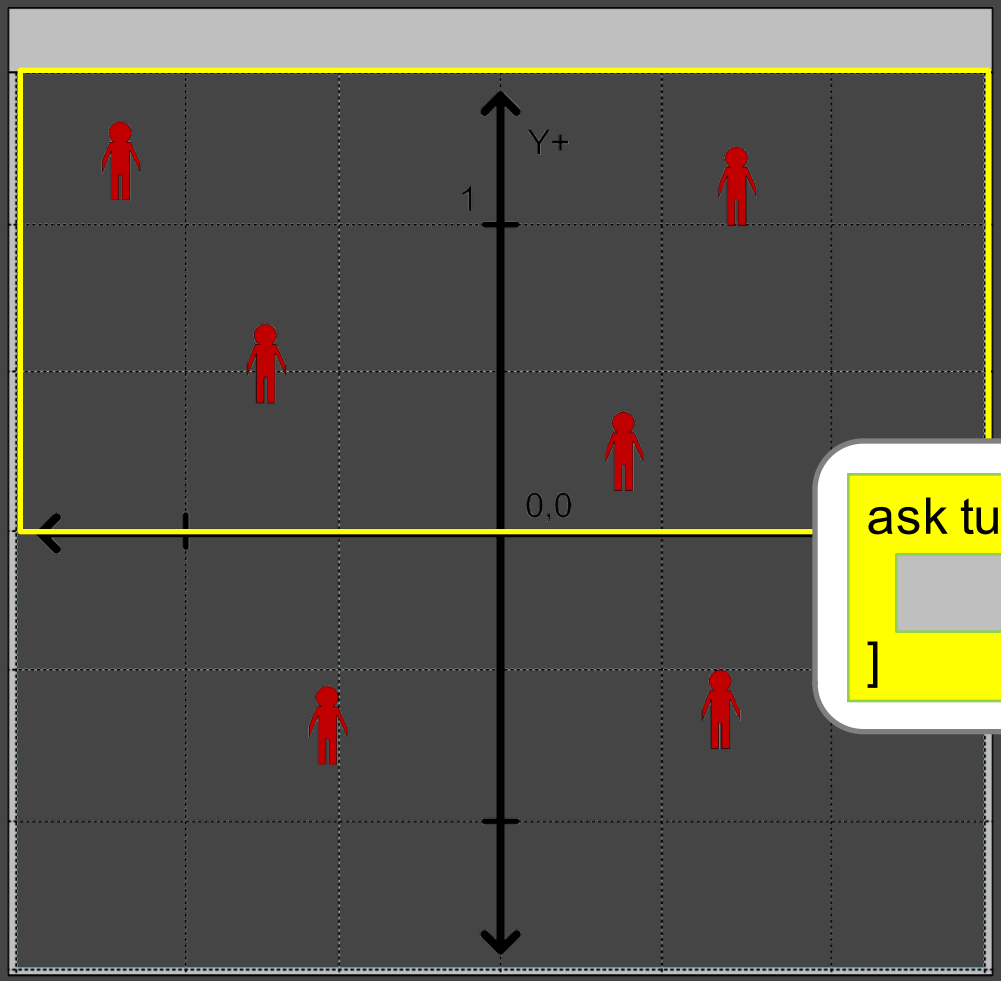


So far...





The "with" query



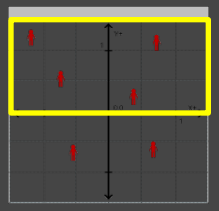
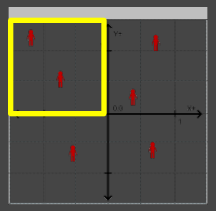
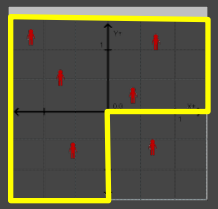
- Can address turtles *based on their properties* using a "with" query:

```
ask turtles with [ycor > 0] [
  
]
```

observer

With in a nutshell

- with takes arbitrary criteria as boolean conditions, e.g.

with [ycor > 0]	
with [ycor > 0 **and** xcor < 0]	
with [ycor > 0 **or** xcor < 0]	

Boolean conditions?

You need to formulate a query which produces true or false,

- for numbers, use comparisons (>, <, >=, <=, !=)
- for Booleans, also use comparisons (=, !=), i.e.

```
turtles with [hidden? = true]
turtles with [hidden? != true]
```

- strings can use string comparisons (=, !=),
  e.g. turtles with [shape = "person"]

Logical operators

You can also use the logical operators **and**, **or** and **not** to tie multiple conditions together or negate their result:

Logical operator	Usage	Example
condition 1 **and** condition 2	All conditions linked by **and** are required	(shape = "person") **and** (ycor > 0) **and** (xcor < 0)
condition 1 **or** condition 2	One condition among those linked by **or** is required	(ycor > 0) **or** (xcor < 0)
not condition	The opposite of the following condition is taken	**not** (shape = "person")

Note: it is sometimes necessary to use brackets for each condition (e.g. in *not shape = "person"*)

Simplified Boolean conditions (Hint)

Instead of writing "turtles with [hidden? = true] " you can use the short form:

turtles with [**hidden?**]

Instead of writing "turtles with [hidden? = false]" you can use the short form:

turtles with [**not hidden?**]

Hands on...

```
to setup
  clear-all
  create-turtles num-turtles [
    set xcor random-xcor
    set ycor random-ycor
    set color BLUE
  ]
  ask turtles with[ycor >= 0][
    set shape "person"
    set color RED
  ]
end
```

```
to go
  ask turtles with[shape = "person"] [
    forward 1
  ]
end
```

