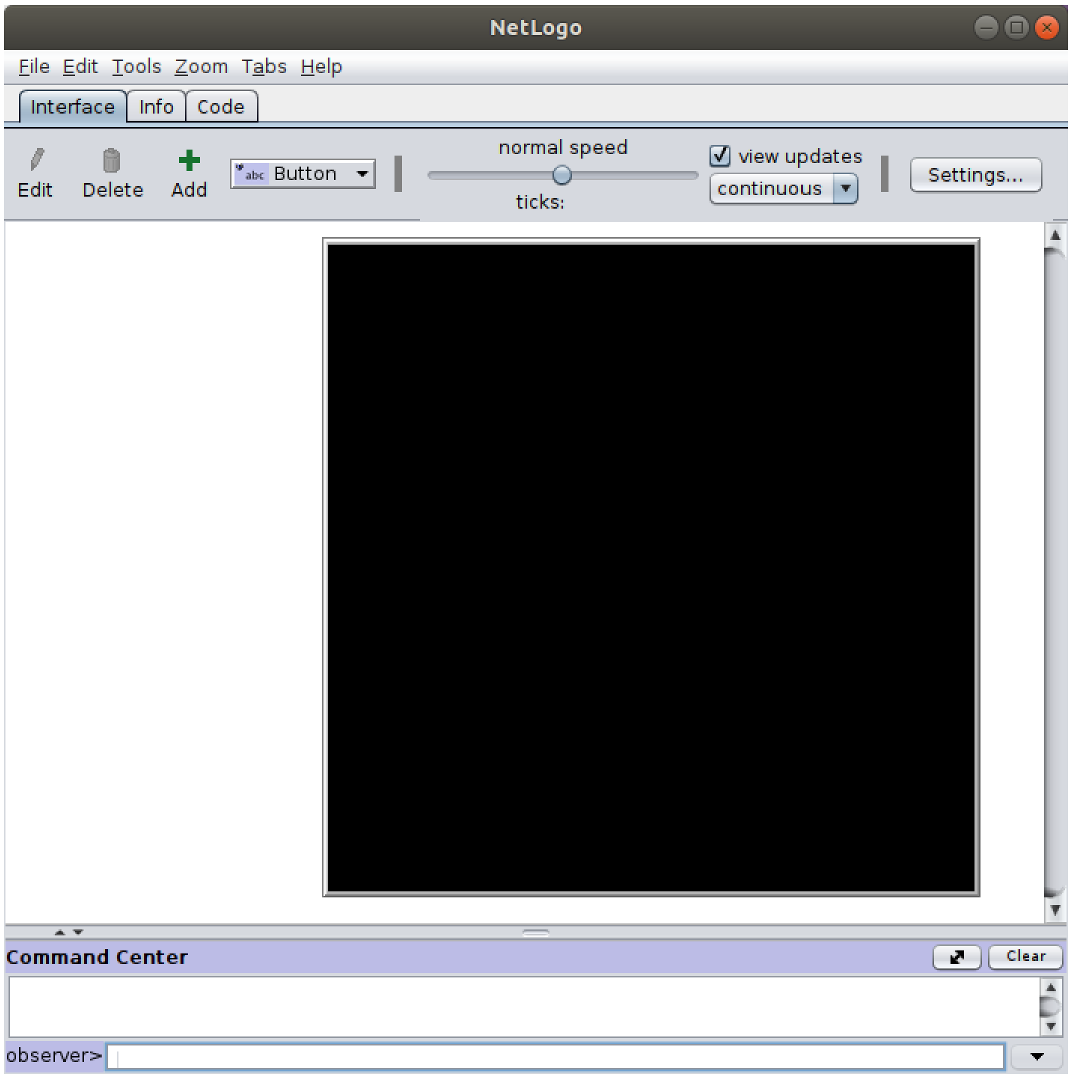
Agent Technology Practical - Lab Assignment 3
Viral network

