NetLogo

- Free!
- Uses Java in the background
  - *Multi platform*
  - *Can be converted into applets (and embedded in websites)*
- Great for quickly putting a model together and thinking through ideas
  - *Easy to build*
  - *Easy to interact with models*
  - *East to extract data and create plots*
- Excellent documentation: http://ccl.northwestern.edu/netlogo/docs/

NetLogo Web

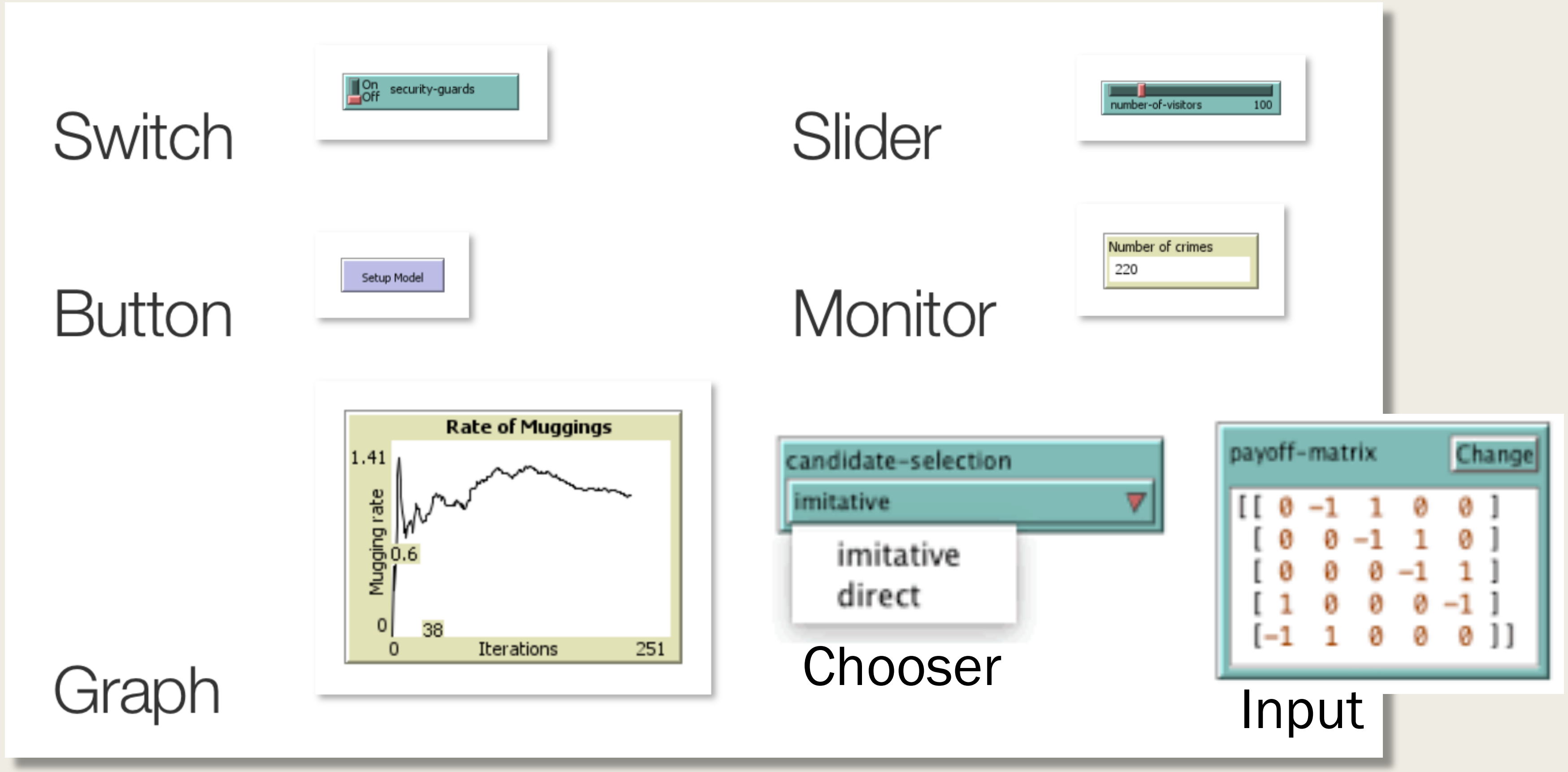
- No need to download the app (though some advanced functions are not available)
- Google ‘NetLogo home page’
- Netlogo home page > NetLogo Web
- https://www.netlogoweb.org/launch#https://www.netlogoweb.org/assets/modelslib/Sample%20Models/Social%20Science/Traffic%20Basic.nlogo
- Make sure the mode is ‘interactive’