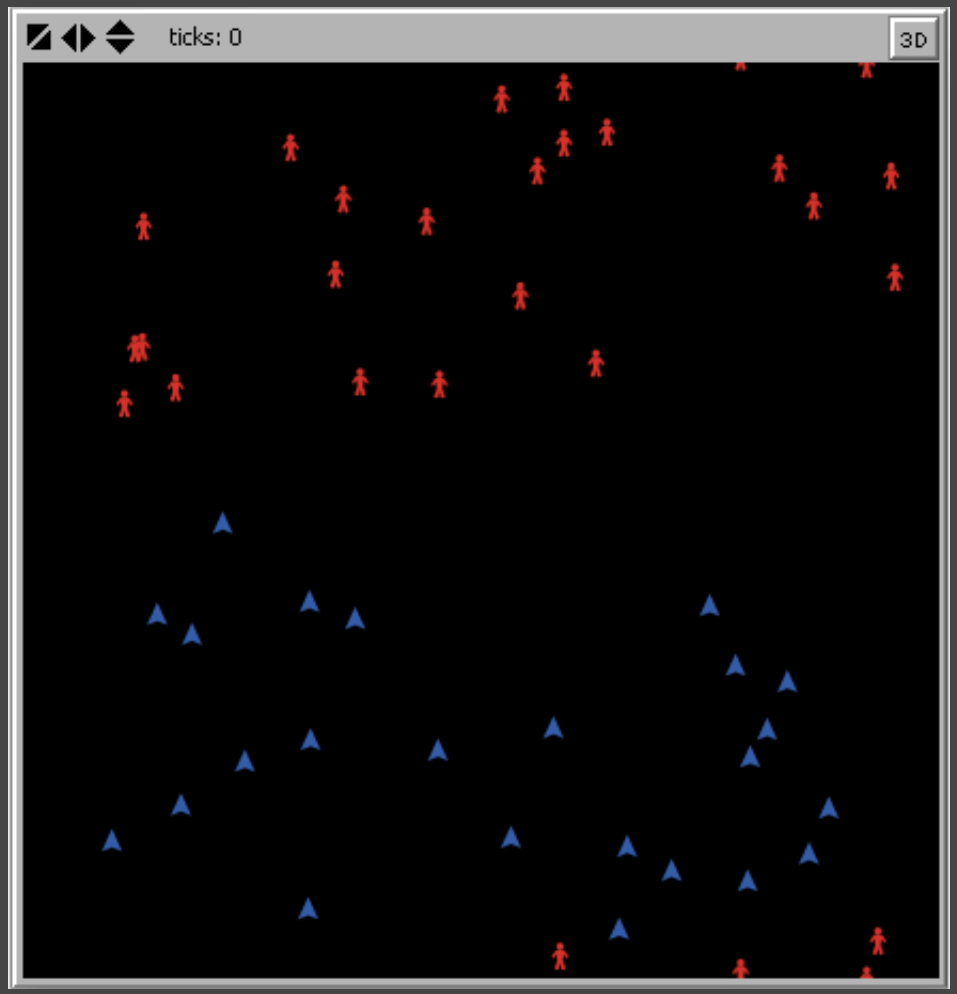
"setup" changes turtles in two steps:

- at creation, all turtles are set to color blue.
- then, those turtles that have a positive y co-ordinate are set to color red and shape "person"

"go" will only move the "person" turtles

Further thoughts on the program

- have visually introduced two distinct groups of turtles
- it would be more elegant to explicitly name the groups rather than changing shapes of turtles
- additional benefit: each group could have additional properties



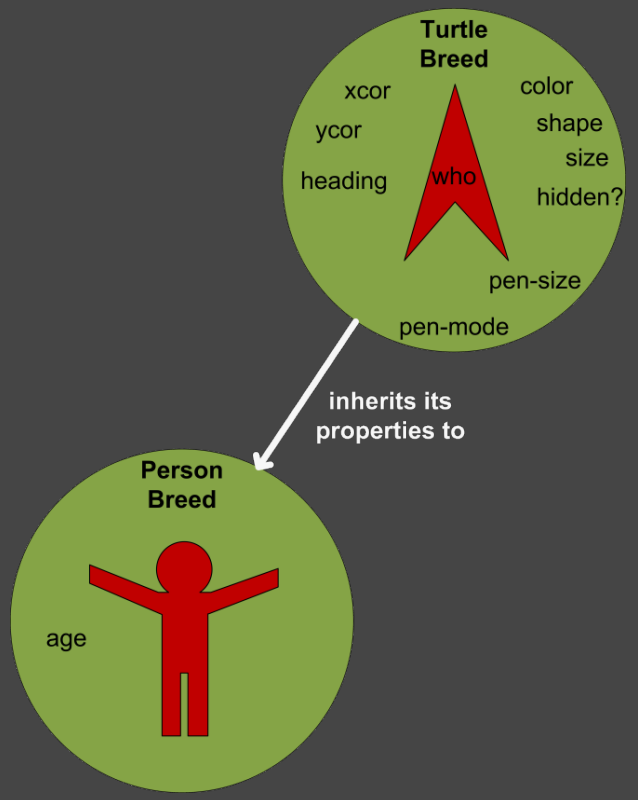
So we come to breeds

- NetLogo predefines one "breed" of agents called turtles
- As programmer, you are free to define more breeds using the "breed" keyword:

```
breed [persons person]
```

plural singular

- A breed has every property a turtle has. It can additionally define a set of properties *specific to that group*.



