Network Science with Netlogo Tutorial

By Tom Brughmans

First version: March 2016

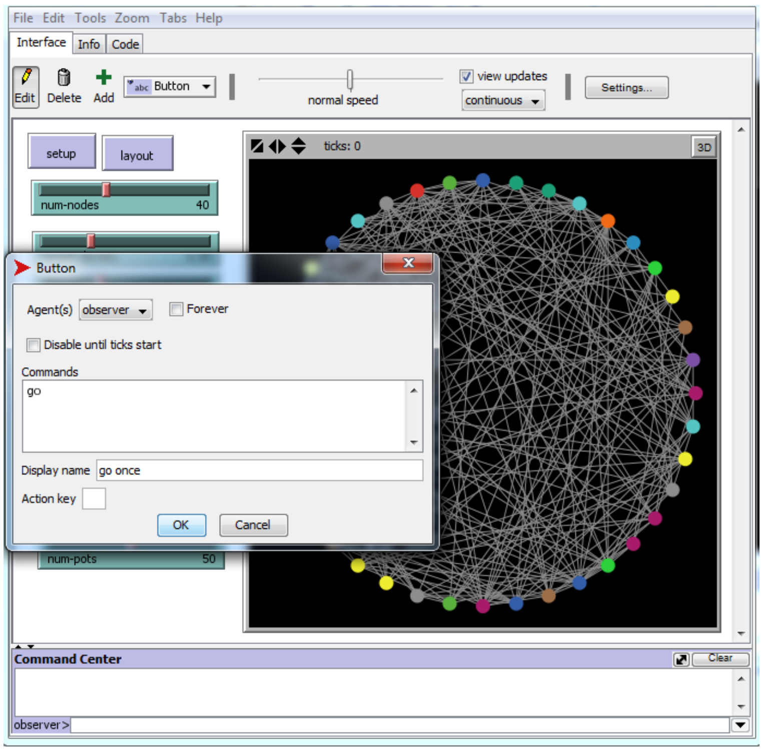
This version created 13/09/2018

Netlogo version used: 6.0.1

Extension used: nw (pre-packaged with Netlogo 6.0.1)
https://github.com/NetLogo/NW-Extension

Cite this tutorial as:

Brughmans, T. (2016). Network Science with Netlogo Tutorial,
https://archaeologicalnetworks.wordpress.com/resources/#netlogo .



Button

Agent(s) observer

Forever

Disable until ticks start

Commands

```
go
```

Display name go once

Action key

OK Cancel

1. Introduction

This tutorial provides an introduction to creating agent-based network models with Netlogo. By working through this tutorial you will learn how to create nodes, create edges, perform layouts introduce probability in edge creation, create a trade process working on the network and how to derive network measures.

2. Conventions, tips and assumed knowledge

This tutorial assumes basic knowledge of Netlogo and of simulation. It is recommended to walk through the introductory tutorials on the Netlogo website or the tutorial on Netlogo for archaeologists on the Simulating Complexity blog:

https://simulatingcomplexity.files.wordpress.com/2014/07/dispersal_tutorial.pdf

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

This tutorial will also refer to some network science jargon, concepts and techniques. You can get a basic definition of all of these from the glossary on my blog:

https://archaeologicalnetworks.wordpress.com/resources/#glossary

Good introductions to network science include the following:

> Brandes, U., Robins, G., McCranie, A., & Wasserman, S. 2013. What is network science? Network Science 1(01): p.1–15.
>
> Newman, M.E.J., 2010. Networks: an introduction, Oxford: Oxford University Press.

Code written in this tutorial will be formatted as in the following example:

```
breed [nodes node]
```

Save your project! Do this regularly and use multiple versions throughout the tutorial so that you can fall back on an earlier version at any time. I will remind you regularly throughout the tutorial to save your project.

3. Download and install Netlogo

Netlogo is open source software and can be downloaded free of charge for Windows, Mac OS X and Linux.

Go to https://ccl.northwestern.edu/netlogo/download.shtml .

Download the Netlogo installer (this tutorial uses version 6.0.1).

Run the installer and install Netlogo.

4. Netlogo resources, manual and interface

Netlogo has great documentation about all its features and code in its user manual: https://ccl.northwestern.edu/netlogo/docs/

This manual includes tutorials, a reference to the software functions, a dictionary to its programming language, documentation of its extensions and much more.

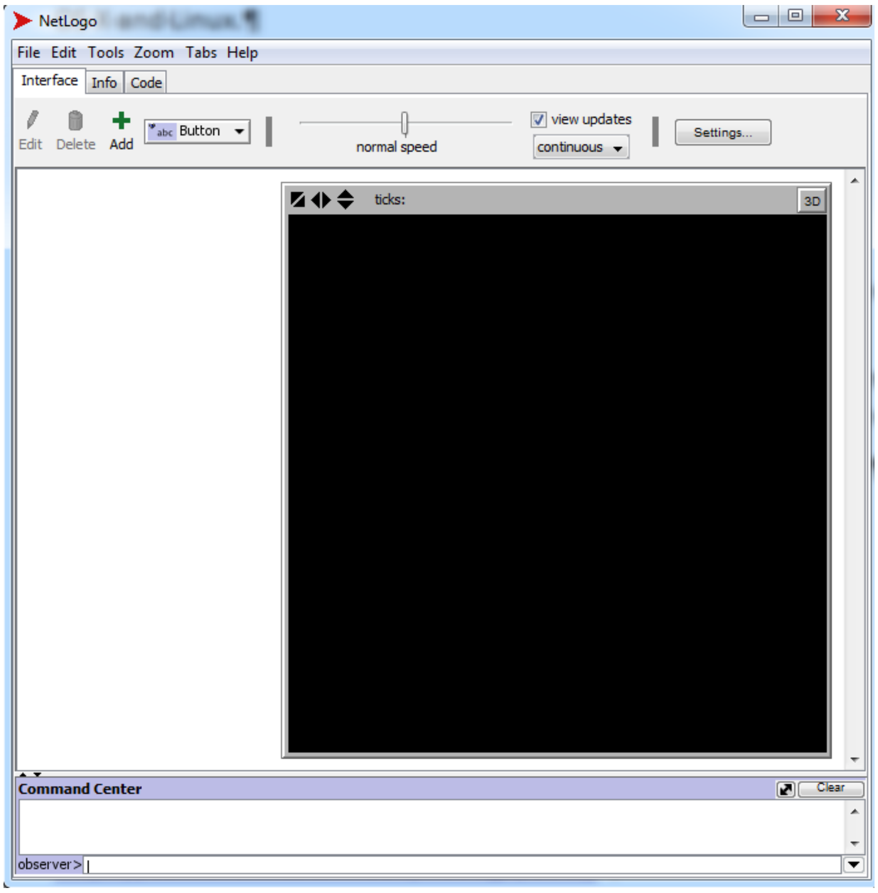
Additional external resources can be found on the resources page:

https://ccl.northwestern.edu/netlogo/resources.shtml

A detailed reference to Netlogo's Interface can be found here: https://ccl.northwestern.edu/netlogo/docs/

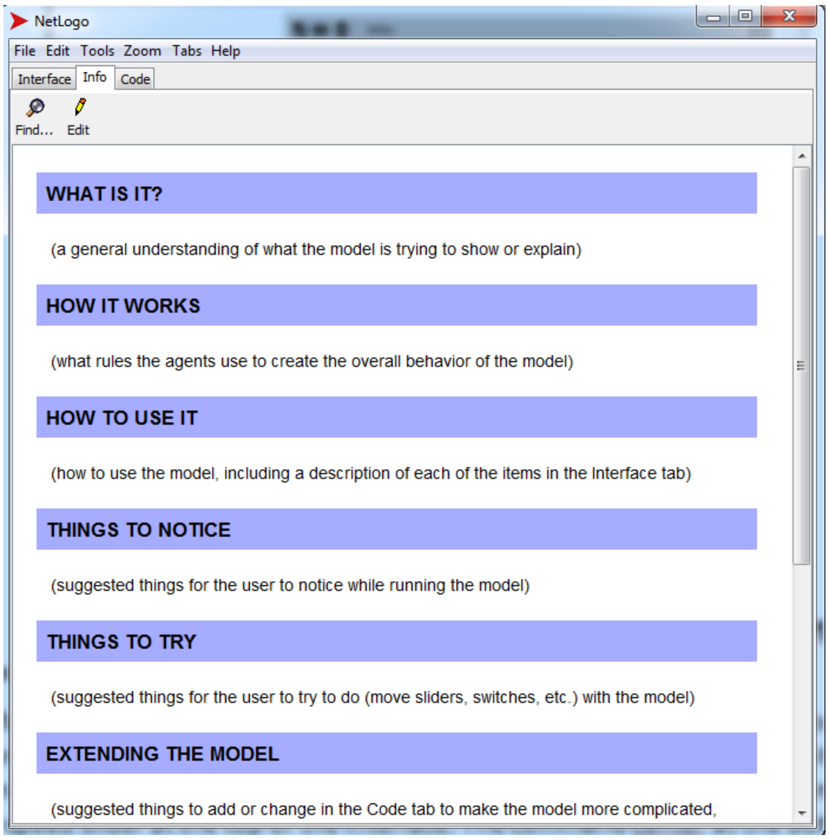
This tutorial will only give a very brief introduction to the key elements of the Netlogo interface you will be using throughout the tutorial.

When you open Netlogo it should look something like this:



It has three tabs: Interface, Info, Code.

The **Interface tab** is where you watch your model run. Throughout the tutorial you will add buttons and sliders to control the variables of your model, and you will add monitors and plots to inspect what your model is doing. You can speed up or slow down the simulation using the speed slider at the top of the interface. The command center at the bottom will display messages you ask the model to produce, and you can also use it to give to commands to the model from the interface tab.



WHAT IS IT?

(a general understanding of what the model is trying to show or explain)

HOW IT WORKS

(what rules the agents use to create the overall behavior of the model)

HOW TO USE IT

(how to use the model, including a description of each of the items in the Interface tab)

THINGS TO NOTICE

(suggested things for the user to notice while running the model)

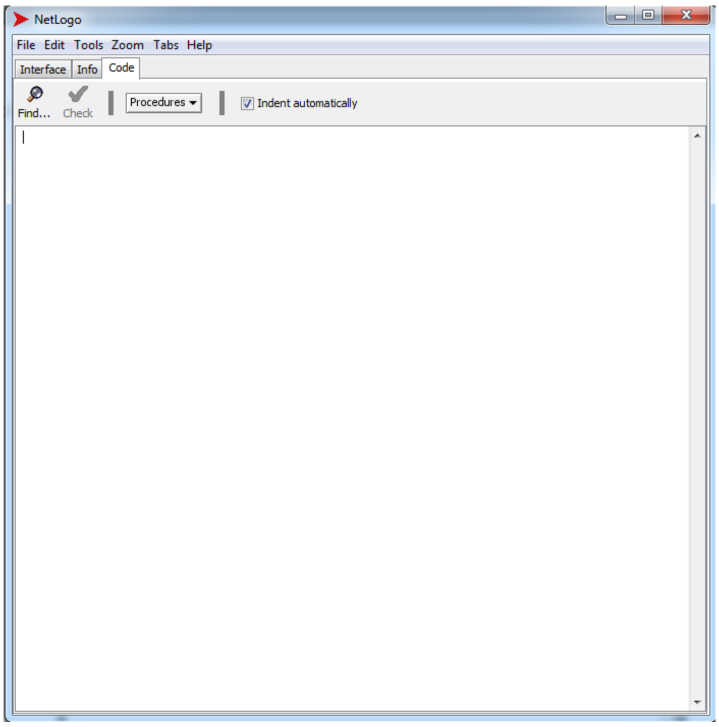
THINGS TO TRY

(suggested things for the user to try to do (move sliders, switches, etc.) with the model)

EXTENDING THE MODEL

(suggested things to add or change in the Code tab to make the model more complicated,

The **Info tab** is where you describe your model using a standardized set of questions. Adding this information when you share your model with other is crucial to enable them to work with your model. We will not work with the Info tab in this tutorial.