NetLogo

- Components of a NetLogo model
  - *Interface (direct interact with the model using control knobs)*
  - *Command centre (direct interact with the model using command)*
  - *NetLogo code*
  - *Model Info (description, metadata, etc)*

Interface Components in NetLogo

- These controls are used to interact with the model and understand what it is doing. You add them yourself, depending on which ones are needed for the particular application



number-of-visitors 100

Number of crimes 220

candidate-selection imitative

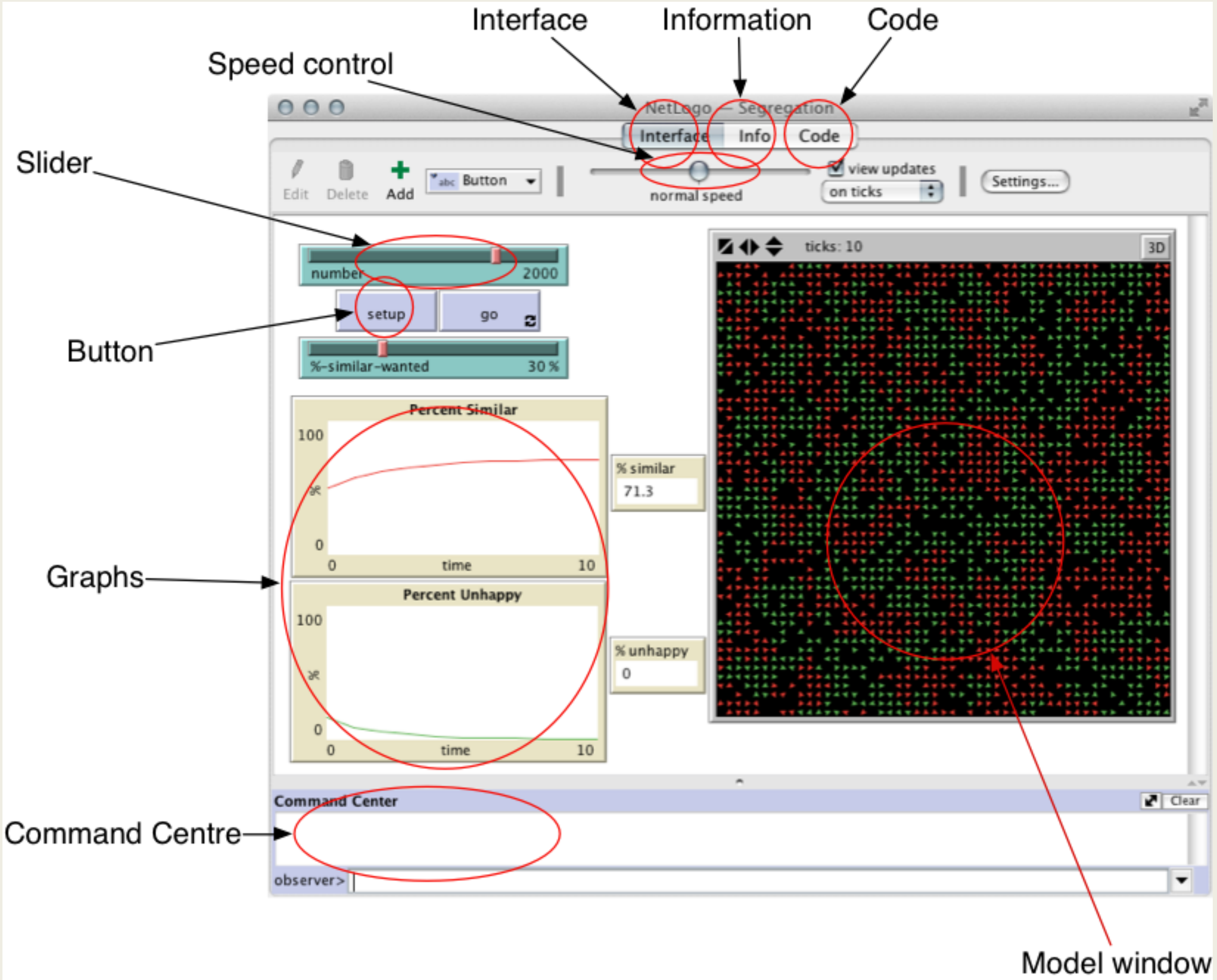
payoff-matrix

```
[[ 0 -1  1  0  0 ]
 [ 0  0 -1  1  0 ]
 [ 0  0  0 -1  1 ]
 [ 1  0  0  0 -1 ]
 [-1  1  0  0  0 ]]
```

Interface and Command Centre

There are two parts:

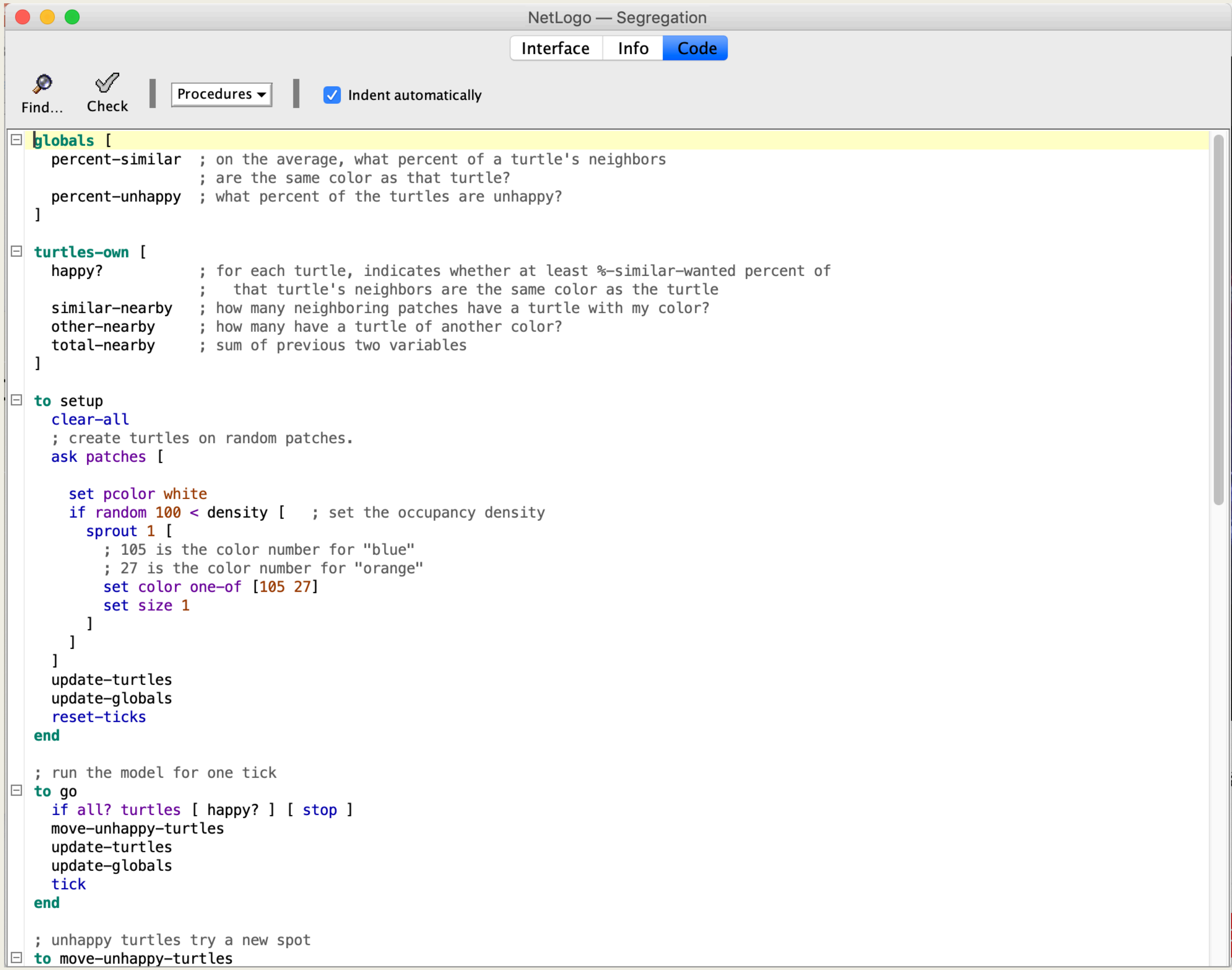
- the top part shows outputs from the model
- the bottom part (command centre) allows you to enter commands that are sent to the model.