Defining additional properties

- Additional properties for each breed may be defined using the "*breed name plural*-own" statement:

```
breed [persons person]
persons-own [
  age
]
```

- To generate turtles belonging to a certain breed, use "create-*breed name plural* " rather than "create-turtles":

```
create-persons 1 [
  set shape "person"
  set age 0
]
```

It is good coding practice to set the additional properties of a breed to sensible default values at creation time

Adressing members of a breed

Similar to the case of turtles, a new breed can be adressed as a whole or by naming a specific agent:

turtles breed

```
ask turtles with [...] [
  ...
]
ask turtle 0 [
  ...
]
```

persons breed

```
ask persons with [...] [
  ...
]
ask person 0 [
  ...
]
```

Putting it into practice (your turn)...

```
breed [persons person]
persons-own [
  age
]

to setup
  clear-all
end
```

- The following code is a rewrite of the previous program
- Before procedures start, a new breed persons is declared and given the additional property "age"
- Setup only clears the world

Putting it into practice (your turn)...

```
to go
  create-persons num-turtles [
    set xcor random-xcor
    set ycor random-ycor
    set color RED
    set age 0
  ]
  ask persons [
    set age (age + 1)
  ]
  ask persons with [age > 80][
    die
  ]
end
```

- In every step, persons are born
- A simulation step represents a year. Therefore, the property "age" of each person is incremented.
- Then, all persons older than 80 die (*new command*)

Note: You need to set num-turtles to a fairly small value (e.g. 2) in order to avoid a too high number of active people

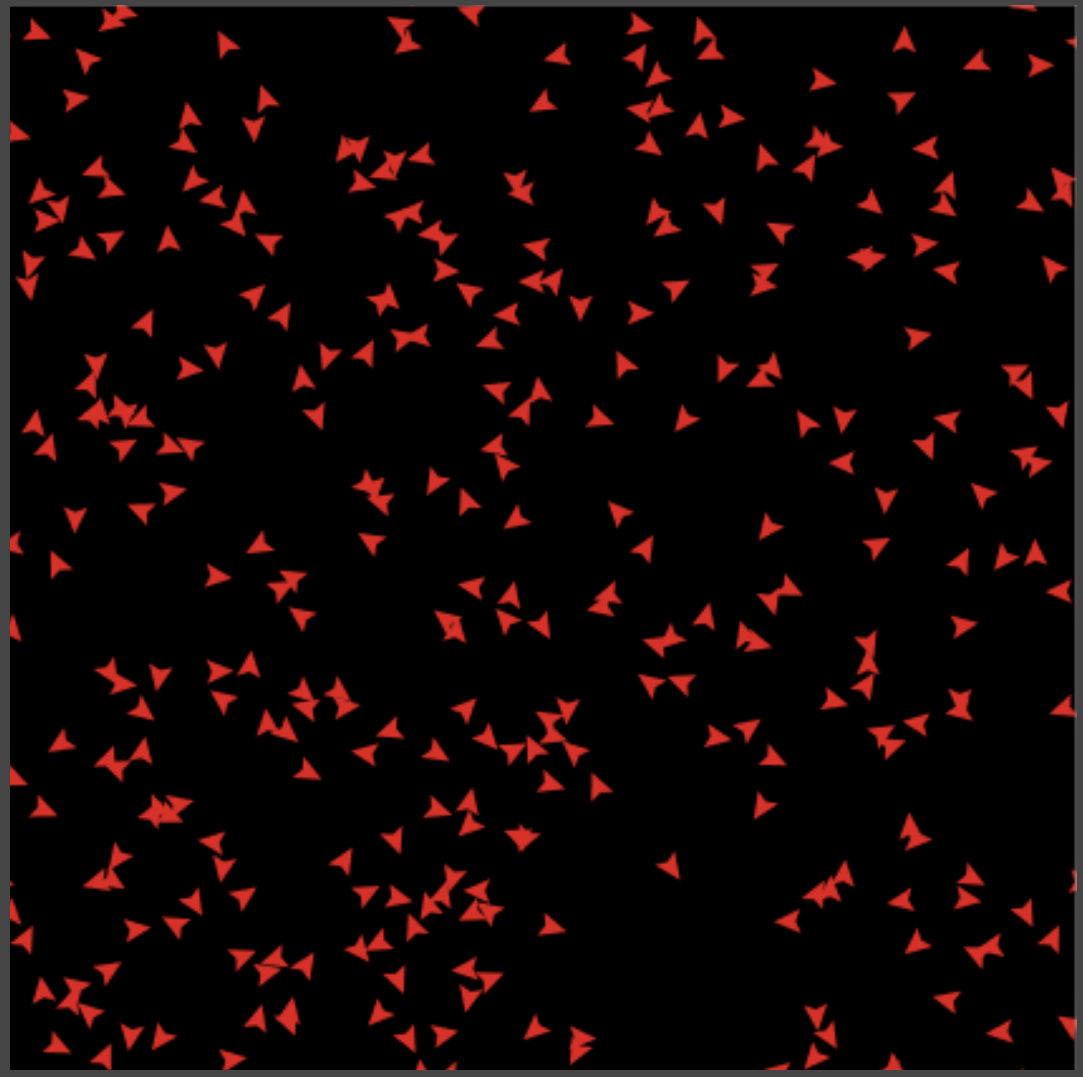
Concluding notes

The turtle breed is predefined *but extensible*:

