# Supplemental Text 3: ABM for Outreach Tutorial

Imagine that you want to communicate the results of the models built in Romanowska et al. (2019) and Davies et al. (2019) to the public; maybe your colleagues want to learn about this ‘agent-based modeling’ thing you are doing, maybe you are using these models to teach students computational archaeology, or maybe you are using agent-based models you’ve built to further explore hypotheses of landscape use. How would you go about communicating your model accurately and well? Here we bring users through a step-by-step process of taking these models, exporting the data, and packaging the model for public consumption.

It is critical to state that one run of a simulation only shows one possible outcome, and that to truly understand model dynamics a systematic ‘sweep’ needs to be performed. Here, the term sweep refers to running a simulation dozens, hundreds, or even thousands of times to find the central tendencies of a model. A single plot from one run is insufficient for directly bolstering narratives or directly challenging false interpretations of the past. Thus, this tutorial demonstrates how to export data from a simulation to enable the examination of central tendencies of multiple runs.

To do this we return to the model developed in Romanowska et al. (2019) and in Davies et al. (2019). To finish this three-part tutorial we are going to show you how to code in a stop procedure, how to export images and data from the simulation, and how to use the exported data to create figures and analyze the data. If you followed the previous tutorials open your model and make a copy. If you intend to only complete this tutorial: download the NetLogo model and open it; the program NetLogo is open source and should be downloaded at https://ccl.northwestern.edu/netlogo/. Please note that the version of the simulation should be 6.0.3 or greater to avoid errors from using earlier versions of the software. All code is written below in the font Courier New.

After downloading and installing NetLogo, open the program. The initial screen has three tabs, the *Interface*, *Info*, and *Code* tabs. Imagining a NetLogo model to be like a computer, the Interface tab is akin to the screen and keyboard, while the Code tab is similar to the internal components of the machine. The Info tab can be used for model description and documentation.

**3.1 Exporting simulation data**

At the end of building your model, and before you communicate your model to the public, you will want to run your model multiple times to examine the model outcome and to understand how different scenarios (modeled using different parameter values) affect this outcome. This process is called ‘experiment design’ (Romanowska 2015). In this tutorial we show you how to export data, via three commands: export-view, export-plots, and export-world.

***3.1.1 Export-view***

This command can be used to create graphics and to create movies that you can deliver with the report. If you wish to export the view panel (the screen showing the simulation), the export-view primitive will achieve that. The command:

export-view “name-of-the-file.png”

will create a .png file of the screen cast. This should be placed either at the end of the go procedure, or within the stopping procedure. If it is placed at the end of the go procedure, the image will be export at the end of every tick; if you run your model for many ticks that will be many images. Since we use a static filename (‘*name-of-the-file*.png’) the image will get repeatedly overwritten. To avoid this, use a dynamically assigned name, for example:

export-view (word ticks “.png”)

or

export-view (word “name-of-the-file” ticks “.png”)

Ticks is a reporter within NetLogo that returns the number of the current step in the model run, thus adding a unique number to every image. The files will be saved in the same folder where the NetLogo code is saved. If you prefer to save them elsewhere, specify the path to the file, e.g.:

export-view (word “C:/My documents/” ticks “.png”)

These files can be then used to create a video or a gif of the forager moving around the landscape. Gimp is one open source platform that allows for creating gifs with multiple png files. There are also numerous free online tools, such as makeagif.com that will turn your images into gifs in a matter of seconds. You can also use R to automate making images into a video with the package “animation” (Yihui Xie 2013). Note, however, that the exporting of an image is fairly computationally heavy, so running this every tick will slow down your simulation. If you only care about the final image, you can move this bit of code to the stop command, so it only performs once.

***3.1.2 Export-plots***

This command does not export the physical view of the plots, but exports the data recorded in the plots to be used by future statistical software for creating graphs. You can either export individual plots, or if you have multiples, you can export the data from them all with one command. To export one plot write:

export-plot "Toolkit Richness" "TRichnessplot.csv"

Where export-plot is the command, “Toolkit Richness” is the name you gave your plot in the interface, and “TRichnessplot.csv” tells NetLogo to save the data from the Toolkit Richness plot in your active folder as a file called TRichnessplot.csv

As above, the files will be saved in the same folder as the NetLogo code is saved. If you prefer to save them elsewhere, specify the path to the file, e.g.:

export-plot (word “C:/My documents/” "Toolkit Richness" "TRichnessplot.csv")

If you have multiple plots and you want to save more than one, you simply change the command to:

export-all-plots “Trichnessplots.csv”

This will then create a .csv file with data from every plot held within.