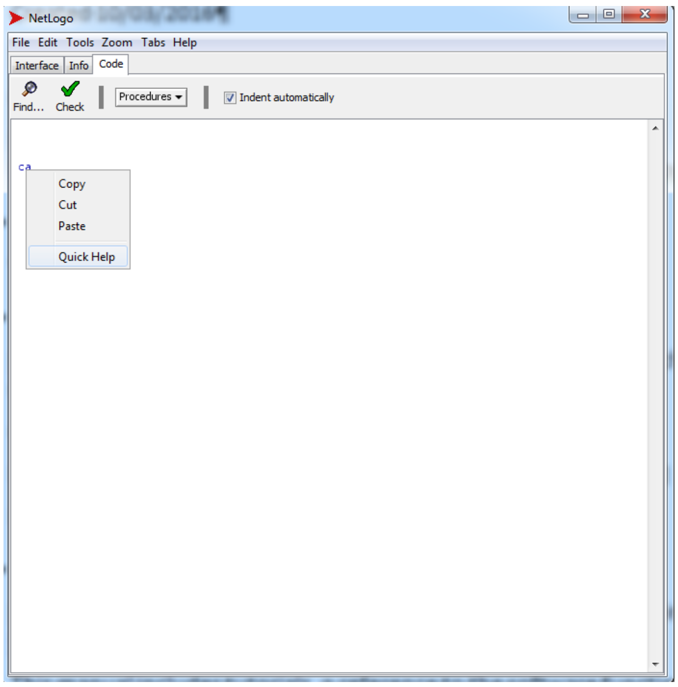
The **Code tab** is where you write and store the code for the model. In this tutorial we will be mainly working in the Code tab. A useful feature is the Check function at the top of this tab: click this to let Netlogo check your code for errors. If it finds errors then you will be guided to the error and asked to resolve it before you can continue, if it does not find errors then you can proceed with coding or viewing your model in action. NOTE: this error checker only checks whether the primitives used and the order of the code comply with the Netlogo rules (i.e. the code's vocabulary and grammar). It will not check whether the code does what you want it to do, so getting no errors is no guarantee that the code works the way it should or the way you think it does. The error checker will always need to be used alongside other error checking techniques, like reporting variable values and checking them against expectations, or testing submodels independently.

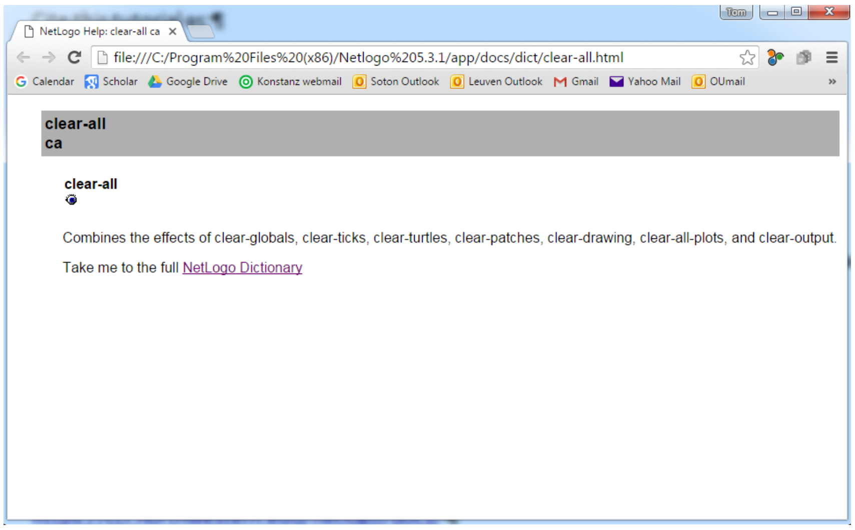
5. Netlogo dictionary

A crucial resource when coding in Netlogo is its dictionary: https://ccl.northwestern.edu/netlogo/docs/

For this tutorial, you will find the section on 'Links' in the Netlogo dictionary particularly useful, as well as the documentation of the 'nw' extension.

You can get direct access (even offline) to the entry about a particular primitive by right-clicking it and selecting **Quick Help**.




NetLogo Help: clear-all ca
file:///C:/Program%20Files%20(x86)/Netlogo%205.3.1/app/docs/dict/clear-all.html
Calendar
Scholar
Google Drive
Konstanz webmail
Soton Outlook
Leuven Outlook
Gmail
Yahoo Mail
OUmail

clear-all
ca

clear-all

Combines the effects of clear-globals, clear-ticks, clear-turtles, clear-patches, clear-drawing, clear-all-plots, and clear-output.

Take me to the full NetLogo Dictionary

6. Create nodes

Networks consist of at least two things: a set of nodes and a set of edges (network science jargon for an undirected relationship). Nodes and edges in Netlogo are a type of agents, or turtles in Netlogo jargon. We will first create the nodes in Netlogo.

To avoid using the confusing term 'turtles' to refer to our nodes, we will create a new **breed** of agents called 'nodes'. A breed is a set of agents that is given a specific name and can be commanded using specific variables given to this agentset.

To create a breed, write at the top of the **Code tab**:

```
breed [nodes node]
```

If you click the check box now you will notice that there are no errors, so we can continue to code.

Save your model!

To create a number of nodes and visualise them in the Interface we will create a setup procedure. Add the following below the breed command in the **Code tab**:

```
to setup
  clear-all
  reset-ticks
end
```

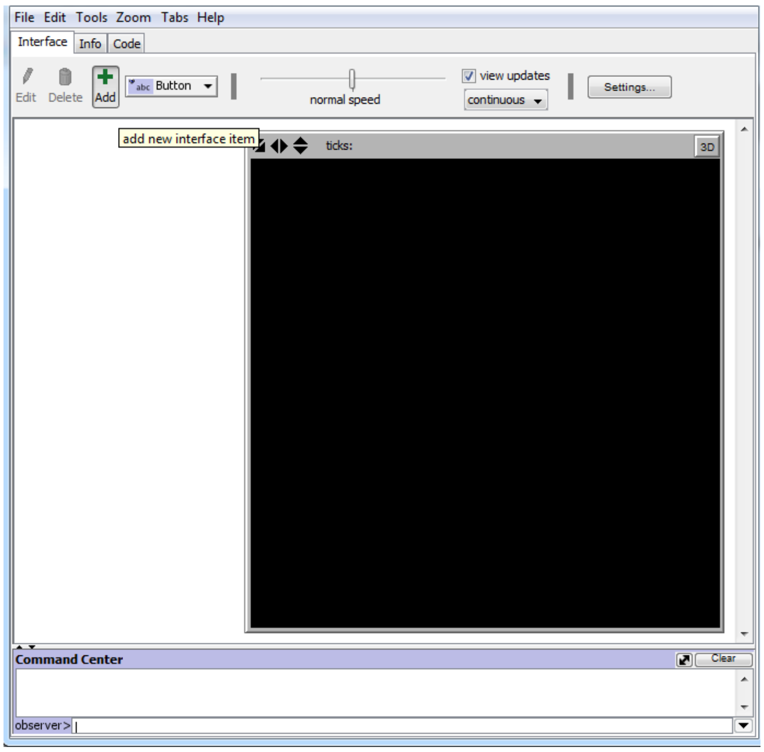
This is a standard setup procedure in Netlogo that resets the Netlogo interface of any settings from previous simulations and resets the ticks to 0 to initiate a new simulation (Netlogo jargon for a timestep). Setting up a simulation means we are ready to run the simulation.

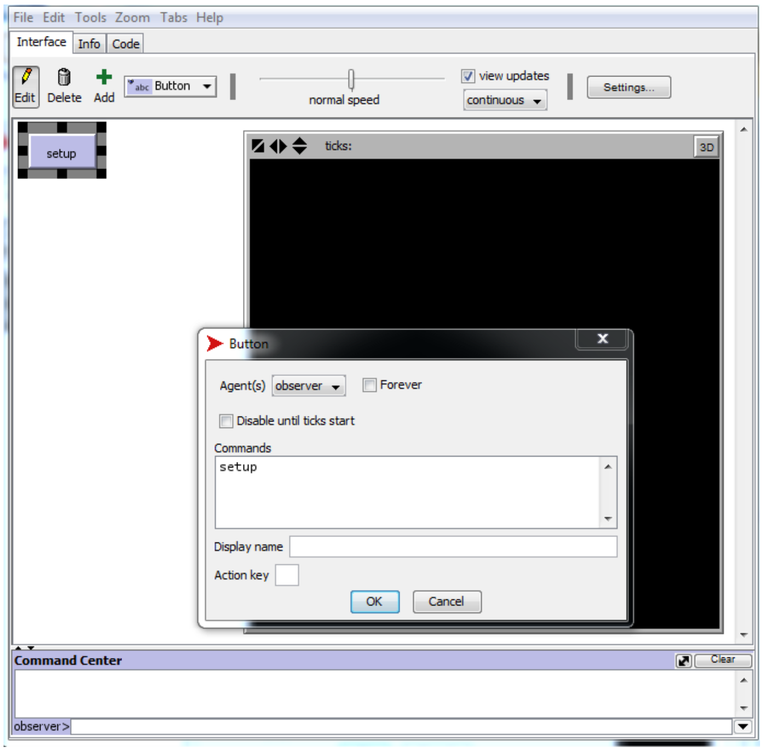
Now we will add commands to this standard setup procedure to create our nodes. Expand the procedure to the following:

```
to setup
  clear-all
  set-default-shape nodes "circle"
  create-nodes 50
  reset-ticks
end
```

Our setup procedure will now create 50 nodes, and they will be represented as circles. To see this in action, we will need to add a button in the Interface tab. Do the following in the **Interface tab**:

- Make sure the drop down box next to the green plus sign has ‘Button’ selected.
- Click the green plus sign called ‘Add’ (see figure below). Your cursor will become a big plus sign.
- Click anywhere in the white space next to the black interface window. A button will appear and a new window will open where you can determine what this button will be used for.
- Write `setup` in the **commands box** of this new window and click **OK** (see figure below).



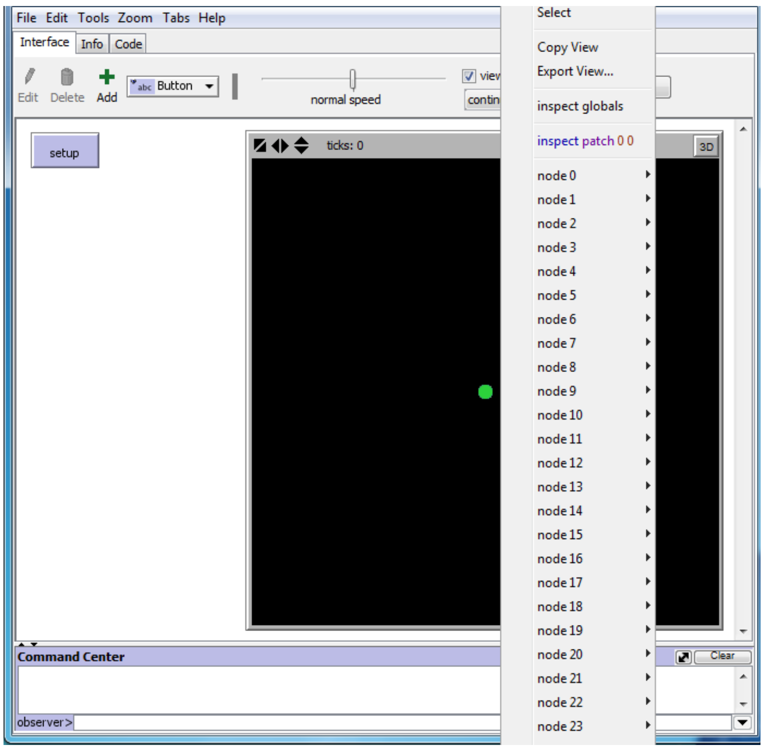


Agent(s) observer

Commands
```
setup
```

You just created a button called ‘setup’. When you press this button the setup procedure we wrote in the Code tab will be executed. Try it!