– You can add new properties using

```
turtles-own [
  ...
]
```

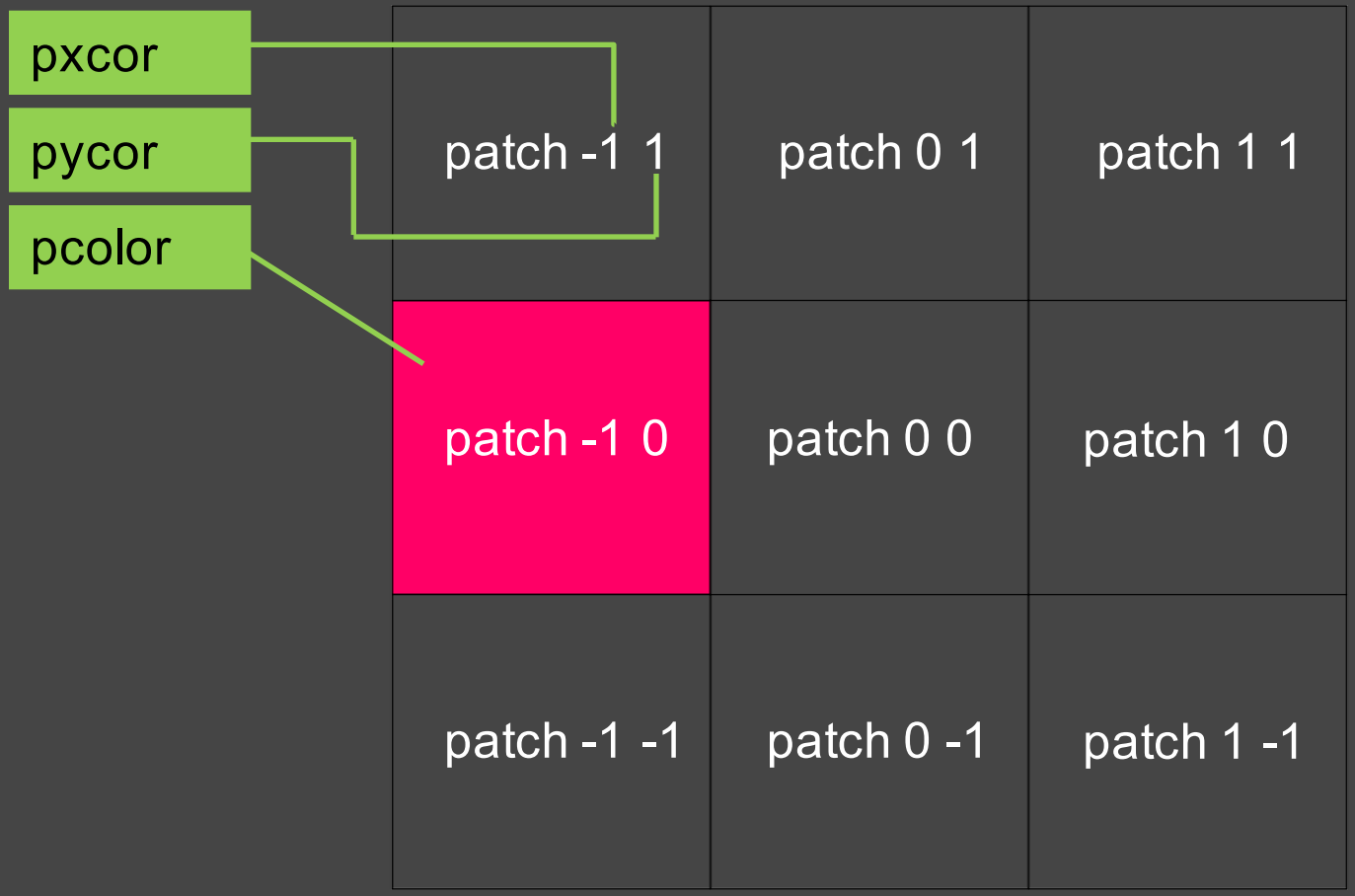
, however, remember that all properties are inherited to the other breeds as well



PATCHES

Introducing patches

Patches are an own (predefined) breed with the following properties:



Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 74, © 2012, Technical University Vienna.

Adressing patches

All patches can be adressed using the breed-name **patches**

- pxcor
- pycor
- pcolor

patch -1 1	patch 0 1	patch 1 1
patch -1 0	patch 0 0	
patch -1 -1	patch 0 -1	patch 1 -1

```
ask patches [
  set pcolor RED
]
```

Adressing patches

All patches can be adressed using the breed-name **patches**

- pxcor
- pycor
- pcolor

patch -1 1	patch 0 1	patch 1 1
patch -1 0	patch 0 0	
patch -1 -1	patch 0 -1	patch 1 -1

```
ask patches [
  set pcolor RED
]
```