Model code

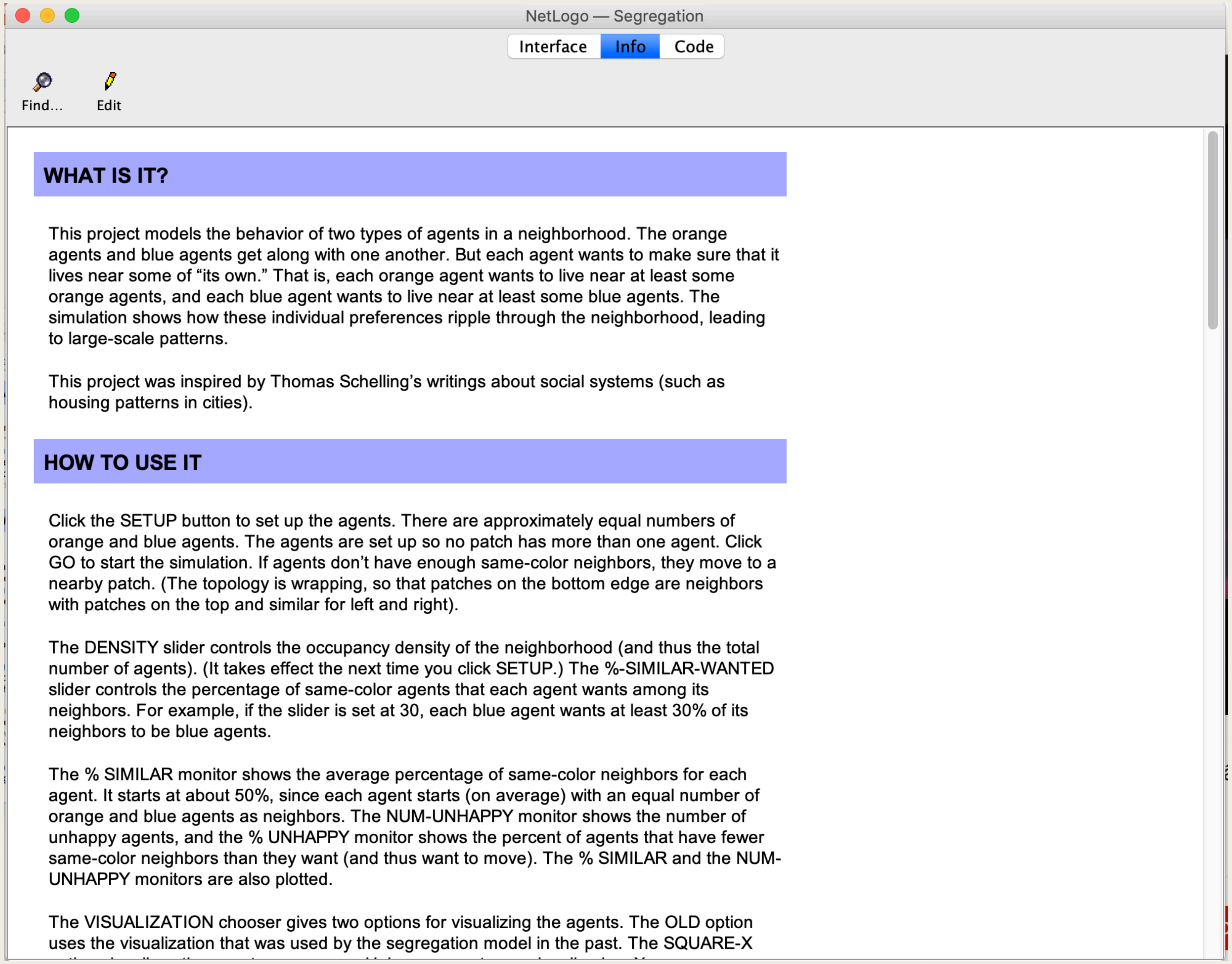
‘Netlogo programming guide’: https://ccl.northwestern.edu/netlogo/docs/programming.html

‘NetLogo Dictionary’ (particularly useful for syntax): https://ccl.northwestern.edu/netlogo/docs/dictionary.html



```
globals [
  percent-similar  ; on the average, what percent of a turtle's neighbors
                   ; are the same color as that turtle?
  percent-unhappy  ; what percent of the turtles are unhappy?
]

turtles-own [
  happy?           ; for each turtle, indicates whether at least %-similar-wanted percent of
                   ;   that turtle's neighbors are the same color as the turtle
  similar-nearby   ; how many neighboring patches have a turtle with my color?
  other-nearby     ; how many have a turtle of another color?
  total-nearby     ; sum of previous two variables
]

to setup
  clear-all
  ; create turtles on random patches.
  ask patches [

    set pcolor white
    if random 100 < density [   ; set the occupancy density
      sprout 1 [
        ; 105 is the color number for "blue"
        ; 27 is the color number for "orange"
        set color one-of [105 27]
        set size 1
      ]
    ]
  ]
  update-turtles
  update-globals
  reset-ticks
end

; run the model for one tick
to go
  if all? turtles [ happy? ] [ stop ]
  move-unhappy-turtles
  update-turtles
  update-globals
  tick
end

; unhappy turtles try a new spot
to move-unhappy-turtles
```

Model info : To help others (and your future self) understand the model
This is very important!



WHAT IS IT?