You will see a circle being created at the very centre of the interface window. Although it looks like a single node, we have actually created 50 nodes that are all placed on top of each other in the centre of the window, because we did not specify where they should be placed. **Right-click the node** and you will notice there are 50 overlapping nodes as in this figure:



We will solve this issue of node placement soon, but first we should give ourselves more control over the number of nodes. Now we have ‘hard-coded’ the number of nodes in our Code tab, which makes it a lot of work to change the number of nodes when exploring our final model and running simulation experiments.

To overcome this issue we will now add a variable that controls the number of nodes. In the Code tab, replace `create-nodes 50` with:

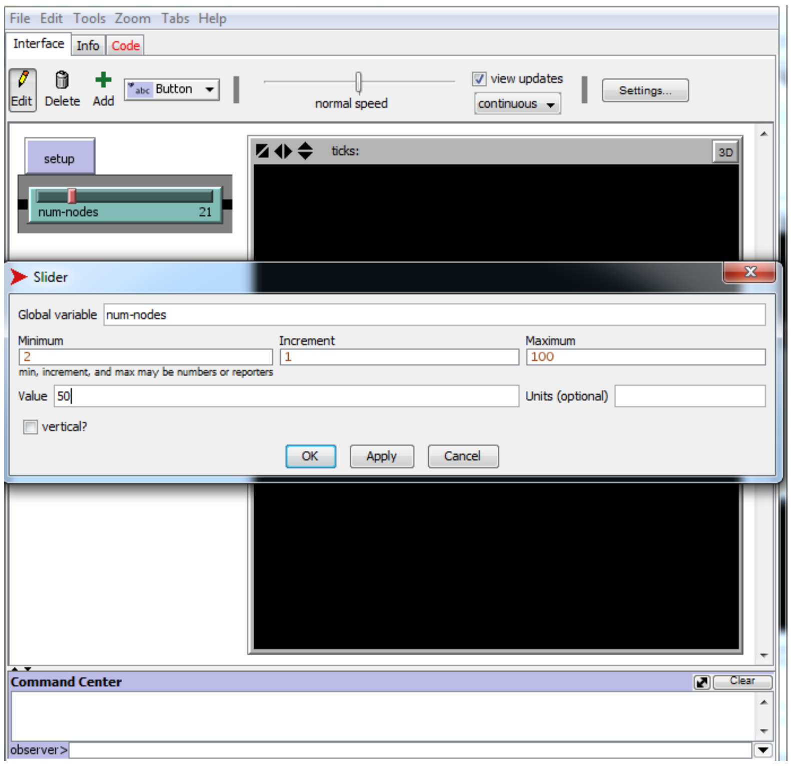
```
create-nodes num-nodes
```

We have now create a variable called `num-nodes` which will determine how many nodes are created by `create-nodes`. When you click the error checker, you will notice that Netlogo has found an error: “Nothing named NUM-NODES has been defined”. This is because we have written a variable that has not been defined earlier in the code or on the interface. We want to control the variable in the interface with a slider so that we can easily change it when running our simulation.

To overcome this error and control this variable in the **Interface tab** do the following:

- Click the dropdown menu next to the ‘Add’ button.
- Select ‘Slider’. Your cursor will change into a large plus sign.
- Click in the white space below the ‘setup’ button. A slider will be created and a new window will open where you can determine what this slider will be used for.

- We will use this slider to control the variable num-nodes, so write num-nodes in the box 'global variable'. Set the 'minimum' to 2 and click OK (see figure below).



Global variable: num-nodes
Minimum: 2
Increment: 1
Maximum: 100
min, increment, and max may be numbers or reporters
Value: 50
Units (optional):
vertical? ☐

You have now created a slider where you can modify how many nodes you create. You will also notice that the error message on the code tab no longer appears.

But the nodes are still overlapping, so let's take care of that right now!

Save your model!

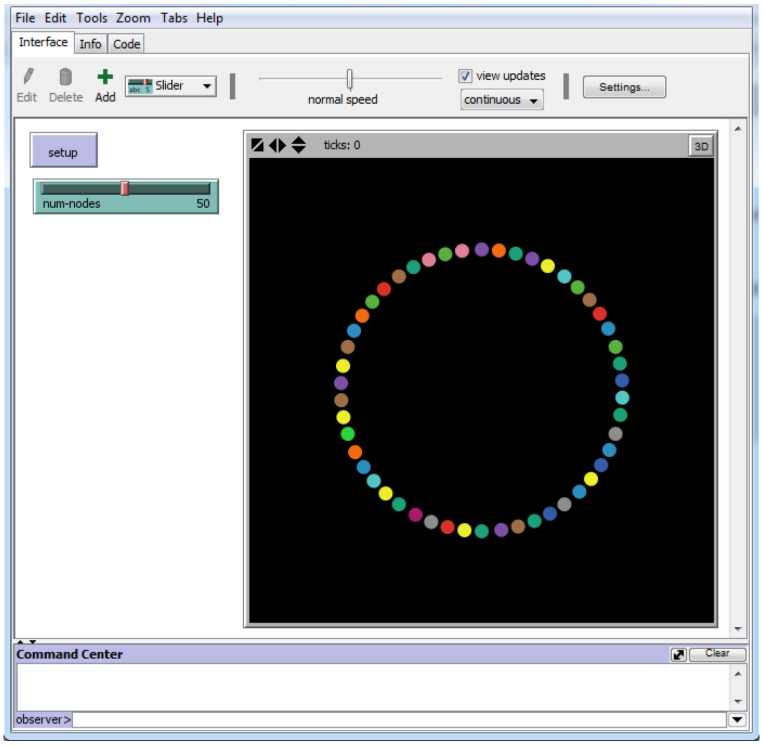
7. Circular layout

You can place your nodes in many different ways depending on what feature of the network you wish to emphasise, as we will see later in the tutorial. But since we only have nodes and no edges yet, let's just place all nodes along a circular layout to avoid overlapping nodes.

In the **Code tab** add the following after the line creating the nodes:

```
    layout-circle nodes 10
```

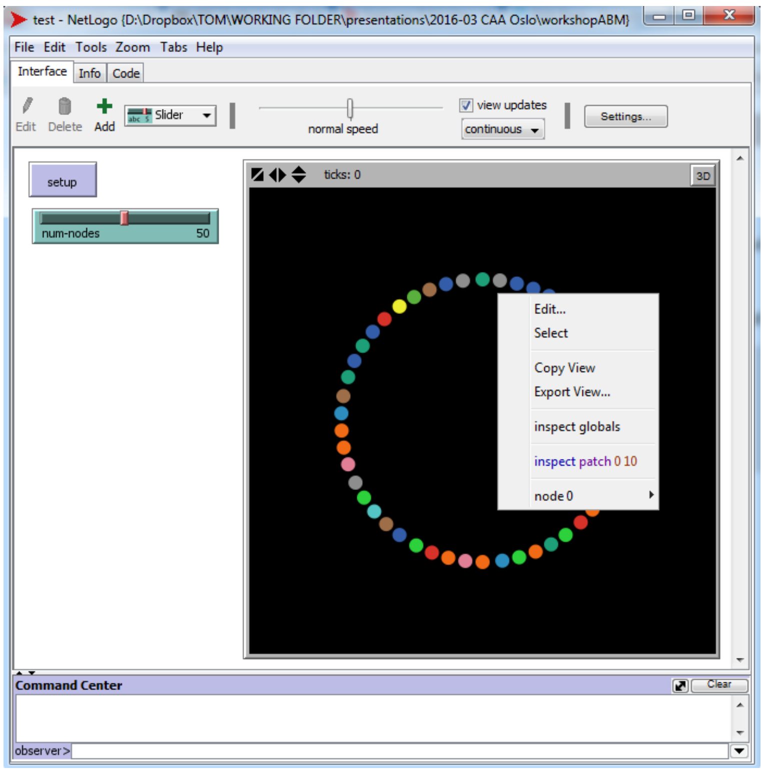
We have used a Netlogo command that positions nodes along a circle with a radius of 10 in a random order. If you go to the Interface tab now and click the setup button then your nodes will be positioned along a circle as in this figure:



However, you might want more control over the order in which the nodes are placed, and not just let them be placed in a random order. To achieve this, rework the layout line in the **Code tab** to read like this:

```
        layout-circle sort nodes 10
```

If you click the setup button in the Interface tab now, you will get a circular positioning of all nodes, but they are ordered according to their ‘Who’ number (the unique ID of a turtle assigned by Netlogo when it is created) clockwise starting from the top middle node in the circle. You can check this by right clicking the nodes in order and you will notice that the order of nodes is 0, 1, 2, 3, …. (see figure below).

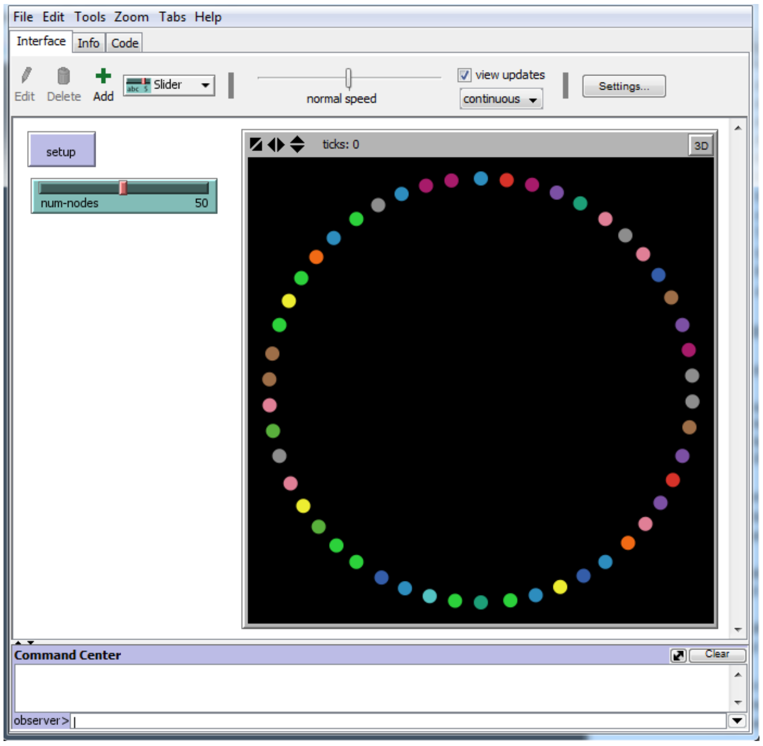


However, once again we hard-coded something that is not very flexible if the model environment changes: the circle radius of 10. It would be far better if this radius was defined by the resolution of the interface window (i.e. the model's 'World' settings expressed in coordinates, which can be viewed by right-clicking the interface window and selecting 'Edit').

In the **Code tab** change the layout line to the following:

```
    layout-circle sort nodes max-pxcor - 1
```

When you click the setup button now on the Interface tab the circular layout will have a radius equal to the radius of the interface window (i.e. the World's max-pxcor) minus 1 to prevent it from being too close to the edges of the window:



Save your model!

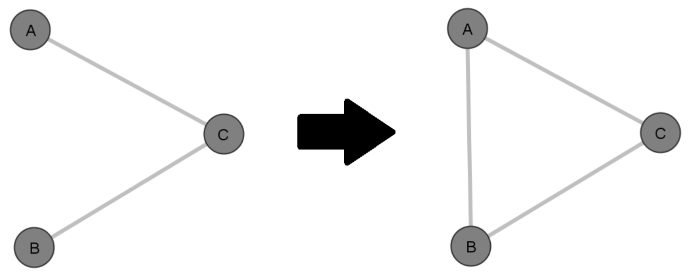
8. Network creation procedures: why do you create a network model?