Adressing patches

All patches can be adressed using the breed-name **patches**

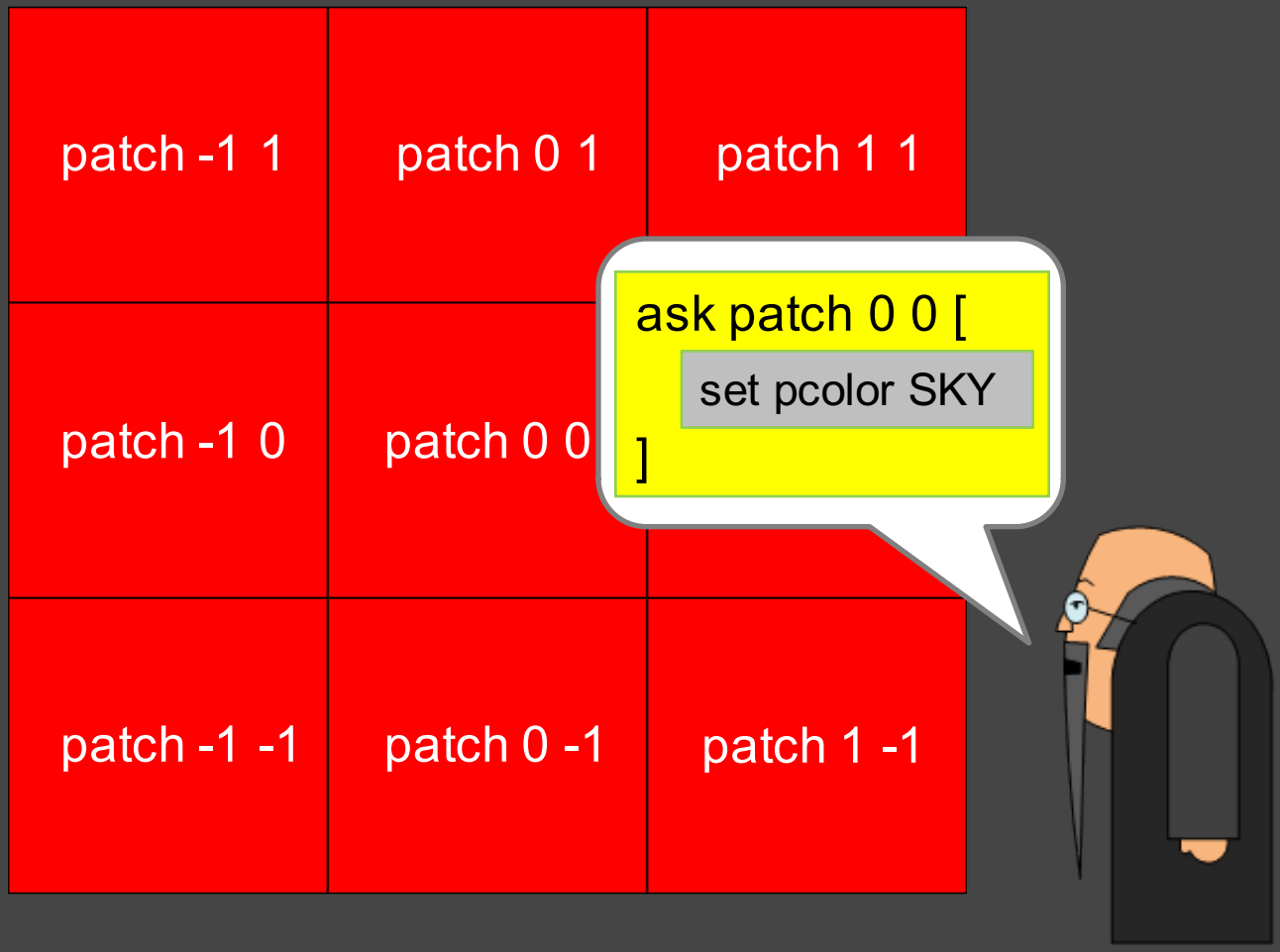
- pxcor
- pycor
- pcolor

patch -1 1	patch 0 1	patch 1 1
patch -1 0	patch 0 0	patch 1 0
patch -1 -1	patch 0 -1	patch 1 -1

Adressing patches

In contrast to turtles, individual patches are addressed using their coordinates

- pxcor
- pycor
- pcolor



Adressing patches

In contrast to turtles, individual patches are adressed using their coordinates

- pxcor
- pycor
- pcolor

patch -1 1	patch 0 1	patch 1 1
patch -1 0	patch 0 0	
patch -1 -1	patch 0 -1	patch 1 -1

```
ask patch 0 0 [
  set pcolor SKY
]
```

Adressing patches

In contrast to turtles, individual patches are adressed using their coordinates

- pxcor
- pycor
- pcolor

patch -1 1	patch 0 1	patch 1 1
patch -1 0	patch 0 0	patch 1 0
patch -1 -1	patch 0 -1	patch 1 -1

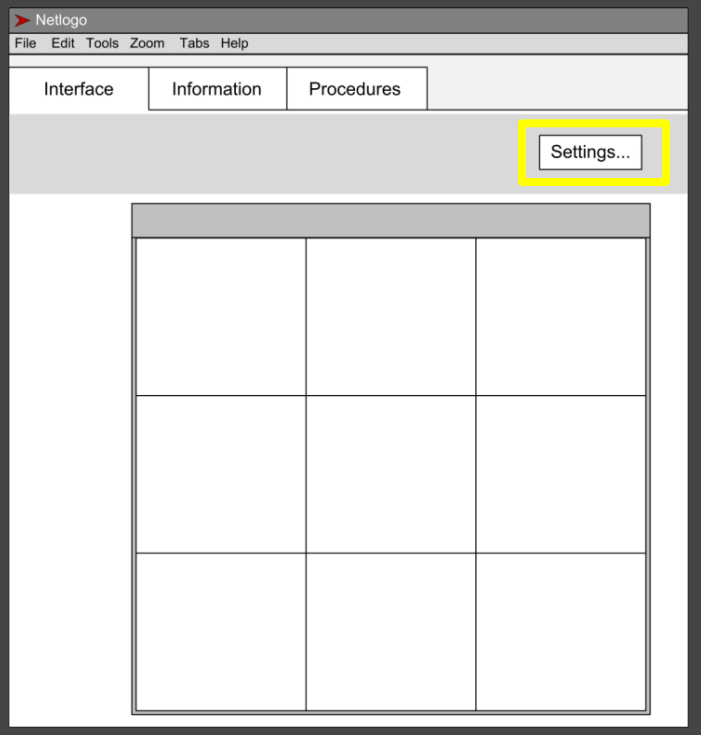
Wurzer: "Mathmod2012 NetLogo Tutorial", Lecture Notes, Slide 80, © 2012, Technical University Vienna.

Extending patches

- Additional properties can be specified via patches-own:

```
patches-own [
  soil-type
]
```

- However, there is no possibility of declaring additional patch breeds that inherit from patches.
- Furthermore, patches are not created using code (e.g. via create-patches 1) but by using the settings window of NetLogo



Try it out for yourself - Hands on exercise!

```
patches-own [
  soil-type
]
to setup
  clear-all
  ask patches [
    set soil-type random 2
  ]
  ask patches with [soil-type = 1][
    set pcolor YELLOW
  ]
end
```

1. Start a new program by entering the code visible to the left in the Procedures tab
2. Notice the new reporter random, and look up help for it in the NetLogo Dictionary. What does it do? Why "random 2" ?
3. After having finished the code, call "setup" using the Command Center

Code improvements

- If the color of each patch could be set in the following fashion:

```
ask patches [
  set soil-type random 2
  set pcolor to either
  yellow or black,
  depending on soil-type
]
```

, the second ask could be eliminated
- The question is: how do we assign *either yellow or black?*

Enter the if statement

- The **if** statement is used to test for a Boolean condition.
- If this yields true, NetLogo will run a series of commands:

```
ask patches [
  set soil-type random 2
  if soil-type = 1 [
    set pcolor YELLOW
  ]
]
```

in all other cases, the patch color remains as it is (black)