This project models the behavior of two types of agents in a neighborhood. The orange agents and blue agents get along with one another. But each agent wants to make sure that it lives near some of “its own.” That is, each orange agent wants to live near at least some orange agents, and each blue agent wants to live near at least some blue agents. The simulation shows how these individual preferences ripple through the neighborhood, leading to large-scale patterns.

This project was inspired by Thomas Schelling’s writings about social systems (such as housing patterns in cities).

HOW TO USE IT

Click the SETUP button to set up the agents. There are approximately equal numbers of orange and blue agents. The agents are set up so no patch has more than one agent. Click GO to start the simulation. If agents don’t have enough same-color neighbors, they move to a nearby patch. (The topology is wrapping, so that patches on the bottom edge are neighbors with patches on the top and similar for left and right).

The DENSITY slider controls the occupancy density of the neighborhood (and thus the total number of agents). (It takes effect the next time you click SETUP.) The %-SIMILAR-WANTED slider controls the percentage of same-color agents that each agent wants among its neighbors. For example, if the slider is set at 30, each blue agent wants at least 30% of its neighbors to be blue agents.

The % SIMILAR monitor shows the average percentage of same-color neighbors for each agent. It starts at about 50%, since each agent starts (on average) with an equal number of orange and blue agents as neighbors. The NUM-UNHAPPY monitor shows the number of unhappy agents, and the % UNHAPPY monitor shows the percent of agents that have fewer same-color neighbors than they want (and thus want to move). The % SIMILAR and the NUM-UNHAPPY monitors are also plotted.

The VISUALIZATION chooser gives two options for visualizing the agents. The OLD option uses the visualization that was used by the segregation model in the past. The SQUARE-X

NetLogo programming

- So what you will mostly do for this practical is
  - *change some of the agent behaviour parameters*
  - *examine the effects*

- The interested amongst you may want to play with the code!

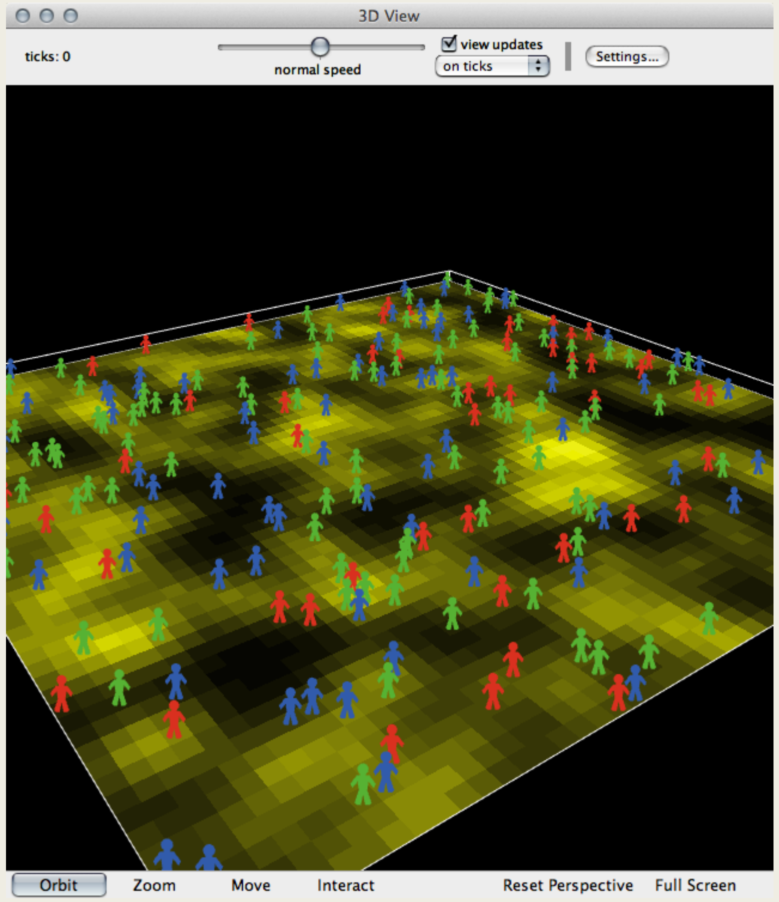
- So let’s have a look at the interface of a typical NetLogo programme

Turtles and Patches

- *Both are agents*
  - They have rules that determine their behaviour
  - They can interact with other agents
- *Main differences:*
  - Patches cannot move
  - You can create different types of 'turtle' (e.g. person, dog, cat, car, etc.)
- *Why turtles?*
  - 'Logo' language originally used to control robot turtles. It seems that the name 'turtle' has stuck..