Now that we have a set of nodes we can create a set of edges connecting the nodes.

The way in which nodes are connected into a network will depend on your research context, and ultimately will be rooted in your reasons for deciding to build a network model in the first place. Edges could be randomly connected, edges can be informed by empirical observations, or the network structure could represent a hypothesised social/spatial/other network structure.

In this tutorial we will randomly create a network structure, using the Erdős–Rényi random graph model (from now on referred to as ER model): a network constructed by connecting randomly selected pairs of nodes with a certain probability independent from every other edge. https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model

This is the most common way of randomly creating networks. Such networks are very useful for comparison with hypothesised social processes, i.e. processes creating edges with probabilities dependent on the existence of other edges. For example, a common hypothesis for social networks of friendship is the idea that a pair of unrelated individuals has a higher probability of becoming friends (i.e. of creating an edge between a pair of nodes) if they have a friend in common (i.e. the probability of the new edge is dependent on the existence of other edges). Nodes A and B in the figure below have node C as a mutual friend, so according to this hypothesis the edge between A and B will be more likely to appear:



We can create simulation models representing this hypothesised network creation process and the random process, and compare how the resulting networks differ. Can observed friendship relationships be reproduced through a random process? Can they be better reproduced through another hypothesised process? What are the structural features of the networks arising from the hypothesis?

Deciding how to connect nodes through edges in your network will depend entirely on your research aims and context. To think about what might be the most appropriate way of creating edges in your context you could ask yourself the following questions:

- Why did I decide to make a network model?
- What are the nodes in my network?
- What are the edges in my network?
- Are nodes and edges taken from empirical observations?
- Will the network model represent a hypothetical scenario that will be compared with empirical observations?
- Why are relationships important in my research context?
- How are relationships dependent on each other?
- What kinds of configurations or patterns of edges are important?
- What are my hypotheses about how edges are created?

9. Create edges

Relationships in Netlogo are called ‘links’ and they can be used in a similar way as ‘turtles’. They can be either undirected in which case network scientists refer to them as ‘edges’, or they can be directed which are commonly called ‘arcs’.

We will create a new breed of undirected ‘links’ called ‘edges’. In the **Code tab** below the previous breed add the following:

```
undirected-link-breed [edges edge]
```

Now that we have created a breed we can start implementing the procedure to create an ER random graph model. We will do this in several steps, slowly building up the lines of code.

In the setup procedure, add a new procedure to connect nodes after the layout of the nodes:

```
connect-nodes
```

Your model should now look like this:

```
breed [nodes node]
undirected-link-breed [edges edge]

to setup
  clear-all
  set-default-shape nodes "circle"
  create-nodes num-nodes
  layout-circle sort nodes max-pxcor - 1
  connect-nodes
  reset-ticks
end
```

When you click the error checker it will give you an error saying ‘connect-nodes’ has not been defined. We will now add a new procedure called `connect-nodes` below the setup procedure:

```
to connect-nodes
end
```

This will resolve the error message.

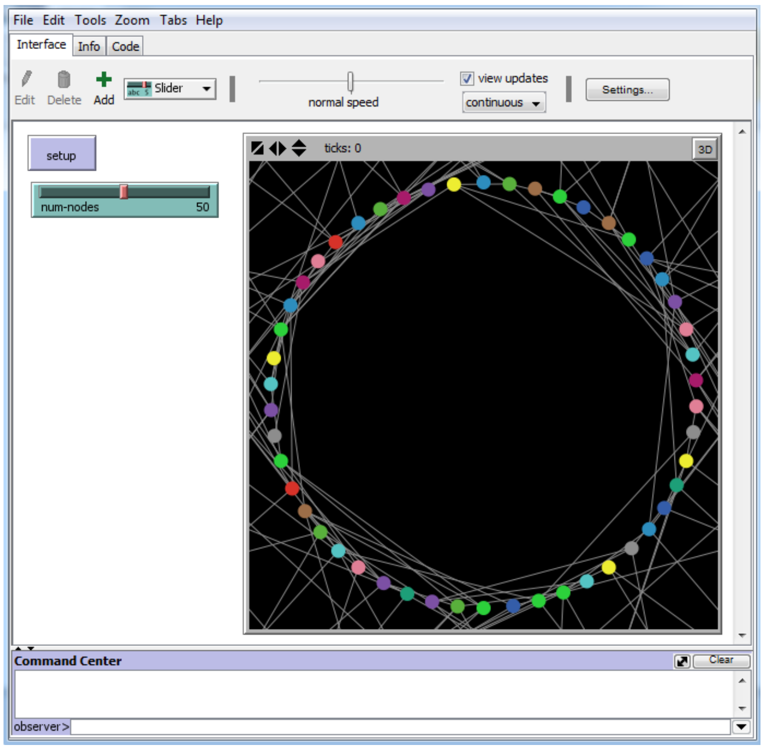
As a first step we will add the following command in this new procedure:

```
to connect-nodes
  repeat 100
  [
    ask one-of nodes [create-edge-with one-of other nodes]
  ]
end
```

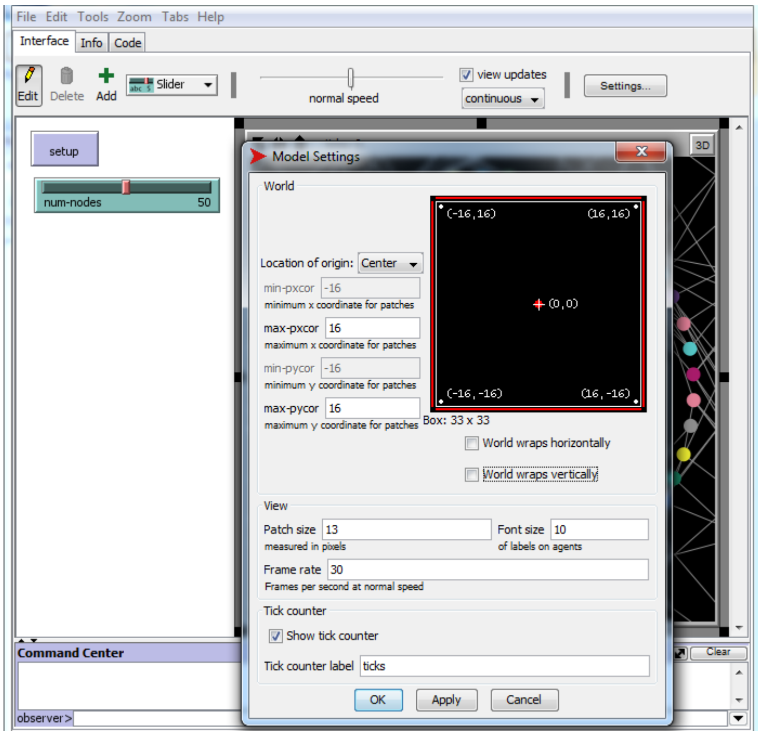
This code will select 100 nodes randomly in turn and each node will be asked to create an edge with another randomly selected node.

Note that the normal command for creating links in Netlogo is `create-link-with`. But because we have created a breed called edges, we can modify this to `create-edge-with`. Have a look at the Netlogo dictionary for more information: https://ccl.northwestern.edu/netlogo/docs/ .

However, now when you click the setup button on the Interface tab you will get a weird network where edges seem to cross the borders of the interface window:



This happens because by default the Netlogo ‘World’ will wrap around the horizontal and vertical edges. This does not make much sense when making a network model! At the very least it is not helpful for visual exploration. To avoid the edges from crossing these boundaries, right-click the interface window, select **Edit**, and in the Model settings window uncheck the boxes for **‘World wraps horizontally’** and **‘World wraps vertically’**:



Model Settings

World

Location of origin: Center

min-pxcor -16
minimum x coordinate for patches

max-pxcor 16
maximum x coordinate for patches

min-pycor -16
minimum y coordinate for patches

max-pycor 16
maximum y coordinate for patches

(-16,16) (16,16)
(0,0)
(-16,-16) (16,-16)

Box: 33 x 33

☐ World wraps horizontally

☐ World wraps vertically

View

Patch size 13
measured in pixels

Font size 10
of labels on agents

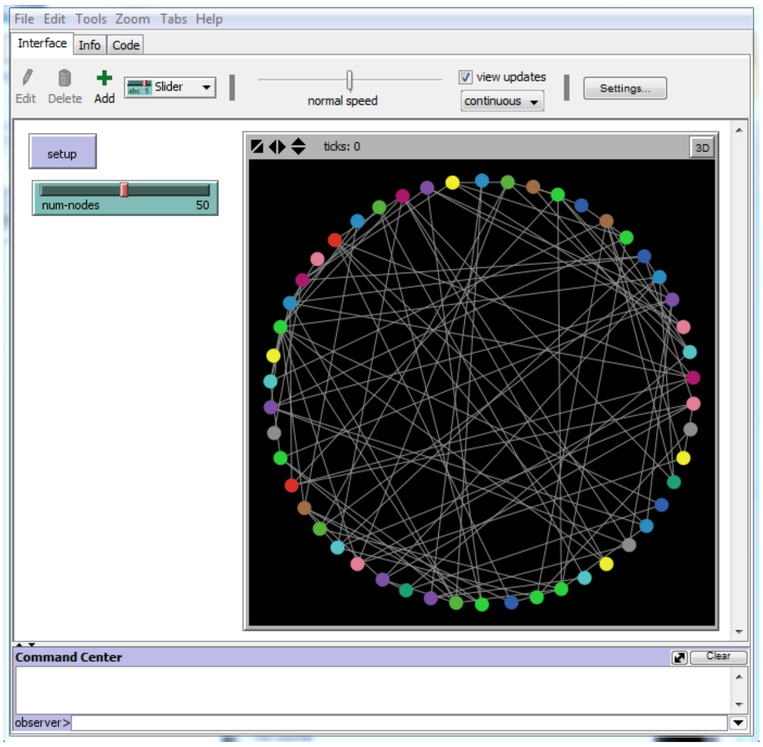
Frame rate 30
Frames per second at normal speed

Tick counter

☑ Show tick counter

Tick counter label ticks

When you do this, the network will look something like this (note that your network will look different because the edges are randomly created):



We now need to add a number of features to make this a real ER model. The current code still allows for pairs of nodes to be connected that are already connected, and the number of edges (currently 100) should not be hard-coded but rather reflect a desired feature like the desired density of the network. Network density is defined as the fraction of the number of edges that are present to the maximum possible number of edges in the network.

First, we will ask nodes to only consider creating an edge with a node that it is not yet connected to, by modifying the edge creation line to this:

```
ask one-of nodes
    [create-edge-with one-of other nodes with [edge-with myself = nobody]]
```

In this code, 'myself' refers to the node that does the asking, i.e. the node we select in the first line of this code.