General syntax for **if**s:

```
if condition [
   commands to execute
   if the condition applies
]
```

The ifelse statement

- Should the patch color be either yellow (when soil-type is 1) or sky (in all other cases), an **ifelse** statement can be used:

```
ask patches [
   set soil-type random 2
   ifelse soil-type = 1 [
     set pcolor YELLOW
   ][
     set pcolor SKY
   ]
]
```

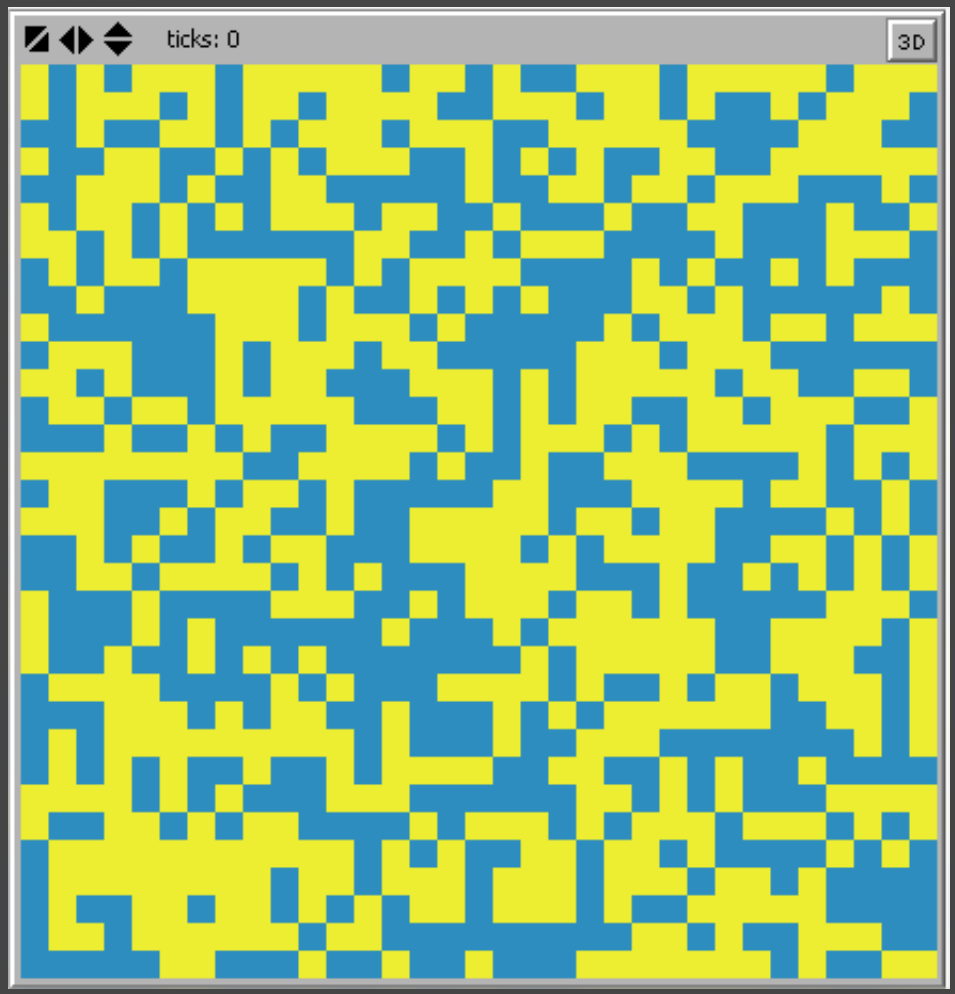
this happens when the condition is not true

General syntax for **ifelse**:

```
ifelse condition [
   commands to execute
   if condition applies
][
   commands to execute
   if condition does not
   apply
]
```