Report requirements

- Your report must be one .pdf file made with LaTeX. A report template can be found on Nestor.
- Since some of the questions require you to modify a model, you must hand in both a report and your NetLogo code (which should be one .nlogo file). You can upload these using the 'browse my computer' button on this assignment's upload page on Nestor.
- The names of your .pdf and .nlogo files must include the name of this course (which you may abbreviate as 'atp'), your student number(s), the name of the lab assignment, and your lab session group number. (e.g. 'atp_s1234567_s8765432_lab1_group1.pdf')
- The first page in your .pdf file must include the same information again: your student number(s), the name of this course, the name of the lab assignment, and your lab session group number.
- Reports must be in English.
- Clearly label each question you are answering.
- You may work in pairs of two students. If you do, only one of you has to hand in the lab assignment, and clearly indicate who you are working with in your report.
- If a question (such as question 2) requires you to make code or GUI changes to a model, you must describe which changes you made in your report (which lines of code you added, which lines you removed, which lines you modified and how, which GUI elements you added/modified, etc.) Give a short description of why your modification solves the issue. You will not receive points if a modification is *only* in your NetLogo code, and not in your report!

Prerequisites

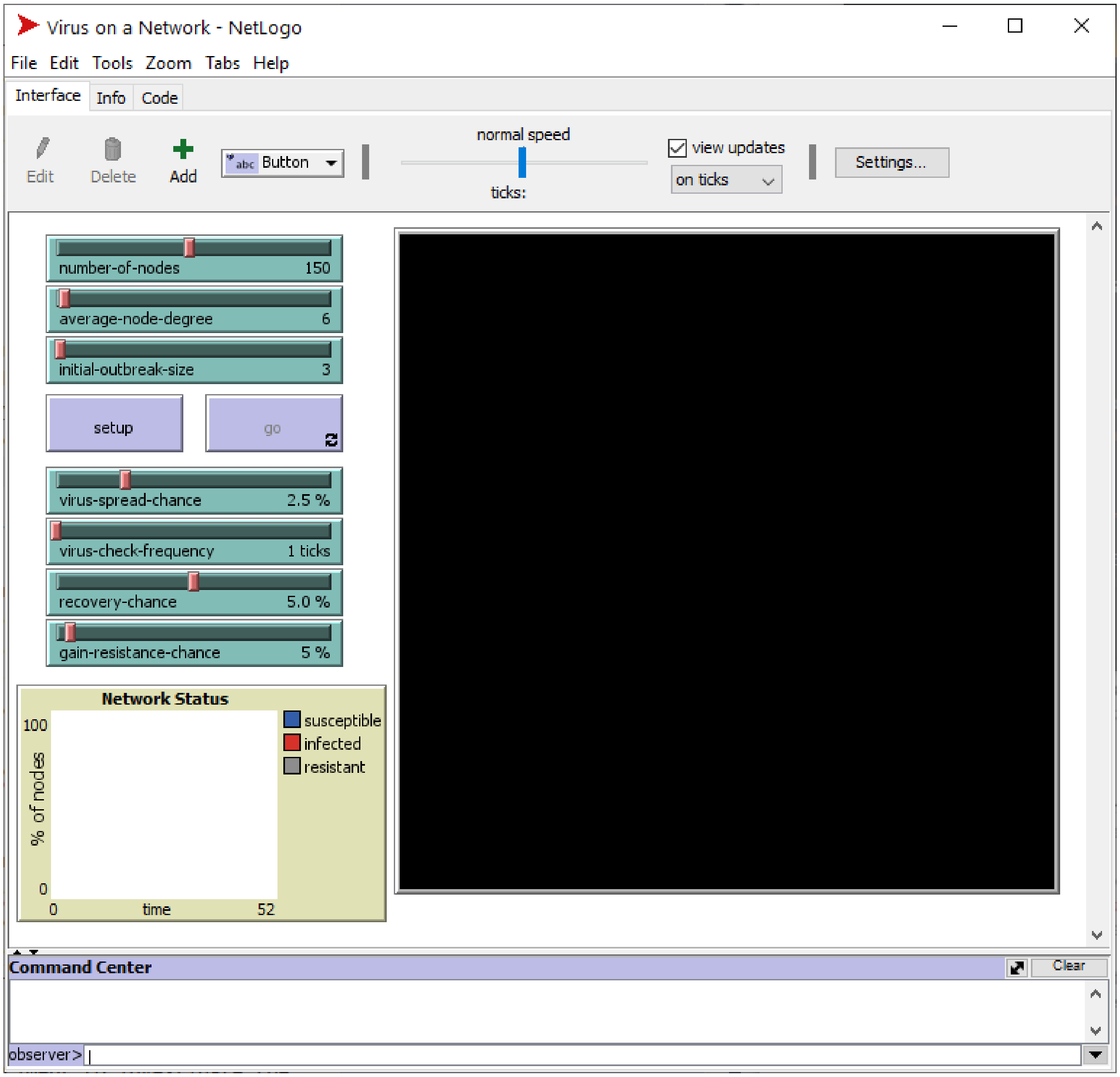
For this lab you will need NetLogo[1]. It can be downloaded at https://ccl.northwestern.edu/netlogo/download.shtml You don't have to fill in the form on the download page - clicking 'download' will work regardless. On Linux, you will have to extract the NetLogo archive and run the NetLogo executable inside. On Windows, you will have to run the installer. Once you have NetLogo launched, it should look something like this:



NetLogo's documentation can be found at http://ccl.northwestern.edu/netlogo/docs/.

For this assignment, we will be using the 'Virus on a Network' model, found in NetLogo's model library at Sample Models - Networks - Virus on a Network. You should open the 'Virus on a Network' model now. You can read more about it in its **Info** tab, which can be found at the top of NetLogo between 'Interface' and 'Code'.

Viral Network



number-of-nodes 150
average-node-degree 6
initial-outbreak-size 3
virus-spread-chance 2.5 %
virus-check-frequency 1 ticks
recovery-chance 5.0 %
gain-resistance-chance 5 %

Figure 1: *The Virus on a Network model, once opened, looks like this. You can also use this image to look up the model's original parameter settings.*

Points possible: 27

Question 1:

Set 'initial-outbreak-size' to 1 and leave the other parameters untouched (to change parameters back to their default value, simply open the model again). Run the model a couple of times and observe what happens. For questions **1a**, **1b**, **1c** and **1d**, you should always use an '`initial-outbreak-size`' of 1, and all other parameters should be set to their defaults.

a) 1 pt. What do you see when running the model?

b) 1 pt. During some model runs, only fewer than five nodes are resistant at the end of the model run. What happens in these cases? Why does it happen?

c) 2 pts. When you look at the plot of the number of susceptible, infected, and resistant nodes, the line for the number of susceptible nodes looks a bit like a mirrored version of the line for the number of infected nodes. Why is this the case? Why is it not an exact mirror image?

d) 2 pt. `random-seed` sets the model's random seed to a specified number. If unspecified, each model run uses a random seed in the range -2147483648 to 2147483647 (inclusive). A model's random seed determines which numbers are generated by the random number generator: running the model with the same random seed produces the exact same results each time. First, click the setup button a couple of times and observe what happens. Then, add the line '`random-seed 6`' between '`to setup`' and '`clear-all`' in the model's code. Now try setting up the model a couple of times, and also try it with different values for `random-seed`. What do you see? What could this functionality be used for?

Note: make sure to remove the line '`random-seed 6`' from the code after completing this exercise.

Question 2:

Currently, the model only reports the current number of susceptible, infected, and resistant nodes. This doesn't tell us how many agents have encountered the infection throughout the entirety of the model run, which may be a more important metric when investigating how to combat real (computer) viruses. In a real-life situation, we would like to prevent nodes (be they computers or people) from being infected in the first place.

For question **2** and its subquestions, you must describe which code and GUI changes you made in your report, and you must give a short description of why your modification solves the issue. Don't forget to mention any inputs and commands you use in GUI elements.

a) 2 pt. First, give nodes (turtles) a new variable: whether they have been infected or not. Note that this is different from whether a node is *currently* infected. In NetLogo, boolean variables usually have a question mark at the end, e.g. 'finished?'. For each node, this new variable should be set to **false** on set-up. When a node becomes infected, this new variable should be set to **true**, and remain true for the rest of the model run, even after the node is no longer infected.

b) 3 pt. Secondly, add a global variable that counts the number of unique nodes that have been infected, past or present. You will need `globals` for this, which can be found in the NetLogo Dictionary at http://ccl.northwestern.edu/netlogo/docs/. The dictionary can be found in the column on the left, under 'Reference'. Once you're in the NetLogo Dictionary, you can use ctrl + f to search for NetLogo functions and variables.