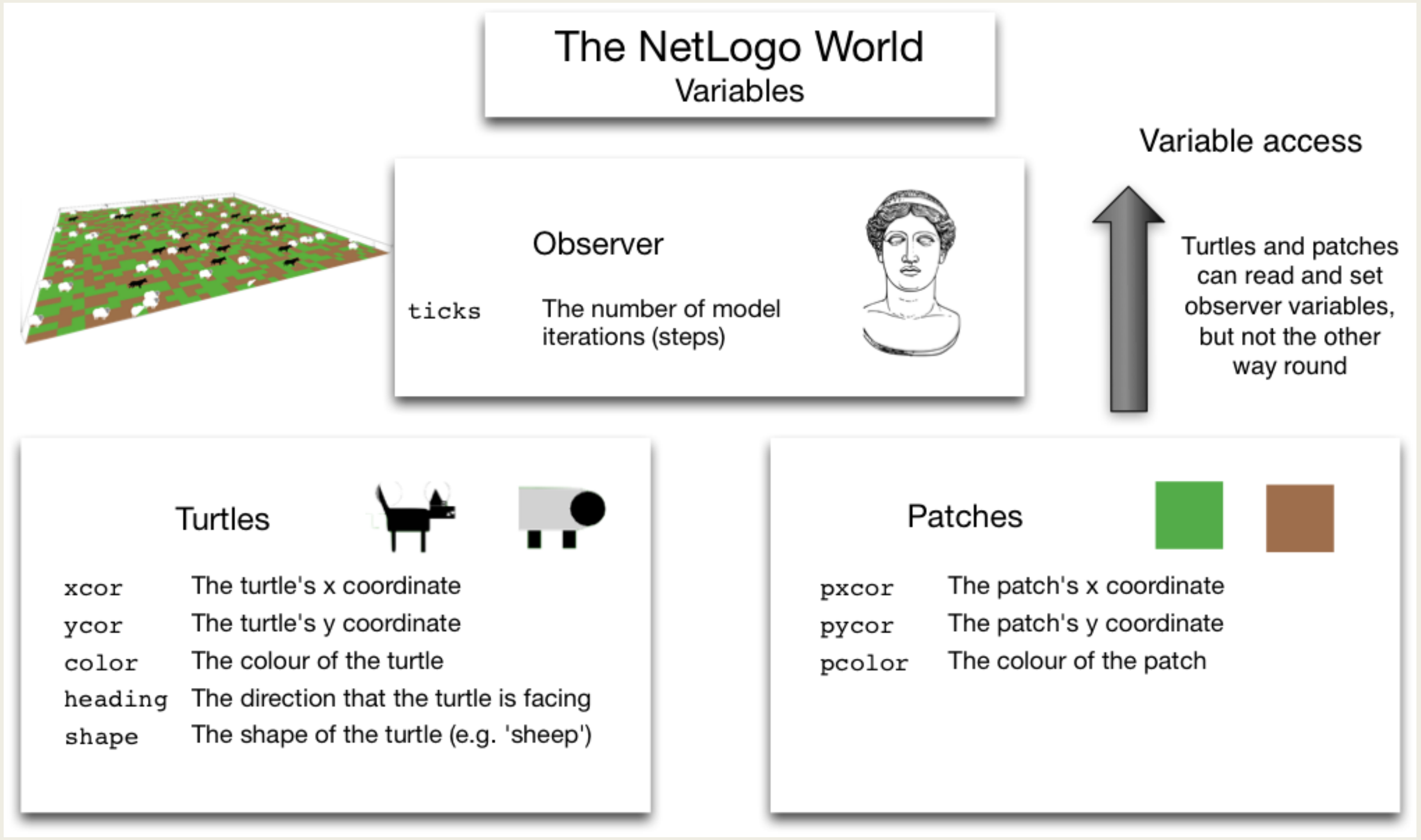
Observer

- The 'god' of a model
- Oversea everything that happens, give orders to turtles or patches, control other things like data input/output, virtual time, etc.



Built-In Variables

- There are some variables that NetLogo uses by default


The NetLogo World
Variables
Variable access
Turtles and patches can read and set observer variables, but not the other way round
Observer
Turtles
Patches

Observer

`ticks`	The number of model iterations (steps)

Turtles

`xcor`	The turtle's x coordinate
`ycor`	The turtle's y coordinate
`color`	The colour of the turtle
`heading`	The direction that the turtle is facing
`shape`	The shape of the turtle (e.g. 'sheep')

Patches

`pxcor`	The patch's x coordinate
`pycor`	The patch's y coordinate
`pcolor`	The colour of the patch