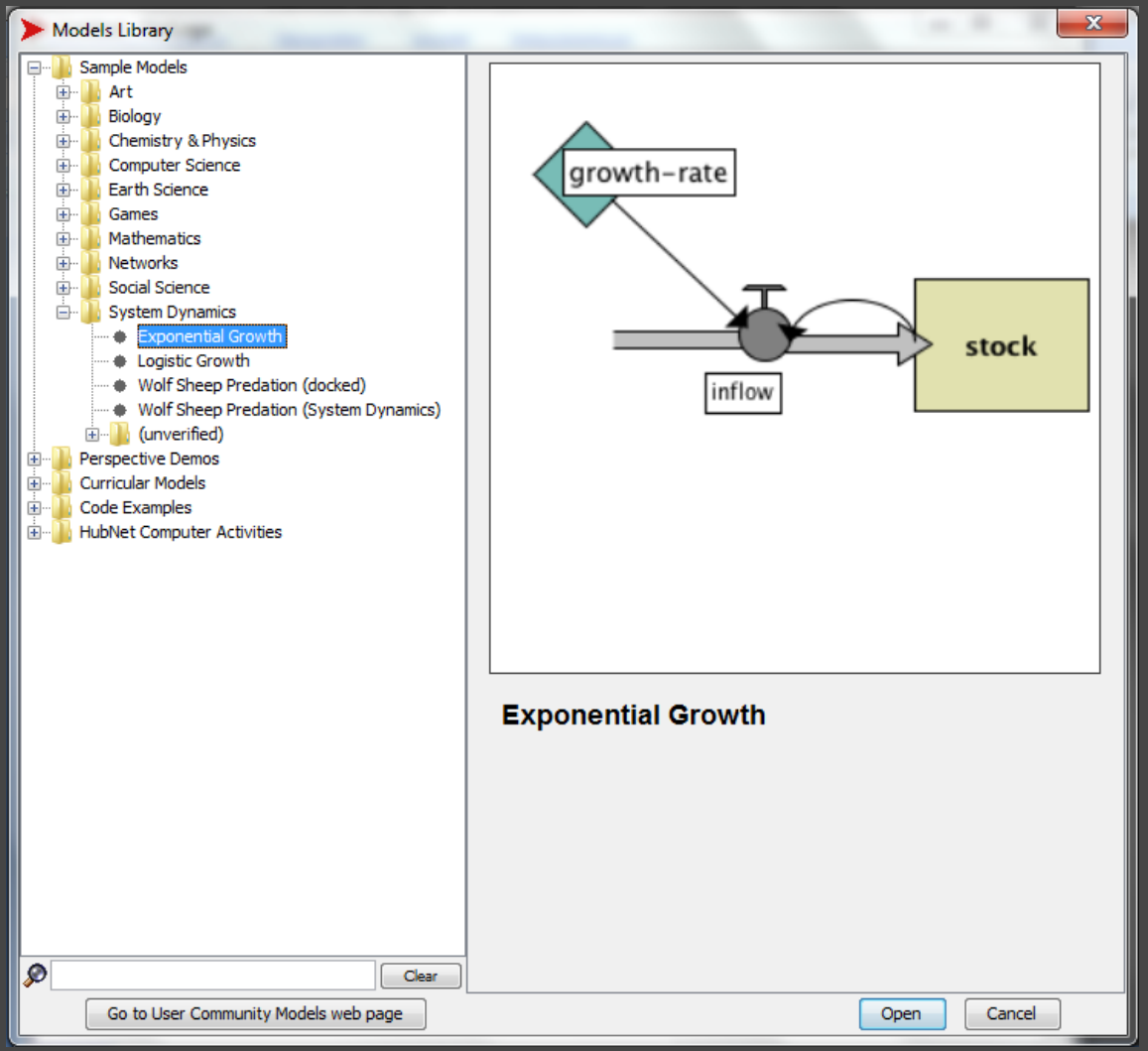
Program using ifelse...

```
patches-own [
  soil-type
]
to setup
  clear-all
  ask patches [
    set soil-type random 2
    ifelse soil-type = 1 [
      set pcolor YELLOW
    ][
      set pcolor SKY
    ]
  ]
end
```



There is more! e.g. System Dynamics (EBM!)

Take a look at "Exponential Growth" from models library!