Now you need to make sure that this global variable is regularly updated. One way to do this would be to add a new function which sets this global variable's value to the current number of turtles that have been infected during the entire model run (see your answer to **2a**), and making sure this function is called in the setup and go commands. You can use `set`, `count`, and `with` for this.

Another way to do this would be to increment this global variable by 1 whenever a node is infected for the first time.

c) 2 pts. Now, add a monitor to the GUI which displays the number of unique nodes that have been infected (past or present) throughout the model run. Again, this number should be equal to the total number of unique nodes that have been infected at least once. If done correctly, you should see the following: after setup, but before you run the model, this number should be equal to the number of initially infected nodes. During the model run, this number should always increase, and never decrease. At the end of the model run, this number should never be higher than the total number of nodes in the network.

It may also be interesting to see how the amount of nodes that have encountered the virus increases over time. To do this, add a line to the model's plot which shows the percentage of nodes that have encountered the virus during the model's run. Make sure this is a percentage, and not a total amount or a fraction.

If you're not sure how to do these things, **Tutorial #3: Procedures** (found in the NetLogo manual, in the leftmost column, under 'Learning NetLogo') has some good examples. Again, the NetLogo manual can be found at http://ccl.northwestern.edu/netlogo/docs/. You can also look at how the existing plot lines work.

Question 3:

Now that we have a measure of how many nodes have encountered the virus during a model run, we can investigate how to decrease this number.

a) 2 pts. Suppose this model is a model of a biological virus outbreak, and the nodes are people. Which of the model's parameters could, and which of the parameters could we ***not***, realistically influence, in real life, when implementing measures during an outbreak? Why?

b) 2 pts. Pick one of the parameters we can influence in real life (see your answer to question **3a**), what is its influence on the final number of people who have been infected (see your answer to question **2**) at the end of each model run? Support your claim with simulation results (run the model several times by hand and include a table or graph).

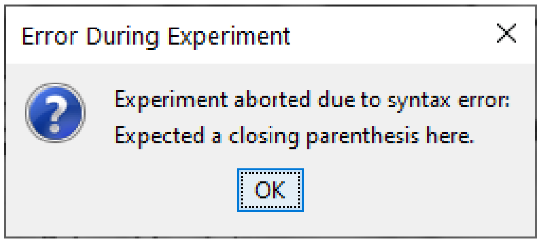
c) 3 pts. It is much easier to use a batch run than to manipulate a model by hand if you want to do an experiment. For this exercise, pick one of the parameters you wish to experiment with (you may use the same parameter you used in **3b**, but you may also use a different one). Next, we are going to use BehaviorSpace to run an experiment to investigate this parameter's effect on the model. In NetLogo, you can find BehaviorSpace in the Tools dropdown list.

Before you continue, we recommend reading the BehaviorSpace guide of the NetLogo manual. It can be found at https://ccl.northwestern.edu/netlogo/docs/, under 'features' in the column on the left. You don't need to read 'advanced usage'.

Create a BehaviourSpace experiment to investigate the effect of your parameter of choice on the duration of the outbreak, as well as its effect on the total number of nodes that have encountered the virus (see question **2b**). Make sure you

- Use a descriptive range of parameter values to test (e.g. not just 0 and 100).
- Select a number of repititions that ensures your results are robust to differences between model runs.
- Make sure to use, as model output, the total duration of the model run, and the number of nodes that have encountered the virus (see question **2b**).
- Do not measure model runs at every step.
- You do not have to make any changes to the setup commands, go commands, final commands, and stop condition, and you do not have to include a time limit.
- When you run the model, make sure to use a spreadsheet output.

Describe the changes you make in BehaviorSpace in your report.


Error During Experiment

Experiment aborted due to syntax error:
Expected a closing parenthesis here.

Figure 2: *You may encounter this error, which is caused by multiple reporters/output variables being on the same line. Simply inserting line breaks between each one of them solves the issue.*

d) 1 pt. Run your BehaviorSpace experiment, and look at the output (which should be a .csv file) using spreadsheet software such as LibreOffice Calc or Excel. Which information do the first six rows contain? Which information can be found in the remaining rows?

e) 4 pts. For this exercise you will need R. RStudio is also highly recommended. You should have encountered R during the course 'Statistics'. You can download R here: https://cran.rstudio.com/, and you can get RStudio here: https://rstudio.com/products/rstudio/download/#download.