Your code in the go procedure should now look like:

; GO PROCEDURE

to go

 ; move agents based on elevation

 ask turtles [

 if random 9 > 0 [

 move-to one-of neighbors with [ patch-elevation > 0 ]

 if [ patch-elevation ] of patch-here > elevation-threshold [

 repeat (elevation-multiplier - 1) [

 move-to one-of neighbors with [ patch-elevation > 0 ]

 ]

 ]

 ]

 ]

 ; if any quarries nearby, reprovision

 ask turtles [

 if gis:intersects? quarries patch-here [

 reprovision-toolkit

 ]

 ]

 ; if any tools in toolkit, discard

 ask turtles [

 if length toolkit > 0 [

 discard-tools

 ]

 ]

 ; if simulation reaches 100,000 time steps, recolor patches

 ; based on artifact density and stop simulation

 if ticks = time-limit [

 ask patches with [ length assemblage > 0 ] [

 set pcolor scale-color red length assemblage 0 30

 ]

export-view (word "Tutorial\_tick\_" ticks ".png")

 export-plot "Toolkit Richness" "plot.csv"

 stop

 ]

 ; advance time

 tick

end

***3.1.3 Export-world***

This command will write values of all variables defined within your simulation. If you are performing a sweep of multiple parameter values, this can allow you to easily access which run of the simulation contained which value. This file will also contain the random-number seed used at the beginning of the simulation, which can be useful for understanding how the random order of which agent does what impacts simulation output. However, as this is usually a big file (it will have data for every pixel where there is a raw material source) you probably only want to have this output once. We recommend putting this in your setup procedure so you have information on your initial conditions. Type in:

export-world "results.csv"

And place it right before you reset the ticks, so the end of your setup procedure will look like this:

 ask one-of patches with [ patch-elevation > 0 ] [

 sprout 1 [

 set size 8

 set shape "person"

 set color yellow

 set toolkit []

 ]

 ]

 ;give patches empty assemblage lists

 ask patches [

 set assemblage []

 ]

 export-world "results.csv"

 ;reset time

 reset-ticks

end

If you want this data to be exported more frequently, move export-world "results.csv" into the go procedure; note, however, this will be time consuming, and likely not necessary if the underlying patch data is static.

**3.2 Behavior Space**

When you build a model you usually want (and need!) to examine how changing one parameter will affect the results and other parameters in the model. This is called a sensitivity analysis and it is necessary in order to understand the chain of causality that produces the results we are reporting. You also want to run any stochastic (containing random elements) simulation multiple times to make sure that you are seeing *trends* and not *random instances* since any one run will be different from another. Behavior Space is a tool provided by NetLogo to run your model multiple times without the need to set it up manually every time.

For example, in our model if we increased the number of raw material sources, that will change the maximum number of types of raw material the agent can carry. However, this is also affected by the quantity of rocks an agent can hold at any one time (in our simulation max\_carry). If the max\_carry parameter was set at 2, yet the number of raw material sources was set at 100, the maximal number of raw material types an agent could carry would still be two (because the agent cannot carry any more). Alternatively, if you set the max\_carry parameter at 100, but set the number of raw material sources at 2, the maximum number of raw material types that can be carried would equally be two (because the agent has no chance of encountering more types). You can see how the different final toolbox richness will depend on the combination of the two parameter values.

We also may want to test hypotheses related to how parameters interact. For example, we may expect in the above example that the proportions of raw materials would move toward disequilibrium (away from 50/50) the farther from one source an agent gets. Therefore we might want to run the model multiple times with these parameter values, as well as recording where on the landscape the agent is located. This could be important for looking at likely proportions of raw materials that we would find in real archaeological sites in the region. Thus we can run the simulation multiple times to generate quantitative output that we can analyze with statistical packages, like Stata, R, Python or even Excel.

To start BehaviorSpace go to Tools→ BehaviorSpace



Figure 3.0.1. Where to find BehaviorSpace window.

This will open a new window that will say “**Experiments**:”. Unless you have used BehaviorSpace before, the field will be blank. Click “**New**” to start a new experiment.

Give your experiment a name in the first line. Use a unique name that is easy to remember. I am calling this “Tutorial\_Sweep” and then adding the date “May\_2018”. Usually we will run multiple sweeps, so it’s good to have a date stamp.