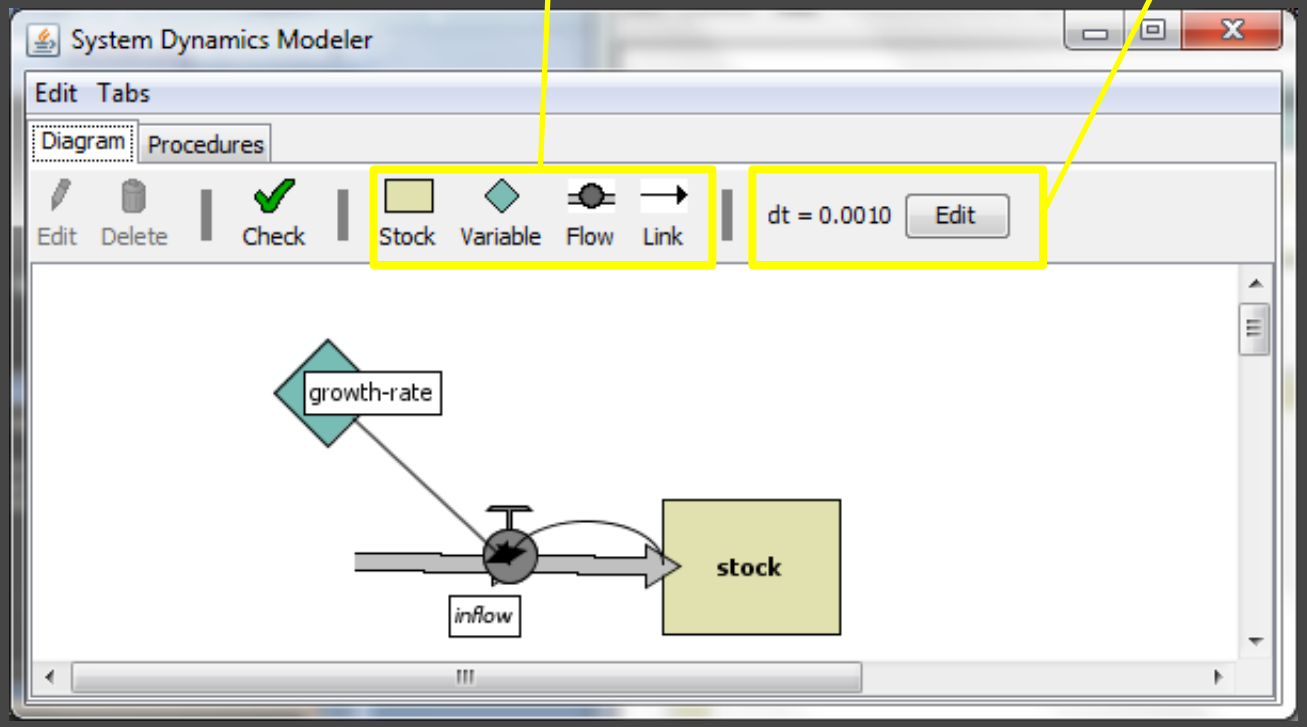
System Dynamics Interface

System Dynamics

opens in own window

stock, flow, converter, and information links

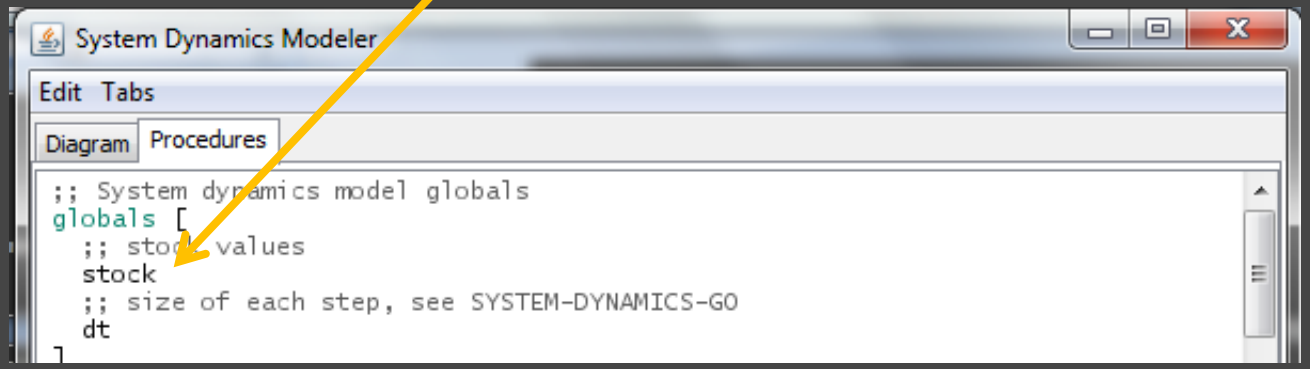
standard editing for time-step amount



Exploring the model…

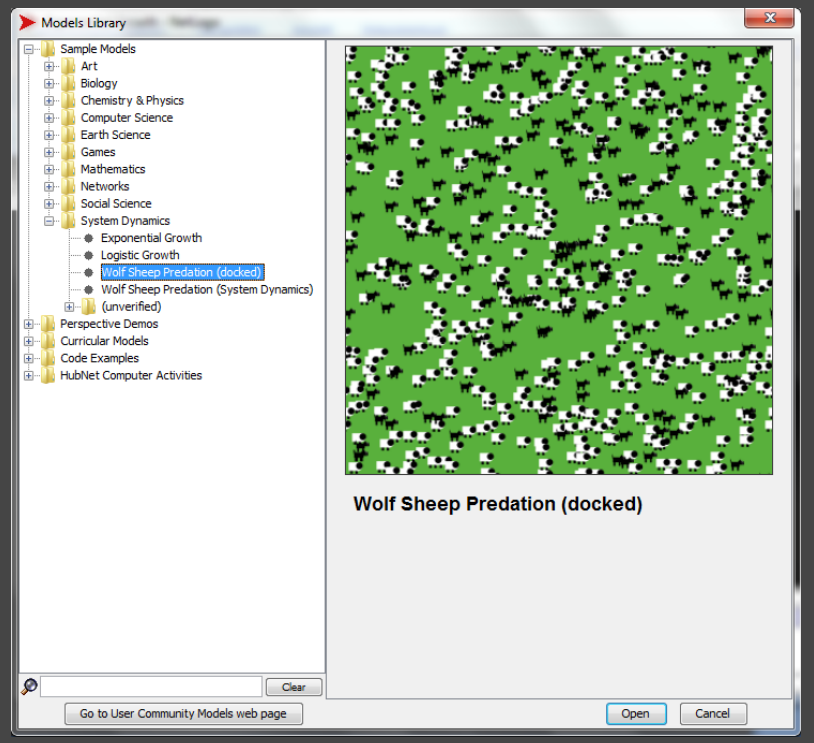
Double-click each element to find out:

- variable growth-rate links to slider "growth rate“ in NetLogo main screen
- flow "inflow" defined as formula "stock * growth-rate" – i.e. system dynamics elements are immediately available for use in formulas.
- why? see procedures!



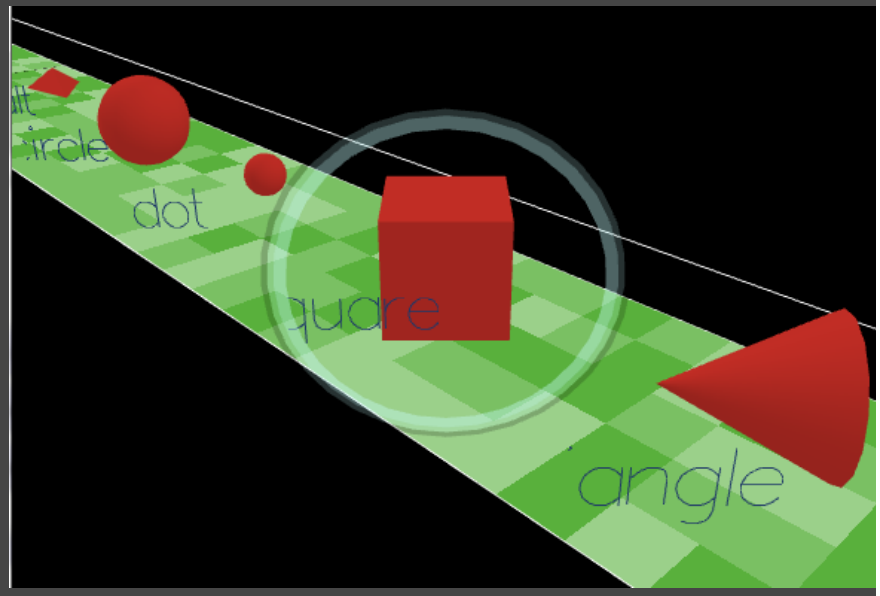
Great way to learn SD

- NetLogo exposes SD calculation as source code
- Therefore, can follow "what is done" and leverage existing NetLogo programming knowledge
- The best thing: can now mix SD/Agent-Based Models, see



And a lot more!

- GIS Extension – leverage your open government data to find a decent flat, for observer‘s sake!
- BehaviourSpace: vary parameters and run models multiple times – monte carlo and latin hypercube analyses
- Impress your friends with the 3D version of NetLogo (note: there is NetLogo 3D and the "3D View", which are different! In NetLogo 3D, the world is a cube and patches have a z-pcor)



go wild now.

THE END