NetLogo variables

- In programming, variables are a way of storing information. E.g.
  - `my-name = ”Lex"`
  - `seconds-per-minute = 60`
  - `pi = 3.142`
  - `infected = True`
- Variables can belong to different objects in the model.
- Examples:
  - *Turtle variables: e.g.* `name, age, occupation, wealth, energy`
  - *Patch variables: e.g.* `height-above-sea, amount-of-grain, building-security, deprivation`
  - *Observer variables: e.g.* `total-wealth, weather, time-of-day, pi`
- Different objects can have different variable values

NetLogo commands

- Some examples

show "Hello World"	Prints something to the screen
set my-age 13	Sets the value of a variable
ask turtles [...]	Ask the turtles to do something
ask turtles [**set** color blue]	Asks the turtles to turn blue

Model Concepts: Brackets

- NetLogo uses both square `[ ]` and round `( )` brackets.
- Round brackets are used to set the *order of operations*. E.g.:
  - `2 + 3 × 4 = 14`
  - `(2 + 3) × 4 = 20`
- Square brackets are used to split up commands. E.g.:
  - `ask turtles [ ... ]`
  - *the `ask` command expects to find some more commands inside the brackets.*

Flow Control and Logic in NetLogo

- Usually, NetLogo will run through your code, one line after the other
- But! Sometimes there are two or more possibilities for what to do next
- `if` statements are one example:

```
... start here ...

if ( age < 18 )
  [ .. do something .. ]

if ( age > 18 )
  [ .. do something else .. ]

... now continue …
```

Flow Control Quiz

- The code below has been taken from the rules that drive the behaviour of a virtual person (or 'agent’).
- What will the person do when the age variable has the values of 10, 50, or 18?

```
if ( age < 18 )
  [ .. go to the cinema .. ]

if ( age > 18 )
  [ .. go to the pub .. ]

.. go to my friend's house ..
```

Age	Actions
10	?
50	?
18	?

Wolf Sheep Predation model

- Sample Models>Biology>Wolf Sheep Predation
- Press the setup button to initialise the model. You should see some sheep, wolves and green grass
- Choose model version to be '**sheep-wolves-grass**'
- The first command that we will experiment with is **show.** Type the following into the command centre:
  - *show "Hello World"*
- Try this again with some other text (don't forget to put quotes around the text).

Wolf Sheep Predation – Model info

- Explores the stability of predator-prey ecosystems
- An ecosystem is unstable if it tends to result in extinction for one or more species involved;
- It is stable if it tends to maintain itself over time, despite fluctuations in population sizes
- Sheep wander randomly around the landscape
- Wolves look for sheep to prey on

Wolf Sheep Predation – Model info cont’d

- Each step costs the wolves and sheep energy
- Wolves must eat sheep in order to replenish their energy - when they run out of energy they die
- The sheep must eat grass in order to maintain their energy - when they run out of energy they die
- Each wolf or sheep has a fixed probability of reproducing at each time step
- Once grass is eaten it will only regrow after a fixed amount of time

Count the number of sheep and wolves

- Now combine two commands: show and **count** as follows:
  - *Show count wolves*
  - *Show count sheep*
- What does this show you? Try this as well:
  - *show count patches*
- What does that tell you?

Count agents with certain attributes

- The **with** command: Try the following:
  - *show count patches with [pcolor = green]*
  - *show count patches with [pcolor = brown]*
  - *What is happening with those two commands?*
- Now run the model for a few seconds (press 'Go' to start it, then press it again to stop). Repeat the two commands above. What result do you get now?
- The with command actually needs two pieces of information. It needs a group of agents or patches on the left, and a test on the right (in this case we test the colour of the patches).
- You can find more information about all the commands available in 'NetLogo Dictionary' (look for the 'with' command)

Global/observer variables

- In NetLogo, it is possible to save values in things called 'variables' (for info, see the documentation on variables). Different parts of the model can check the values of the variables to decide what they should do.
- For example, in the Wolf Sheep Predation model, each of the sliders changes the value of a variable. So, if you move the wolf-gain-from-food slider from 20 to 40, that will change the value of a variable called wolf-gain-from-food.
- Each time a wolf eats a sheep it checks the value of that variable to decide how much new 'energy' it will gain. In this case, it will now get 40 units rather than only 20.
- Now change the values of wolf-gain-from-food from to 40, and run the model again. How are the result different? Change the value of some other variables using the sliders. How are the result different?

Turtle and patch variable

- Individual turtles and patches can also have their own variables; this is one of the ways that models can account for heterogeneity (we'll cover this in more detail in later lectures).
- For example, in the Wolf Sheep Predation model, the wolves and sheep have a variable called energy. This records how much energy each individual wolf or sheep will have at any given time. If this reaches zero, then they will die.
- Select ‘show-energy?’ to see the energy value of individual sheep and wolf

Set variable values (the set command)

- Try the following command **each at a time,** and press setup. Then let the model run for 10-20 seconds. What do you observe? Does the results change after setting the variable values?
  - *set sheep-gain-from-food 12*
  - *set sheep-gain-from-food 50*
  - *set initial-number-wolves 100*
  - *set wolf-reproduce 10*
- Now set different variable values yourself, and run the model for 10-20 seconds. How are the result different?