The next line says “**Vary variables as follows**.” You will note that only those variables that have sliders or buttons are included in this list. For now that includes “max\_carry” which is the maximum amount of raw material an agent can carry. We will sweep across multiple values of the “max\_carry” parameter to see how the carrying capacity affects the richness of the agent’s toolkit. We can write this two ways:

[“max\_carry” 10 20 30 40 50 60 70 80 90 100]

In this version we specify the exact numbers we want to examine. This is useful syntax to know. However, a simpler (and more elegant) way to write this would be:

[“max\_carry” [10 10 100]]

This way of writing says that we are starting at 10, ending at 100, and iterating by 10 each step. It saves space, and does the same thing as the above. We will leave the other two variables (time-limit and elevation-threshold) set to their defaults.

Another variable you will want to use is the random number seed. In agent-based models a random number seed is used to initiate running model components. For example, a random number generator will ensure that agents are randomly chosen to execute functions at different times so that the same agent doesn’t always go first in a task (e.g., foraging) enabling a bit of stochasticity in the model. Since we want true reproducibility we want to let people know which random numbers we used so they can recreate our runs. Usually I will choose a few numbers to just place in here. It doesn’t really matter what numbers you choose--it just matters that you do it and that you report them.

[“seed” 14 34 87 62 43]

Following this is the bar for “**Repetitions**.” This controls how many times the simulation will run with each of the above parameter values. Since this is a stochastic simulation (the agent chooses where to move next randomly) it is necessary to run the experiment multiple times and then average across the values (also known as finding **central tendencies**). We will set the number of repetitions at 10 but in normal circumstances it should depend on the amount of variance in your model. Put simply, the higher the variance, the more runs you should perform to capture the trends, as well as expose unexpected yet interesting results.

Next, “**Measure runs using these reporters**” refers to the data that you are going to record at every timestep. You see that the default is count turtles, which is useful for simulations where you are examining population growth and decline. In this simulation, however, since we are keeping the population steady (at one agent) we want to count something else. This is where we ask to export toolkit richness. Type in there:

length [remove-duplicates toolkit] of turtle 0

Your BehaviorSpace window should now look like this:



Figure 3.0.2. The BehaviorSpace window itself.

Hit OK, then hit Run. Choose if you want spreadsheet or table output, which is just a different way of arranging the data (you can click both, though we tend to prefer Table Output). BehaviorSpace also gives you the option of running the simulations in parallel. This isn’t necessary for a little model like this one, so you can change “**Simulations run in Parallel**” to 1, or run as many in parallel as you want. Once you hit OK again, you can name your file and run your simulation!



Figure 3.3. Run output options.

Some scientists (like the lead author of this publication) don’t particularly like the format of the data that BehaviorSpace produces. If you identify with this, you can write your own output, which we will demonstrate below.

Return to the code tab. Scroll to the very bottom, where we are going to write our own .csv output file.

Write this code:

to start-file

 if (file-exists? "Tutorial\_Output.csv")

 [

 carefully

 [file-delete "Tutorial\_Output.csv"]

 [print error-message]

 ]

file-open "Tutorial\_Output.csv"

 file-print "behaviorspace-run-number,tick,toolkit size"

 file-close

end

to write-file

 file-open "Tutorial\_Output.csv"

 file-print (word behaviorspace-run-number "," ticks "," length toolkit)

 file-close

end

Briefly, what this code does is create a file. If the file already exists it deletes the old version, so you’re not just appending the same file over and over every time you run a new parameter sweep. It records the run number from BehaviorSpace, the timestep, and the length of the toolkit. To get this to work, you have to add write-file and start-file earlier in the code. Put start-file at the end of your setup procedure right before end. Put write-file in your go procedure (within the turtle context), which will have it write new data every tick.

Note that we have now given you multiple ways to output your data: your own written .csv, export-plot, export-world, export-view, and the native export in BehaviorSpace.

Once this data is exported you can then analyze the data. Unfortunately, there is no time or space to discuss in this tutorial how to best analyze the data. Indeed, another tutorial would be needed to explain how to use R or another data science library! But there are many tutorials online on how to input this kind of data and create line graphs, for example:

https://simulatingcomplexity.wordpress.com/2014/11/12/creating-gorgeous-graphics-in-r/

**3.3 Packaging your simulation for public consumption**

The penultimate step in finishing your simulation is to finally open the “Info” tab. You’ll see many purple-highlighted bits of text. Each of these purple highlights are for standard headings for describing a model. Take your time and fill some of these out. For more information on how these should be filled out, read Grimm et al.’s (2010) description of the ‘Overview, Design concepts, and Details’ standardized protocol for ABMs. This is where you get to describe to your stakeholders why this model is important, how it works and what it shows.

Finally, your last step in packaging a NetLogo model for use by stakeholders is to save the model with the parameter values you wish the stakeholder to use on the outset. Do not save the model with strange values, as when the simulation is opened by the user it will open to those erroneous values.