Second, we will replace the 100 to only create the number of edges needed to create the desired density. In the Code tab replace `repeat 100` with the following:

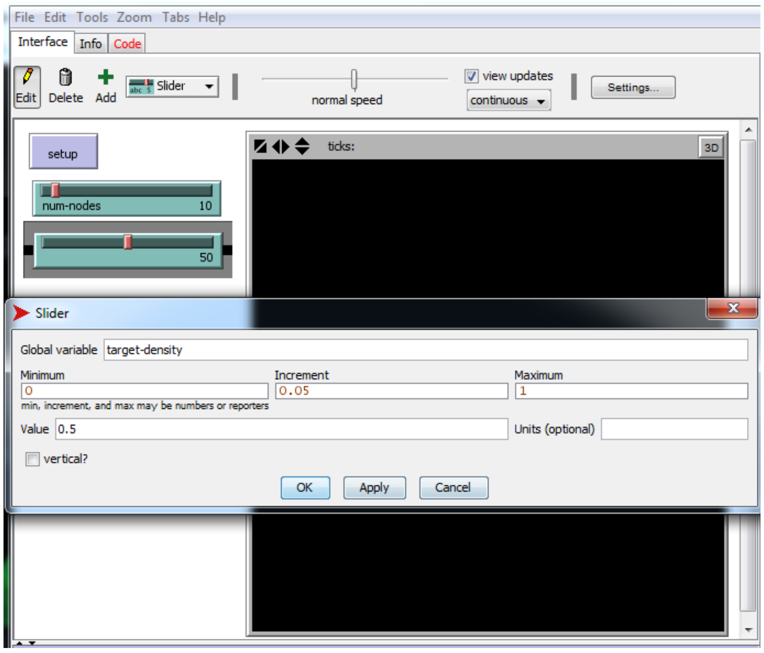
```
repeat (target-density * (((num-nodes * num-nodes) - num-nodes)) / 2)
```

This is a calculation of density in undirected networks. The maximum number of edges in such a network can be calculated as: the maximum number of edges is the number of nodes times the number of nodes, minus the number of nodes (i.e. nodes cannot be connected to themselves), and divided by two (i.e. edges are not directed: in directed networks there are double as many maximum links because they can go in two directions). Since the density measure represents a certain fraction of this maximum number of edges, we multiply it by the variable `target-density`, which represent the density we desire for our network: 0 represents no edges, 0.5 represents half the maximum number of edges being present, 1 represents all the maximum number of edges being present.

When you click the error checker you will notice an error because we have not yet defined the variable `target-density`. We will add this variable in the **Interface tab** as a slider:

- Make sure 'Slider' is selected in the dropdown box.

- Click the 'Add' button.
- Click in the white space below the 'num-nodes' slider. A window will appear.
- Enter `target-density` in the 'Global variable' field. The minimum will be 0, the increment 0.05, the maximum 1 and the Value (default value) 0.5. Click **OK**.



Global variable: target-density
Minimum: 0
Increment: 0.05
Maximum: 1
min, increment, and max may be numbers or reporters
Value: 0.5
Units (optional):
vertical?

The error message will be resolved.

Now when you click the setup button your network will always have only so many edges as required to reach the desired density. For example, if your network has 40 nodes and your desired density is 0.5 then you will create 390 edges:

$$0.5 * \left(\frac{(40 * 40) - 40}{2}\right) = 390$$

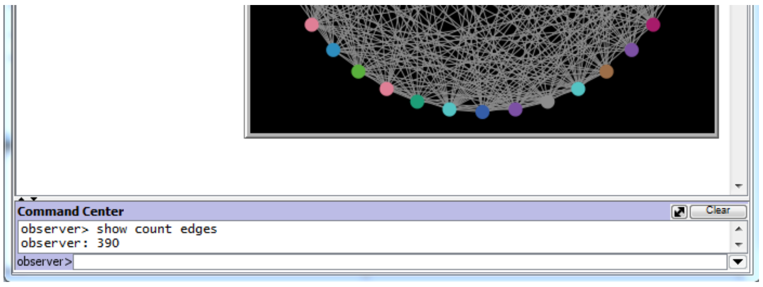
Note: if you get a 'Runtime Error' at this point, click 'Dismiss'. We will resolve this error later.

However, you don't know how many edges there are until you count them! Neither do we know whether the network actually has the correct density until we calculate it. There are a few ways we could do this: using the Command Center or using monitors.

In the Command Center command line at the bottom of the Interface tab write the following:

```
Show count edges
```

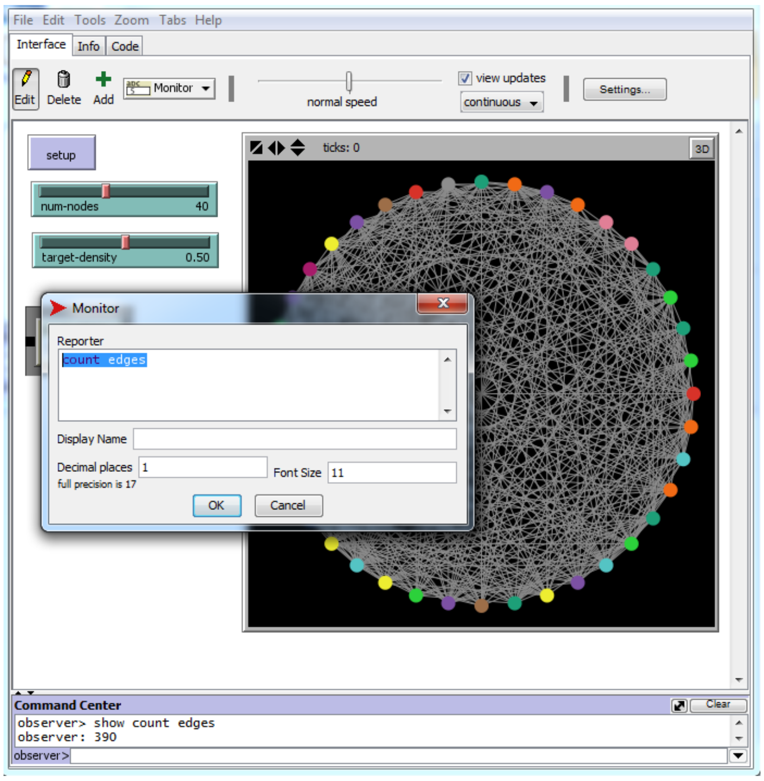
The result will appear in the Command Center window:



```
observer> show count edges
observer: 390
observer>
```

A more convenient way to count the edges is to add a monitor that will show you the number of edges. In the **Interface tab** do the following:

- Select Monitor from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what will be reported.
- Write `count edges` in the 'Reporter' box, set decimal places to 1 (there should be no decimals but it is good to add this for error checking: if you see a decimal number in a reporter window where you don't expect it, you know you did something wrong).



num-nodes 40

target-density 0.50

Reporter: count edges

Display Name:

Decimal places: 1 (full precision is 17)

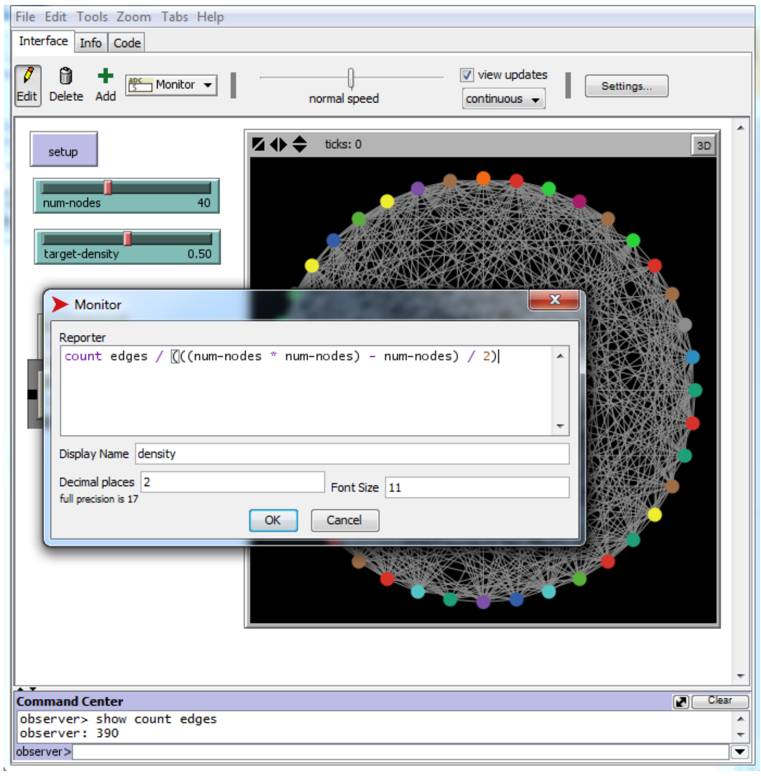
Font Size: 11

```
observer> show count edges
observer: 390
observer>
```

When you click the setup button you will automatically see the number of edges in the monitor box you just created.

Now we will create a monitor for the density. In the **Interface tab** do the following:

- Select Monitor from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what will be reported.
- Write `count edges / (((num-nodes * num-nodes) - num-nodes) / 2)` in the 'Reporter' box, set decimal places to 2, and set the Display Name to 'density' (otherwise the display name of this monitor will be the entire command we entered in the Reporter box). This command calculates the fraction of the number of edges that are present over the maximum number of edges. The number that this monitor gives you should therefore be the same as the setting of the target-density variable. If it is not, then you know you did something wrong so you can search for the problem and fix it.

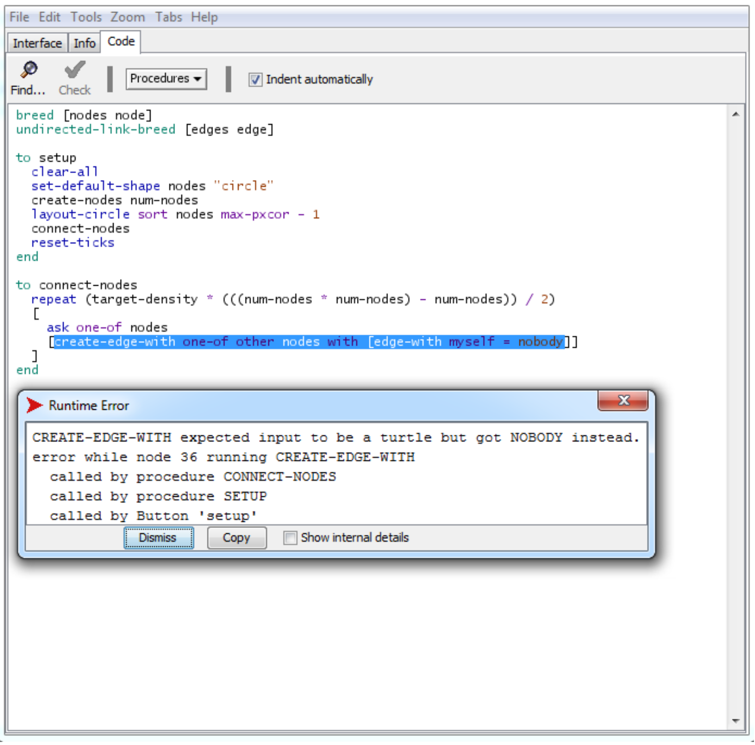


num-nodes 40
target-density 0.50

Reporter
```
count edges / (((num-nodes * num-nodes) - num-nodes) / 2)
```
Display Name density
Decimal places 2
full precision is 17
Font Size 11

```
observer> show count edges
observer: 390
observer>
```

However, there is a problem with this code. When you put the target-density variable high then you will often get a 'Runtime Error'. This Error is very informative and will help us resolve it. It highlights the part of the code where the problem lies and tells us it cannot find another turtle (i.e. node) that fits the criteria:



```
breed [nodes node]
undirected-link-breed [edges edge]

to setup
  clear-all
  set-default-shape nodes "circle"
  create-nodes num-nodes
  layout-circle sort nodes max-pxcor - 1
  connect-nodes
  reset-ticks
end

to connect-nodes
  repeat (target-density * (((num-nodes * num-nodes) - num-nodes)) / 2)
  [
    ask one-of nodes
    [create-edge-with one-of other nodes with [edge-with myself = nobody]]
  ]
end
```

```
CREATE-EDGE-WITH expected input to be a turtle but got NOBODY instead.
error while node 36 running CREATE-EDGE-WITH
  called by procedure CONNECT-NODES
  called by procedure SETUP
  called by Button 'setup'
```

The error tells us that we are asking a node to identify another node it is not yet connected to, when such a node does not exist: i.e. the asking node is already connected to all other nodes.

