Summary of NetLogo

This document provides a general overview of NetLogo structure and syntax and can be used as a glossary alongside the tutorials (Romanowska et al. 2019; Crabtree et al. 2019; Davies et al. 2019) or as a quick 'cheat-sheet'.

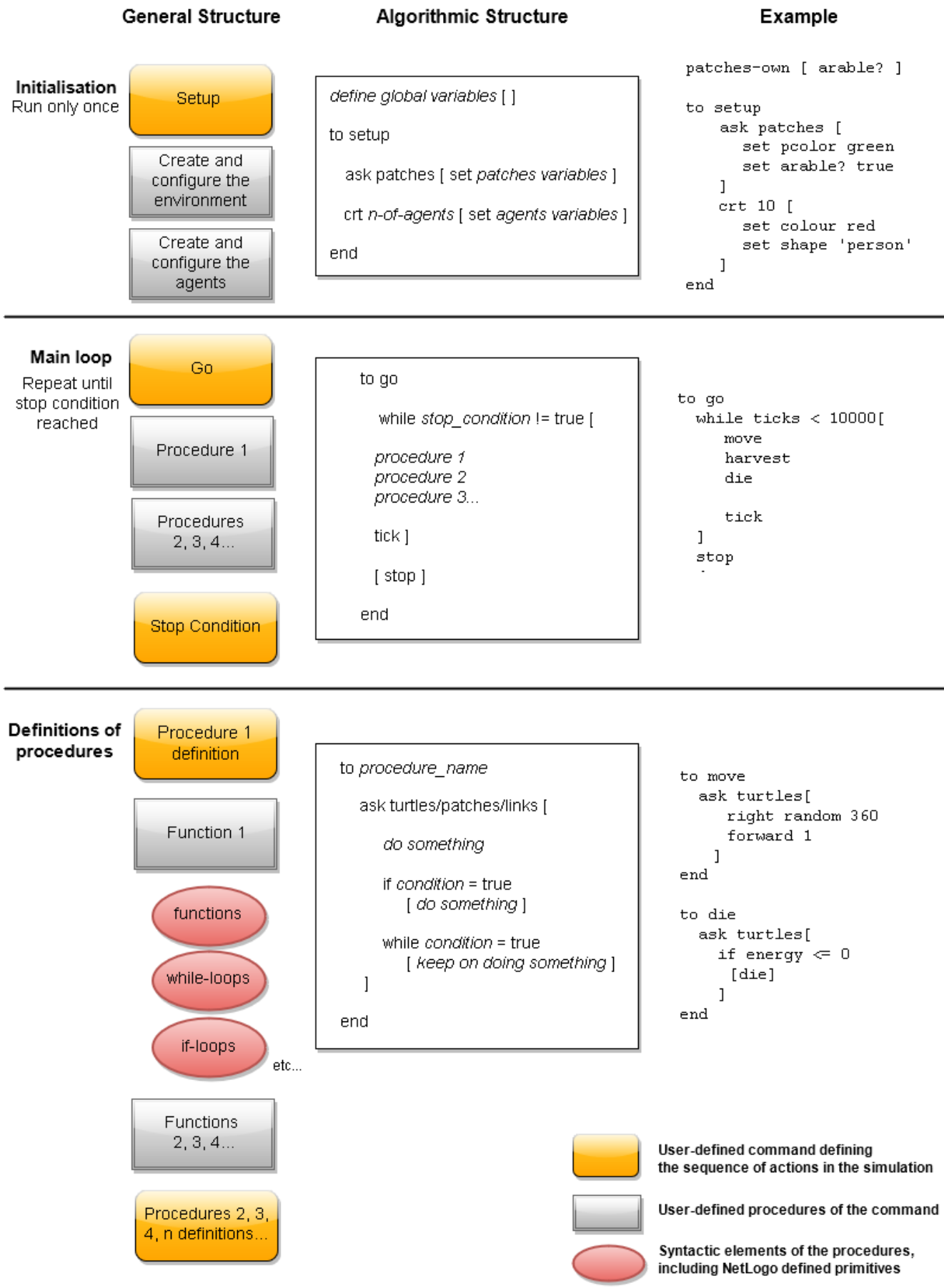
NetLogo (Wilensky 1999) is a user-friendly simulation platform commonly used for agent-based modeling in social and natural sciences. It is based on the Logo language originally designed as an educational tool for teaching programming to kids, making it a 'low threshold, high ceiling' platform. NetLogo combines ease of use and quick development with high level capacity and a wide suite of built-in tools such as visualizations, automated scenario running, etc. There are a number of other ABM platforms (RePast, Mason, AnyLogic; overview: Abar et al. 2017) and simulations can also be built using any of the general use programming languages (Python, C++, Java). NetLogo is by far the most dominant ABM platform in Archaeology (Davies and Romanowska 2018) and is very popular in social sciences and ecology. However, like every tool, NetLogo has some limitations which we discuss at the end of the document.

1.1. Installation

NetLogo can be downloaded from https://ccl.northwestern.edu/netlogo/. It is available for Windows, Mac OS X and Linux. The installation is a simple "point and click" and in most cases is unproblematic. In case of any issues it is worth consulting the FAQ of the NetLogo User Manual (https://ccl.northwestern.edu/netlogo/requirements.html).