Think about the Wolf Sheep Predation model

- Notice that increases and decreases in the sizes of each population are related. In what way are they related?
- What is the explanation for this?
- What eventually happens?
- Why do you suppose that some variations of the model might be stable while others are not?

Virus model

- Sample Models>Biology/Virus
- This model simulates the transmission and perpetuation of a virus in a human population.
- In the beginning, there are 150 people, of which 10 are infected
- People move randomly about
- People have three states:
  - *Healthy but susceptible to infection (green)*
  - *Sick and infectious (red)*
  - *Healthy and immune (grey)*
- People may get infected when they come into contact with each other

Virus model – cont’d

- How likely does a person get infected when come into contact with an infected person?
  - *Controlled by the (global variable)* ***infectiousness*** *slider*
- What is the chance that a person will die from the virus?
  - *Controlled by the (global variable)* ***chance-recover*** *slider*
- How long is a person infected before they either recover or die?
  - *Controlled by the (global variable)* ***duration*** *slider*

The ‘ask’ command

- Use the **ask** command to send commands to turtles or patches *directly*
- In the command centre, write: ask patches [set pcolor blue]
- The ask command expects two inputs.
  - *The first input is the turtles or patches that we want to do something to.*
  - *The second input is a command, or a number of commands, that will be sent to those turtles or patches.*

Virus model – ask command

- Now try this command: ask turtles [set color brown]
- Now try two commands to set the x and y coordinates of the turtles:
- ask turtles [set xcor 1 set ycor 5]
- Where have all the turtles moved to? Try this as well
- ask turtles [set xcor -10 set ycor -5]

Using ask and with

- First reset the model by pressing setup
- Execute the following command, that will have an affect only on the people that have been infected by a virus
  - *ask turtles with [sick? = true] [set color brown]*
- Here, instead of giving all the turtles to the ask command, we give it the group of turtles who have a value of true stored in their variable called ‘sick’ (this is a special variable created specifically for the 'virus' model)

Using ask and with

- Now try it again, but change the colour of those who aren't ill
  - *ask turtles with [sick? = false] [set color blue]*
- Each person also has a variable that is their age. So we can use ask to run commands on people of different ages:
  - ask turtles with [age > 20] [set color yellow]
- You can also ask to run commands on people who are older than 20 AND sick
  - ask turtles with [age > 20 and *sick? = false*] [set color orange]
- Try some other ask command yourself

Move people's location using ask

- Move the 'people' slider from 150 down to 10. This will reduce the number of people in the model and make it easier to see the impact that the following commands will have.
- Setup the model again. You should now see only a few (10) people in the world.
- issue the following command:
  - *ask turtles [forward 1]*
- What is happening? What happens if a negative number is sent to the forward command?
  - *ask turtles [forward -1]*

Move people's location using ask

- Now try these commands and see what happens (they have to be entered into the Command Centre one by one):
  - *ask turtles [facexy 0 0]*
  - *ask turtles [forward 1]*
  - *ask turtles [forward 1]*
  - *ask turtles [forward 5]*
- Now make the turtles move around as you wish

Think about the Virus model

- Did you notice the initial outbreak?
  - *Often there will initially be an explosion of infection since no one in the population is immune*
- What makes a ‘successful’ virus?
  - *Viruses that are too successful (high infectiousness) at first (infecting almost everyone) may not survive in the long term. Since everyone infected generally dies or becomes immune as a result*
  - *Ebola has a very short duration, a very high infectiousness value, and an extremely low recovery rate. How successful is it? Set the sliders appropriately and watch what happens.*
  - *The HIV AIDS virus, has an extremely long duration, an extremely low recovery rate, but an extremely low infectiousness value. How successful is it?*

Exercise: Virus model

- Write a command to change the colour of all of the patches to brown.
- Write a command that will ask all agents to move forward one step.
- Write a command that will ask all blue turtles to move forward five steps. (Hint: use a combination of ask and with).
- Write a command that will ask the turtles to change the colour of the patch that they are standing on to blue. (Hint: ‘patch-here’ reports the patch under the turtle)
- Write a command that will ask all of the red turtles to change the colour of the patch that they are standing on to orange. Hint: remember that turtle colour is stored in a variable called ‘color’ and patch colour is stored in a variable called ‘pcolor’.