We resolve this error by specifying that we will only ask nodes that are not yet connected to all other nodes, by adding the following code:

```
ask one-of nodes with [count edge-neighbors < (num-nodes - 1)]
```

The `connect-nodes` procedure will now look like this:

```
to connect-nodes
  repeat (target-density * (((num-nodes * num-nodes) - num-nodes)) / 2)
  [
    ask one-of nodes with [count edge-neighbors < (num-nodes - 1)]
    [create-edge-with one-of other nodes with [edge-with myself = nobody]]
  ]
end
```

Note: in Netlogo we call a pair of nodes that are connected `link-neighbors`, but since we created a breed called `edges` we call can use the command `edge-neighbors`.

Save your model!

10. Probability of edge creation

In order to get an ER model we need to change one final thing: edges need to be created with a certain probability. In our current code a pair of nodes is randomly selected and connected, whereas in an ER model this pair is only connected with a certain probability. Being able to attach probabilities to the creation of edges is very important and useful when you want to express your own hypotheses of network creation. For example, in the friendship network hypothesis mentioned above we stated that the probability of some edges being created is higher than that of others: people with mutual friends will be more likely to become friends themselves.

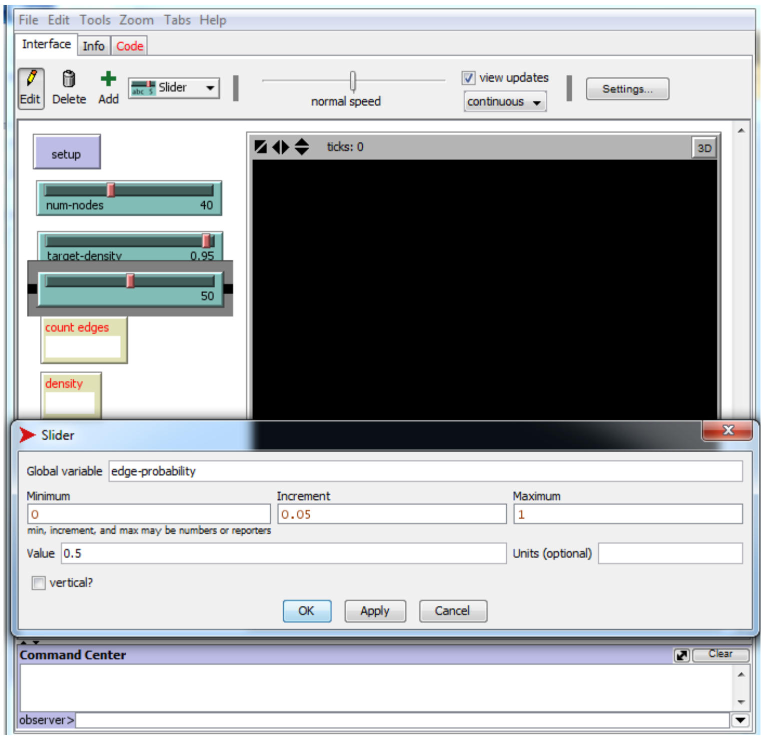
Here we will introduce a simple method of creating edges with a probability variable that you can manipulate to represent your hypothesis of network creation.

First, we modify the `connect-nodes` procedure to only create an edge if a random floating point number between 0 and 1 is smaller than the new variable `edge-probability`:

```
repeat (target-density * (((num-nodes * num-nodes) - num-nodes)) / 2)
[
 ask one-of nodes with [count edge-neighbors < (num-nodes - 1)]
     [if random-float 1 < edge-probability
       [create-edge-with one-of other nodes with [edge-with myself = nobody]]
     ]
 ]
```

We now get an error message because the new variable does not exist yet, so do the following in the **Interface tab**:

- Select Slider from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the slider will do.
- Write edge-probability in the 'Global variable' box, set the minimum to 0, the increment to 0.05, the maximum to 1 and the default value to 0.5. Click **OK**.



Global variable edge-probability
Minimum 0
Increment 0.05
Maximum 1
min, increment, and max may be numbers or reporters
Value 0.5
Units (optional)
vertical?

If this variable is set to 1 then an edge will always be created, if it is set to 0 it will never be created, if it is set to 0.5 it will be created 50% of times.

When you click the setup button now you will notice something is wrong thanks to your monitors: the density in the monitor is not the same as the `target-density` slider and the `count edges` monitor reports a different value every time you press the setup button. This is because this process is repeated a fixed number of times, but not every time an edge is created.

A second change is needed, this time replacing the code repeating the creation of edges. We will implement this by calculating how many edges we need to have at the end of the procedure (stored as the variable `target-edges`), and counting how often an edge is actually created (we let a variable called `counter` start at 0, only when an edge is created do we add one to the counter: `set counter counter + 1`). We only stop the procedure if we have created the desired number of edges (we use the `while [counter < target-edges]` command to repeat the process until the variable `counter` equals the variable `target-edges`).

Modify the connect-nodes procedure to read like this:

```
to connect-nodes
  let target-edges (target-density * (((num-nodes * num-nodes) - num-nodes)) /
  2)
  let counter 0
  while [counter < target-edges]
  [
    ask one-of nodes with [count edge-neighbors < (num-nodes - 1)]
    [if random-float 1 < edge-probability
      [
        create-edge-with one-of other nodes with [edge-with myself = nobody]
        set counter counter + 1
      ]
    ]
  ]
End
```

When you press the setup button you will notice that the monitors report what they should.

Save your model!

11. Force-directed layout

Now that we have a set of nodes and a set of edges, we can place our nodes in a more interesting way than just a circle: we can use layout algorithms to highlight structural features of the network to aid visual exploration of the network. We will do this by using a popular force-directed layout algorithm called the Fruchterman-Reingold layout algorithm. It is sometimes called a spring-embedded layout because the edges act like springs between the nodes that repel each other. More on force-directed layout algorithms can be found here: https://en.wikipedia.org/wiki/Force-directed_graph_drawing

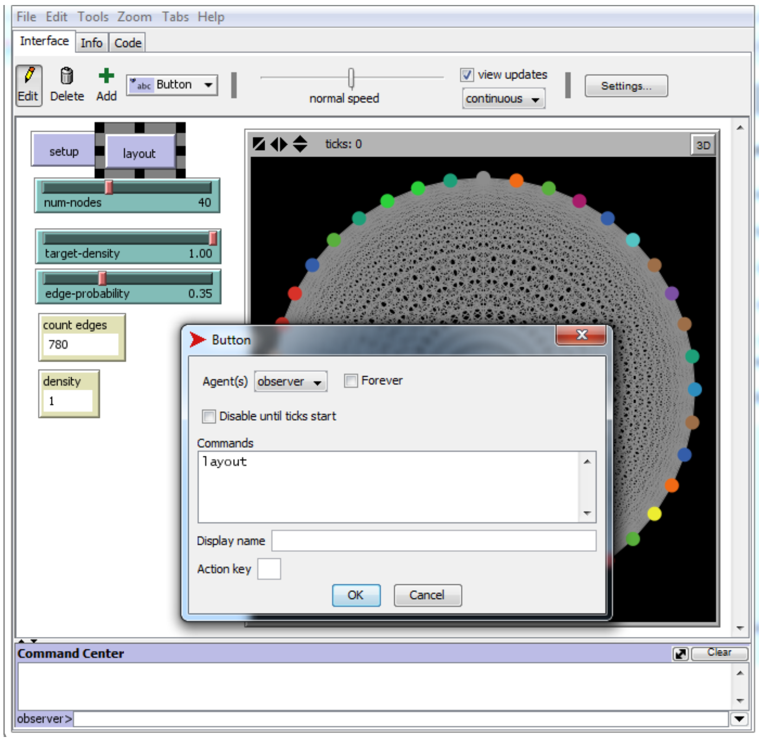
Netlogo makes it incredibly easy to implement this. We will create a new procedure called `layout`, using the Netlogo command `layout-spring`. Add the following code below the `connect-nodes` procedure:

```
to layout
  layout-spring nodes edges 0.2 5 1
end
```

Right-click the `layout-spring` command to access the Netlogo dictionary and find out what this code does. It will perform the layout considering the agent-set `nodes` as the nodes and the link-set `edges` as the edges. The three numbers are variables of the algorithm determining features of the "springs": their resistance, length and repulsion.

In order to use this new `layout` procedure we need to add a button to the **Interface tab**:

- Select Button from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the button will do.
- Write layout in the 'Commands' box. Click **OK**.



num-nodes 40
target-density 1.00
edge-probability 0.35
count edges 780
density 1

Button
Agent(s) observer
☐ Forever
☐ Disable until ticks start
Commands
```
layout
```
Display name
Action key
OK Cancel

Now you can press the layout button to perform the layout.

Note that this algorithm works differently from the circular layout. It does not have one single 'best' outcome, but instead will iteratively change the position of one node after another taking into account the "springs" that connect nodes. The more you click the layout button, the more the nodes will find a suitable position.

This algorithm is most useful for visual exploration with low density networks. Try to set your target-density slider low and perform the layout: the algorithm will reveal interesting structural features that are hidden in the circular layout, such as isolated nodes, nodes with only one connection, and sets of nodes that are all tightly connected with each other.

Randomly created networks are typically very dense and rarely show striking structural features in force-directed layouts. However, this group of layout algorithm is great at revealing structural features that are common in social networks: dense clusters and "bridging" nodes connecting different clusters.

Try changing the values of the three numbers in the `layout-spring` command. You will notice that the effect of the springs works differently. By changing these variables you will be able to tailor your layout algorithm to most appropriately highlight those structural features of your network you want to explore or visualise. Try changing the values in such a way that it provides a more interesting representation of a very dense random network.

Save your model!

12. Network measures

We already introduced three important measures of your network structure: the number of nodes, the number of edges and the network density. Now we will calculate and report two more measures: average degree and average shortest path length.

The **degree** of a node is defined as the number of edges connected to this node. The **average degree** of a network is the sum of the degrees of all nodes in this network divided by the number of nodes.

A **shortest path** between a pair of nodes is the shortest number of edges that need to be crossed in order for the two nodes to be connected. The **average shortest path length** is the average of the shortest paths between all pairs of nodes in the network.

These measures will reveal aspects of the structure of your network and can be linked to features of the network that might interest you in your research: e.g. information is shared much faster to all individuals in a social network if the average shortest path length is low.

We will use a reporter to calculate the average degree in the Code tab and then display it in the Interface tab.

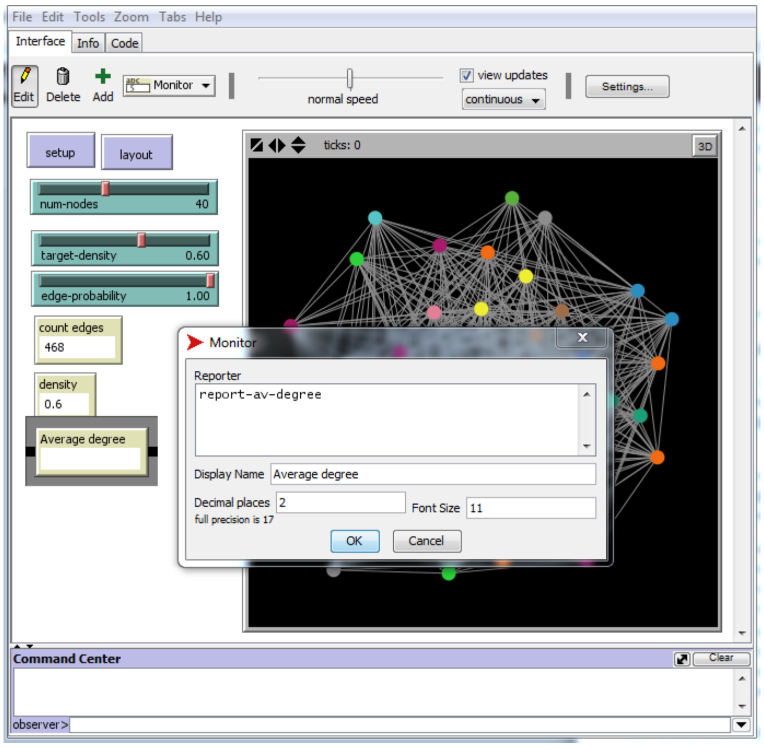
Add the following code to the **Code tab** at the very bottom:

```
to-report report-av-degree
  let av-degree sum([count edge-neighbors] of nodes) / num-nodes
  report av-degree
end
```

In this code we calculate the variable `av-degree` as the `sum` of all numbers of connections (`[count edge-neighbors] of nodes`) of all nodes divided by the number of nodes (`num-nodes`), and we then report the variable `av-degree` so that we can use it in a monitor on the **Interface tab**:

- Select Monitor from the dropdown box.

- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the monitor will do.
- Write `report-av-degree` in the 'Reporter' box, write 'Average Degree' in the 'Display Name' box, include two decimal places. Click **OK**.



Reporter
```
report-av-degree
```
Display Name: Average degree
Decimal places: 2 (full precision is 17)
Font Size: 11

To calculate the average shortest path length we will use a Netlogo extension called 'nw'. In order to use an extension in a Netlogo model we need to add the following at the very top of the code:

```
extensions [ nw ]
```

The 'nw' extension needs to know exactly which sets of agents are the nodes and which are the links. We have created a breed of nodes and a breed of edges and we will work with these. You can specify this by setting the so-called 'context' of the network in the **Code tab**. Add the following to the setup procedure just below the line creating the nodes:

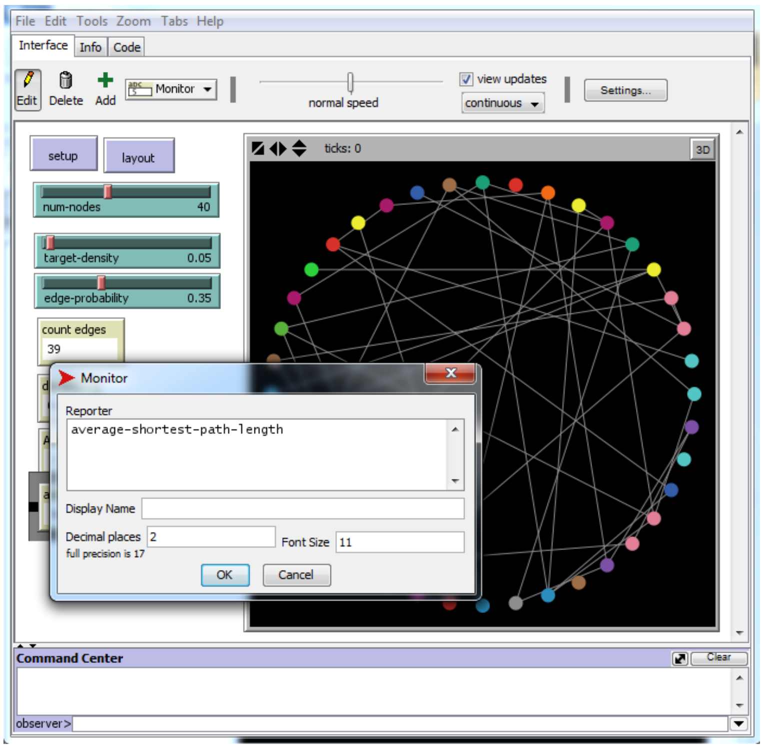
```
nw:set-context nodes edges
```

We will now create a reporter in the **Code tab** that reports a new variable `average-shortest-path-length` by using the command from the 'nw' extension `nw:mean-path-length`:

```
to-report average-shortest-path-length
  report nw:mean-path-length
end
```

We can now add a monitor to the **Interface tab**:

- Select Monitor from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the monitor will do.
- Write `average-shortest-path-length` in the 'Reporter' box, include two decimal places. Click **OK**.



Monitor

Reporter
```
average-shortest-path-length
```
Display Name

Decimal places 2 Font Size 11

full precision is 17

OK Cancel

Now you will be able to measure the average degree and average shortest path length of any network you create in your model.

Note that the average shortest path length monitor will sometimes report 'false'. This happens when your network has isolated nodes, i.e. nodes that are not connected to any other nodes. It will typically occur in random networks with a low density. In such cases, the algorithm cannot calculate the shortest path between all node pairs because some nodes are not connected.

The 'nw' extension was here used in a very simple example to illustrate how extensions can be implemented in a Netlogo model. The 'nw' extension offers a huge range of other techniques that will enable you to perform the most common network science tasks, including some of the things we have implemented manually above. Full documentation for the extension can be found here:

https://github.com/NetLogo/NW-Extension

Advanced task: in networks with isolated nodes you could still calculate the average shortest path length of all nodes that are connected in the largest single connected component, by creating a subset of nodes that only includes the nodes in largest connected component and calculating the average shortest path length for these only. Give it a try! Use the Netlogo dictionary and the 'nw' extension documentation.

Save your model!

13. Trading pots on a network

When performing applied network science research, you are not just interested in creating a certain network structure but rather in understanding what its consequences are within your research context. What is enabled by the relationships? What are the processes taking place on the relationships? The flow of materials or ideas? How does the structure of the network influence this flow? Does it lead to a particular pattern, like distributions of materials or the rapid spread of an idea? These are again questions that you need to ask yourself at the outset and answer in light of your research context.

In this section we will introduce a very simple research context and process: trade and the distribution of pots over a network.

We will assume all our nodes are traders involved in the trade of pottery, and they are able to trade with each other if they are connected in the network. We will give all traders a certain amount of pots and every tick (Netlogo jargon for time step) all pots in the model will be considered for trade. To keep it simple, we will just assume that all traders want to sell the pots they own, want to obtain new items, and that a transaction is successful with a certain probability.

First, we will create a variable for the nodes (the traders) called 'pots' by adding the following to the top of the Code, just below where we define the breeds:

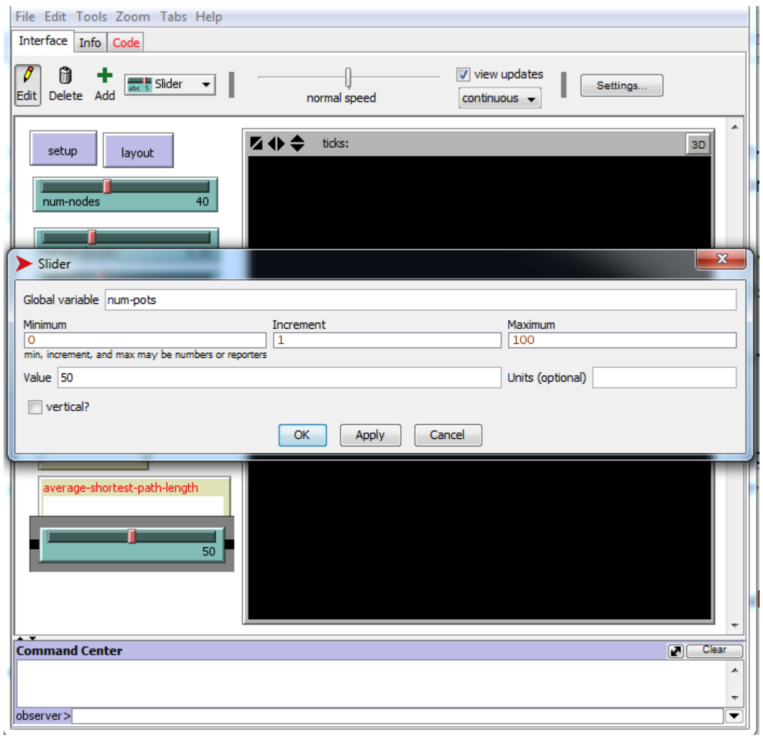
```
        nodes-own [pots]
```

Second, we will give all nodes pots during the setup procedure by adding the following code just after the creation of nodes:

```
        ask nodes [set pots num-pots]
```

Note that we just mentioned a new variable called `num-pots`, so Netlogo will give us an error and we will need to add a slider to control this variable in the **Interface tab**:

- Select Slider from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the slider will do.
- Write num-pots in the 'Global variable' box, set the minimum to 0, the increment to 1, the maximum to 100 and the value to 50. Click **OK**.



num-nodes 40

Global variable num-pots

Minimum 0 | Increment 1 | Maximum 100

min, increment, and max may be numbers or reporters

Value 50 | Units (optional)

☐ vertical?

50

Now when you click the setup button all nodes will be given as many pots as defined by the variable `num-pots`.

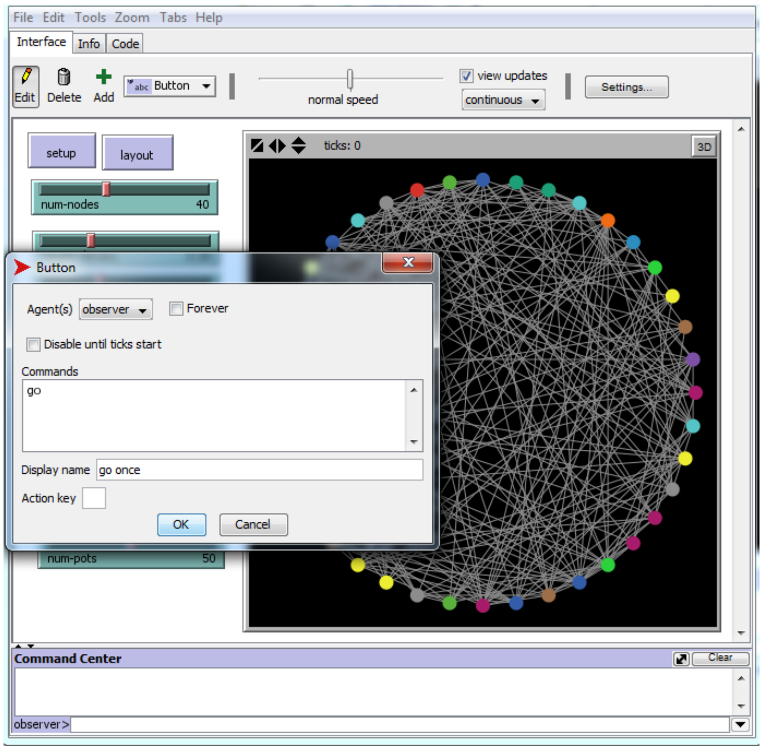
Third, we will create a new procedure that determines what processes will take place when the model runs. So far we have restricted ourselves to determining how the model is set up, and now we will determine what happens after that for every tick (time step).

Add a procedure called `go` by writing at the bottom of your code the following:

```
to go
  tick
end
```

The command `tick` will increase the number of time steps of the model by one every time the `go` procedure happens. To use the `go` procedure we will add a button for it on the **Interface tab**:

- Select Button from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the button will do.
- Write `go` in the 'Commands' box and 'go once' in the 'Display name' box. Click **OK**.



Agent(s) observer
Forever ☐
Disable until ticks start ☐
Commands
```
go
```
Display name go once
Action key

You have just added a button that will run the `go` procedure once every time you click it. If you do so, you will notice that the number of ticks in the Interface window will increase by one each time. You might also want to create a second button that runs the go procedure continuously until you ask it to stop:

- Select Button from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the button will do.
- Write `go` in the 'Commands' box and 'go' in the 'Display name' box. Make sure the 'Forever' box is ticked. Click **OK**.

You have now created a button that will continuously run the go procedure.

Fourth, we will start adding more interesting trade-related commands to the `go` procedure, beginning with a count of the total number of pots during any one tick. In this model this sum will be the same for every tick within a single setup, but we will add this sum in case you decide to modify the model by allowing pots to be added or removed from the model. Write the following in the `go` procedure before `tick`:

```
    let total-pots sum [pots] of nodes
```

Fifth, now that we know how many pots there are in this tick we can repeat trade as many times as there are pots by adding the following code in the go procedure immediately after the sum of all pots:

```
      repeat total-pots
      [
      ]
```

Every command that we now add within these brackets will be repeated as many times as there are pots at the start of that tick.