For this exercise, you must import the data you generated in questions **3c** and **3d** into R. An example and explanation of how to do this can be found in the Appendix of this document, and an R script example can be found on Nestor.

Include the code you use to import your data into R in your report, but don't upload your R script separately. You don't have to include the path you used in the `setwd` function, as that is private information.

Next, think of a good way to summarize your data in R (think of plots, mean, SD, tables, etc...), and include your 'data summary' (as an image or table) in your report, as well as the R code needed to make it. What kind of relation/phenomenon/information about the model can be seen in this summary, if any?

f) 2 pts. Looking at your data summary, which property or relation could be tested using a statistical test? Which test would be needed? If not, why not?

You don't actually have to do this test in R. Just describe your reasoning in your report.

Appendix

Code example and explanation for question 3e

The code you need to load your data into R and convert it into a usable format will look something like this. However, this exact code will not work for your dataset, and you will need to modify it. How to do this, and how it works, can be found on the next page. This R script can also be found on the assignment's Nestor page.

```
setwd("path\\to\\data")
dataset <- read.csv("filename.csv", skip=6, row.names=1, header=TRUE)
dataset <- dataset[-c(1,2),] #Make sure to remove the correct rows

dataset[1,] <- as.numeric(dataset[1,])
dataset[2,] <- as.numeric(dataset[2,])

dataset1 <- dataset[,seq(1,ncol(dataset),by=2)]
dataset2 <- dataset[,seq(2,ncol(dataset),by=2)]
dataset2[1,] <- dataset1[1,]

row.names(dataset1)[1] <- "name1"
row.names(dataset2)[1] <- "name2"

row.names(dataset1)[2] <- "name3"
row.names(dataset2)[2] <- "name4"

#dataset1 <- t(dataset1)
#dataset2 <- t(dataset2)
```

Code explanation

```
setwd("path\\to\\data")
dataset <- read.csv("filename.csv", skip=6, row.names=1, header=TRUE)
```

First, load your data file, the one you created in question **3b**, into R. You can do this using `setwd` and `read.csv`. Recall that assigning values to variables is done with <-. For `read.csv`, you can use the argument `skip` to remove the first six rows of the data. If you don't, you'll have to remove these lines after reading in the data file. You will also want to use `row.names = 1` to use the leftmost column as the row titles. If `header` is set to `TRUE`, the values in the first row will be used as column names. You can use the name of your dataset as a command to view the dataset. You can view a specific row in the dataset using `dataset[1,]` for row 1, `dataset[2,]` for row 2, et cetera.

```
dataset <- dataset[-c(1,2),]
```

The only rows you are interested in are the rows containing the parameter you are investigating, and the row containing the output variables. You may want to remove every other row. As an example, if you want to remove the first and second rows from a data frame in R, you can use `dataset <- dataset[-c(1,2),]`. Here, '`dataset`' is a variable name, not an R function. Make sure your data only has two rows! You can check this using `nrow`.

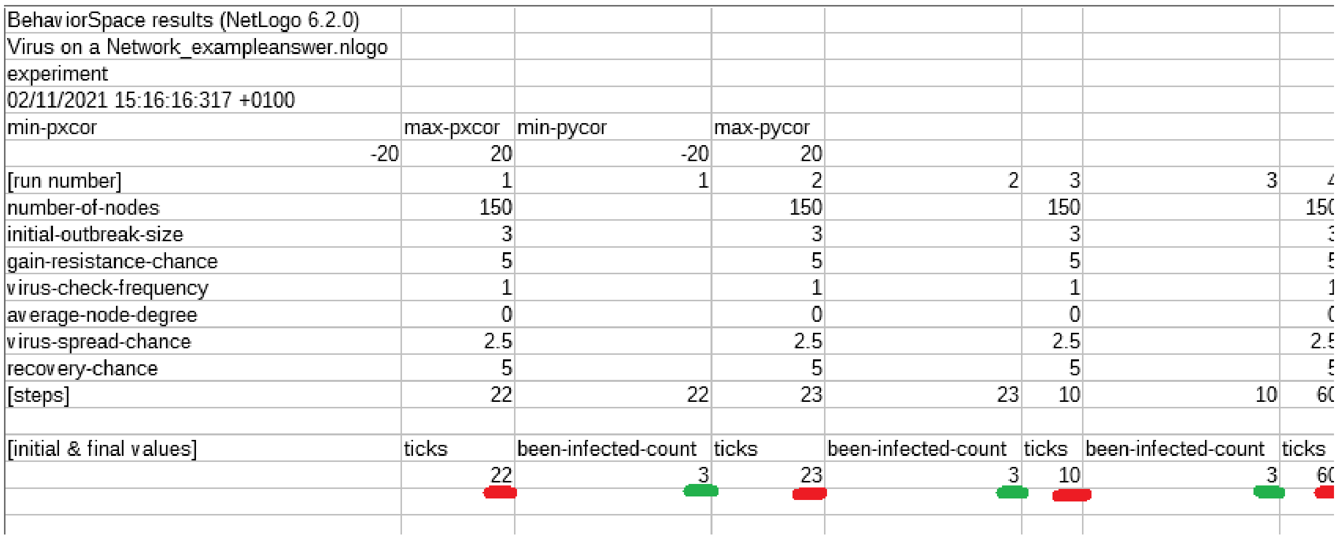
```
dataset[1,] <- as.numeric(dataset[1,])
dataset[2,] <- as.numeric(dataset[2,])
```