Turn time-limit to something sensible (e.g., 100000 steps), and set max-carry to 50. Save the model.

And that’s it. You created a model, from start to finish, and now you can use it communicate your results to the public. Congratulations!

**3.4 Code (comments preceded by semi-colons)**

;;This code is associated with a forthcoming publication:

;;S. Crabtree, Harris, K., Davies, B., and I. Romanowska. USING AGENT-BASED

;;MODELS FOR ARCHAEOLOGICAL OUTREACH. Submitted May 2018 to Advances in

;;Archaeological Practice.

;;Additional materials can be found at REPO GOES HERE

; ADD GIS EXTENSION

extensions [ gis ]

; GLOBAL VARIABLES

globals [ elevation quarries ]

; AGENT (TURTLE) VARIABLES

turtles-own [ toolkit ]

; PATCH VARIABLES

patches-own [ patch-elevation assemblage material-type]

; SETUP PROCEDURE

to setup

 ; clear model

 clear-all

 ; load patch-elevation data from ascii raster

 set elevation gis:load-dataset "dem.asc"

 ; load the lithic source vector data

 set quarries gis:load-dataset "quarries.shp"

 ; resize the world to fit the patch-elevation data

 gis:set-world-envelope gis:envelope-of elevation

 ; add elevation data to patch data

 let mx gis:maximum-of elevation

 ask patches [

 set patch-elevation ( gis:raster-sample elevation self )

 ifelse patch-elevation > 0 [

 set pcolor scale-color green patch-elevation 0 mx

 ]

 [

 set pcolor blue

 ]

 ]

 ; create an agent with an empty toolkit list

 ask one-of patches with [ patch-elevation > 0 ] [

 sprout 1 [

 set size 8

 set shape "person"

 set color yellow

 set toolkit []

 ]

 ]

 ;give patches empty assemblage lists

 ask patches [

 set assemblage []

 ]

 start-file

 ;reset time

 reset-ticks

end

; GO PROCEDURE

to go

 ; move agents based on elevation

 ask turtles [

 if random 9 > 0 [

 move-to one-of neighbors with [ patch-elevation > 0 ]

 if [ patch-elevation ] of patch-here > elevation-threshold [

 repeat (elevation-multiplier - 1) [

 move-to one-of neighbors with [ patch-elevation > 0 ]

 ]

 ]

 ]

 ]

 ; if any quarries nearby, reprovision

 ask turtles [

 if gis:intersects? quarries patch-here [

 reprovision-toolkit

 ]

 ]

 ; if any tools in toolkit, discard

 ask turtles [

 if length toolkit > 0 [

 discard-tools

 ]

 ]

 ; if simulation reaches 100,000 time steps, recolor patches

 ; based on artifact density and stop simulation

 if ticks = time-limit [

 ask patches with [ length assemblage > 0 ] [

 set pcolor scale-color red length assemblage 0 30

 ]

export-view (word "Tutorial\_tick\_" ticks ".png")

 export-plot "Toolkit Richness" "plot.csv"

 stop

 ]

 ; advance time

 tick

end

; PROCEDURE FOR REPROVISIONING

to reprovision-toolkit

 ; stores the ID of a nearby quarry as a temporary variable t

 let t gis:property-value first (filter [ q -> gis:contained-by? q patch-here] (gis:feature-list-of quarries)) "ID"

 ; repeatedly adds value t to toolkit until toolkit is full

 while [ length toolkit < 100 ] [

 set toolkit lput t toolkit

 ]

end

; PROCEDURE FOR DISCARDING TOOLS

to discard-tools

 ; selects random item from toolkit

 let i random length toolkit

 ; adds that item to the local assemblage

 ask patch-here [

 set assemblage lput (item i [ toolkit ] of myself) assemblage

 ]

 ; removes that item from toolkit

 set toolkit remove-item i toolkit

end

; PROCEDURE FOR CREATING OUTPUT FILE

to start-file

 ; if file with the same name already exists, deletes existing file

 if (file-exists? "Tutorial\_Output.csv")

 [

 carefully

 [file-delete "Tutorial\_Output.csv"]

 [print error-message]

 ]

 ; open new output file

 file-open "Tutorial\_Output.csv"

 ; print column headings to output file

 file-print "behaviorspace-run-number,tick,toolkit size"

 ; close output file

 file-close

end

; PROCEDURE FOR WRITING DATA TO OUTPUT FILE

to write-file

 ; open output file

 file-open "Tutorial\_Output.csv"

 ;print the run number, ticks, and

 file-print (word behaviorspace-run-number "," ticks "," length toolkit)

 ; close output file

 file-close

end