Sixth, we will randomly select a node which has at least one pot and is connected to at least one other node, i.e. a node that can trade:

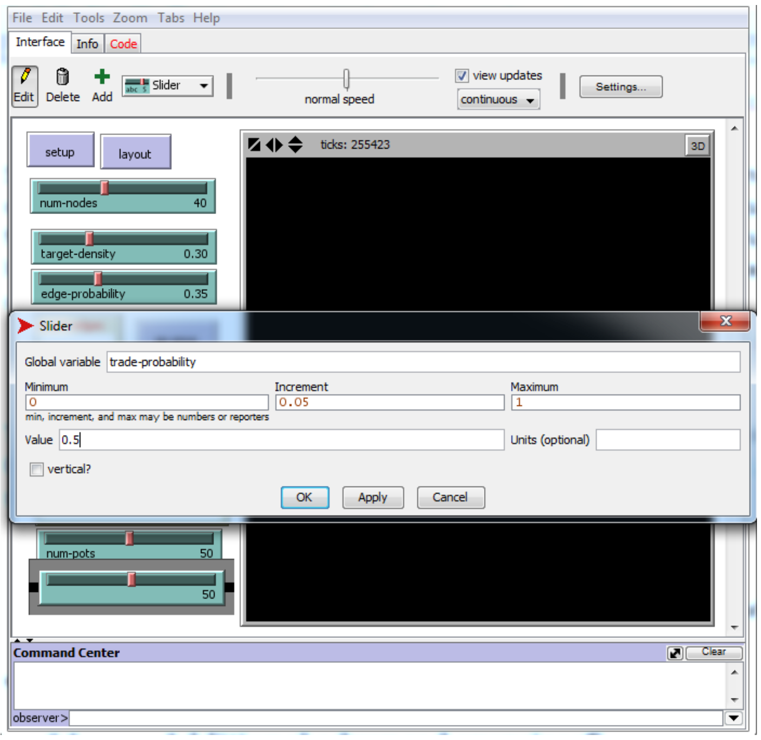
```
      repeat total-pots
      [
        ask one-of nodes with [pots > 0 and count edge-neighbors > 0]
          []
      ]
```

Seventh, we will ask this node to perform a transaction (sell a pot to another node it is connected to) with a certain probability. We will use the same technique as we used earlier for implementing probability: checking whether a randomly selected number is higher or lower than a variable representing the trade probability. Add the following to the `ask` command we just wrote:

```
        ask one-of nodes with [pots > 0 and count edge-neighbors > 0]
          [if random-float 1 < trade-probability
            []
          ]
```

This code will perform an action only if a randomly selected floating point number between 0 and 1 is smaller than the value we set for the variable `trade-probability`. Let's create a slider for this new variable in the **Interface tab**:

- Select Slider from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the slider will do.
- Write `trade-probability` in the 'Global variable' box, set the minimum to 0, the increment to 0.05, the maximum to 1 and the default value to 0.5. Click **OK**.



num-nodes 40
target-density 0.30
edge-probability 0.35
num-pots 50
50

Global variable trade-probability
Minimum 0
Increment 0.05
Maximum 1
min, increment, and max may be numbers or reporters
Value 0.5
Units (optional)
vertical?

Finally, we will add the transaction that will take place if a randomly selected number is lower than the threshold determined by the `trade-probability` variable. Add the following to the `if` command we just wrote:

```
[if random-float 1 < trade-probability
  [set pots pots - 1
    ask one-of edge-neighbors
    [set pots pots + 1]
  ]
]
```

This code will perform a transaction by first reducing the number of pots of the seller (the randomly selected node), then it will randomly select one of the other nodes this node is connected to (this second node becomes the buyer), and finally it will increase the number of pots the buyer owns by 1.

Your complete go procedure should now look like this:

```
to go
  let total-pots sum [pots] of nodes
  repeat total-pots
  [
    ask one-of nodes with [pots > 0 and count edge-neighbors > 0]
    [if random-float 1 < trade-probability
      [set pots pots - 1
        ask one-of edge-neighbors
        [set pots pots + 1]
      ]
    ]
  ]
  tick
end
```

You now have a model where you can create a randomly constructed network with certain features, you can apply a layout algorithm to it, and you can have the nodes trade pots with each other.

Save you model!

14. Plots of pots

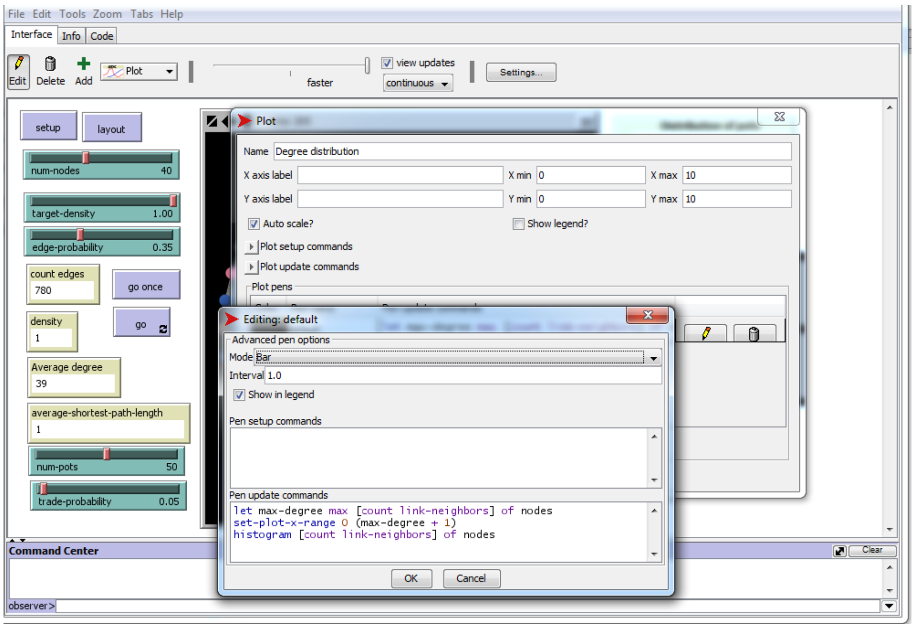
At the moment we cannot explore the trade procedure we just created very easily, because we do not have anything that reports its effects. Let's add two plots that tell us something about the network and the trade processes.

First, let's make a plot showing the frequency distribution of nodes' degrees in the **Interface tab**:

- Select Plot from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the Plot will do.
- Write 'Degree distribution' in the 'name' box.
- Click on the yellow pencil to modify the 'default pen'. A new window will open where you can determine what this pen will plot.
- Set the 'Mode' dropdown box to 'Bar', set the 'Interval' to 1.0, and write the following code in the 'Pen update commands' box:

```
let max-degree max [count edge-neighbors] of nodes
set-plot-x-range 0 (max-degree + 1)
histogram [count edge-neighbors] of nodes
```

- Click **OK** twice.



num-nodes 40
target-density 1.00
edge-probability 0.35
count edges 780
density 1
Average degree 39
average-shortest-path-length 1
num-pots 50
trade-probability 0.05

Name Degree distribution
X axis label | X min 0 | X max 10
Y axis label | Y min 0 | Y max 10
☑ Auto scale? ☐ Show legend?

Mode Bar
Interval 1.0
☑ Show in legend
Pen setup commands
Pen update commands

```
let max-degree max [count link-neighbors] of nodes
set-plot-x-range 0 (max-degree + 1)
histogram [count link-neighbors] of nodes
```

This plot will create a histogram, displaying the frequency distribution of the degree (the number of edges) of nodes. On the X axis we see the degree, and on the Y axis we see the number of nodes that have a certain degree.

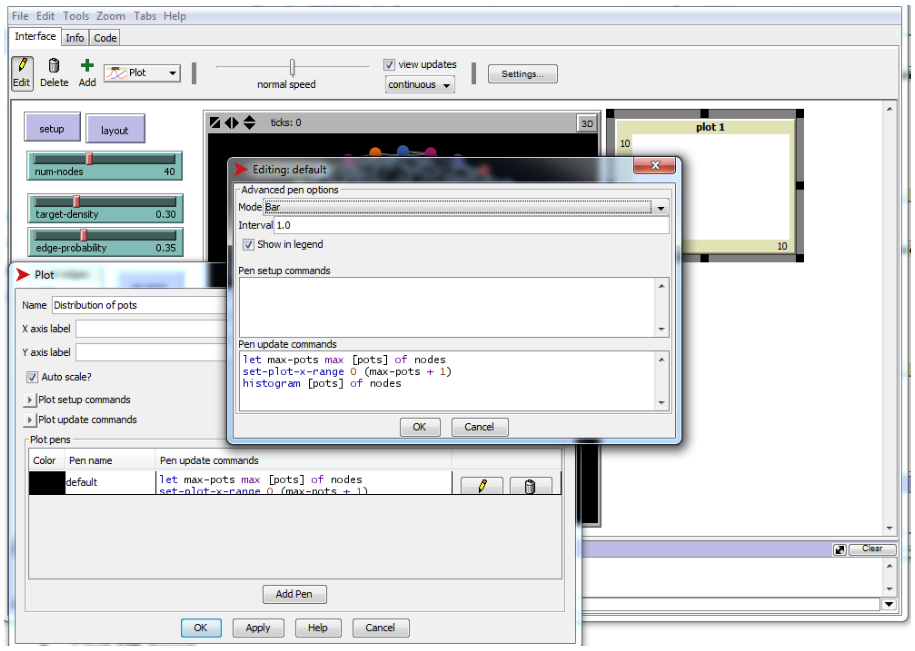
One of the most interesting things to explore would be to see how many pots each trader has: the distribution of pots. We will create a plot that shows us this in the **Interface tab**:

- Select Plot from the dropdown box.
- Click the 'Add' button.
- Click anywhere in the white space. A new window will appear where you can determine what the Plot will do.
- Write 'Distribution of pots' in the 'name' box.

- Click on the yellow pencil to modify the 'default pen'. A new window will open where you can determine what this pen will plot.
- Set the 'Mode' dropdown box to 'Bar', set the 'Interval' to 1.0, and write the following code in the 'Pen update commands' box:

```
let max-pots max [pots] of nodes
set-plot-x-range 0 (max-pots + 1)
histogram [pots] of nodes
```

- Click **OK** twice.


Editing: default
Advanced pen options
Mode Bar
Interval 1.0
☑ Show in legend
Pen setup commands
Pen update commands
```
let max-pots max [pots] of nodes
set-plot-x-range 0 (max-pots + 1)
histogram [pots] of nodes
```

Plot
Name Distribution of pots
X axis label
Y axis label
☑ Auto scale?
Plot setup commands
Plot update commands
Plot pens

Color	Pen name	Pen update commands
■	default	let max-pots max [pots] of nodes set-plot-x-range 0 (max-pots + 1)

This plot will create a histogram, displaying the distribution of pots among nodes. On the X axis we see the number of pots, and on the Y axis we see the number of nodes that own a certain number of pots.

Now you can start exploring the interesting behaviour of this model: the interaction between the network structure and the trade process.

What shape does the distribution of pots have for a dense network?
What does it look like for a sparse network?
Does this change over long time periods?
How does the probability of trade affect this distribution?

If you consider the amount of pots a trader has as their wealth, how would you interpret these different results?
Under what conditions do you see a more equal wealth distribution?
Under what conditions do you see a less equal wealth distribution?
Do these different distributions correlate with certain network structures as defined by the average degree, the degree distribution, and the average shortest path length?
Is the wealth of a node dependent on its position in the network, such as its degree?

Additional advanced exercise: Try changing the way nodes are given pots, by giving some nodes more pots than others, or letting the distribution of pots over nodes follow a normal, uniform or exponential distribution. How does this affect the running of the model? Can you identify a correlation between the number of pots a node starts out with in life and how many pots they manage to keep and obtain?