Currently, the values in your datasets may be strings, and not numbers. You should convert your data from string to numeric. `dataset[1,] <- as.numeric(dataset[1,])` is an example of how to convert a row.

```
dataset1 <- dataset[,seq(1,ncol(dataset),by=2)]
dataset2 <- dataset[,seq(2,ncol(dataset),by=2)]
```

Note the odd columns in your data contain one of the output variables, while the even columns contain the other output variable.

Here is an example, where we use '`ticks`' and '`been-infected-count`' as output variables. Green indicates the values for '`been-infected-count`', and red indicates the values for '`ticks`'.



BehaviorSpace results (NetLogo 6.2.0)							
Virus on a Network_exampleanswer.nlogo							
experiment							
02/11/2021 15:16:16:317 +0100							
min-pxcor	max-pxcor	min-pycor	max-pycor				
-20	20	-20	20				
[run number]	1	1	2	2	3	3	4
number-of-nodes	150		150		150		15
initial-outbreak-size	3		3		3		3
gain-resistance-chance	5		5		5		5
virus-check-frequency	1		1		1		1
average-node-degree	0		0		0		0
virus-spread-chance	2.5		2.5		2.5		2.
recovery-chance	5		5		5		5
[steps]	22	22	23	23	10	10	6
[initial & final values]	ticks	been-infected-count	ticks	been-infected-count	ticks	been-infected-count	ticks
	22	3	23	3	10	3	6
	🟥	🟩	🟥	🟩	🟥	🟩	🟥

It may be useful to separate your data in two: the data for the duration of each model run, and the data for how many nodes encountered the infection. You can do this using a command such as `dataset[,seq(1,ncol(dataset),by=2)]`. Change the '1' to '2' for the other half of the data. The function `seq` creates a sequence of numbers.

```
dataset2[1,] <- dataset1[1,]
```

Now you should have two datasets, one for each output variable. One of these datasets will have an empty row where the parameters should be (marked with `<NA>`). To solve this, you can simply copy the row containing the parameter values from one dataset into the other. You can use code such as `dataset1[x,] <- dataset2[x,]`, where ‘x’ is the row number for your parameter values. Make sure you’re copying *from* the data *with* the parameter values, *to* the data *without* the parameter values.

```
row.names(dataset1)[1] <- "name1"
row.names(dataset2)[1] <- "name2"

row.names(dataset1)[2] <- "name3"
row.names(dataset2)[2] <- "name4"
```

It may be useful to change the datasets’ row names to something descriptive. Some of your rows may not even have names yet. As an example, you can use `row.names(dataset1)[1] <- "name"` to set the name of the first row to ‘name’. Make sure your row names do not use any special characters, or you may get in trouble later.

```
#dataset1 <- t(dataset1)
#dataset2 <- t(dataset2)
```

Some R functions may require your data to be transposed, which can be done with `t()`. Note that `#` is used for commenting out code in R.

References

[1] U. Wilensky. *NetLogo*. Northwestern University, Center for Connected Learning and Computer-Based Modeling, Evanston, IL., 1999. `http://ccl.northwestern.edu/netlogo/`.