1.2. Code building blocks

The four main entities in Netlogo are the agents ('turtles'), the grid squares ('patches'), the connections between agents ('links') and the observer. The observer governs the simulation flow, for example, by progressing the time counter or scheduling the order of actions. The building blocks in NetLogo consist of commands and reporters. Built-in commands are called 'primitives,' user defined ones are called 'procedures' and 'reporters'. The latter calculate a value and then report it. Most NetLogo simulations are composed of two main procedures: **`to setup`** and **`to go`**. In the `setup` procedure the world and the agents are initialized and it is executed only once at the beginning of a run. Setup may include loading up the GIS raster, creating the initial population of agents and giving them a set of characteristics (e.g., sex, age, profession). The `go` procedure is the core of a model and consists of a list of actions (often themselves procedures) that are repeated at each time step of the simulation until the stop condition is reached. All procedures come from the observer environment and are defined using the `to` and `end` keywords. If not specific to the observer commands apply to one of NetLogo entities: turtles, patches, or links. Therefore, they are always enclosed between square brackets following the keywords: `ask` *`turtles/patches/links`*. Figure 1 shows the common structure of NetLogo code with a list of procedures inside the `go` and the '`ask turtles/patches`', etc. command inside each of the procedure's definitions. It is important not to have '`ask turtles`' inside another '`ask turtles`', e.g., do not use `ask turtles [move]` if there is another '`ask turtles`' inside the '`move`' procedure - it will inevitably give you an error.


General Structure
Algorithmic Structure
Example
Initialisation
Run only once
Setup
Create and configure the environment
Create and configure the agents
define global variables [ ]
to setup
ask patches [set patches variables]
crt n-of-agents [set agents variables]
end
patches-own [arable?]
to setup
ask patches [
set pcolor green
set arable? true
]
crt 10 [
set colour red
set shape 'person'
]
end
Main loop
Repeat until stop condition reached
Go
Procedure 1
Procedures 2, 3, 4...
Stop Condition
to go
while stop_condition != true [
procedure 1
procedure 2
procedure 3...
tick]
[stop]
end
to go
while ticks < 10000[
move
harvest
die
tick
]
stop
Definitions of procedures
Procedure 1 definition
Function 1
functions
while-loops
if-loops
etc...
Functions 2, 3, 4...
Procedures 2, 3, 4, n definitions...
to procedure_name
ask turtles/patches/links [
do something
if condition = true
[do something]
while condition = true
[keep on doing something]
]
end
to move
ask turtles[
right random 360
forward 1
]
end
to die
ask turtles[
if energy <= 0
[die]
]
end
User-defined command defining the sequence of actions in the simulation
User-defined procedures of the command
Syntactic elements of the procedures, including NetLogo defined primitives

1.3. Variables

There are two types of variables in NetLogo: global and local variables. Global variables are defined at the beginning of the code and ascribed to one of the NetLogo entities: the observer (`globals []`), agents (`turtles-own[]`), grid squares (`patches-own[]`) or links (`links-own[]`). In the Interface tab, button, slider, and chooser also define global variables. Global variables can be used throughout the code and accessed via any procedure. Usually, we use global variables for things that are set at the beginning of the run and do not change during it, such as the initial number of agents, or the capacity of agents' backpack.

In contrast, local variables, defined by the `let` primitive, can only be accessed from within the procedure it has been defined in. For example, the following line of code:

```
let old_turtles turtles with [age > 50]
```

defines `old_turtles` as all turtles whose attribute `age` is greater than 50. We can then ask all 'old turtles' to do a specific task that is different from the one performed by youngsters. However, in the next time step, the list of 'old turtles' will change as some of them would have died while others would have reached the required age. Thus, local variables are dynamic and changeable throughout the run.

1.4. Control statements

Similar to all programming languages, NetLogo supports three main types of loops: if-loops, while-loops and for-loops (the latter will be discussed in section 1.4, Lists). If the conditional statement of an if-loop evaluates to 'true' a list of actions specified in the square brackets is performed. For example, the following command:

```
ask turtles[
      if energy <= 0 [die]
]
```

can be read as 'ask each turtle: if your energy is equal or lower than zero, please die'. In contrast, the while loop keeps on repeating the set of actions enclosed by the square brackets until the specified condition is reached. For example, this command:

```
ask turtles [
      while energy > 0 [move]
]
```

can be read as 'ask each turtle to move as long as their energy is greater than zero'. It is important to ensure that the while loop can (and will) reach the condition, otherwise the code will be run forever (or rather, until your computer's memory limit is reached - this is known as the 'stack overflow'). For example, this will give the while-loop a closure:

```
ask turtles [
      while energy > 0 [move]
      set energy energy - 1
]
```

1.5. Lists

A list is an object that stores multiple pieces of information. Lists are useful for keeping track of groups of things where group membership might change over time; for example, a forager's toolkit. A list can be initiated using the `set` primitive:

```
set example_list [10 20 30 40 50]
```

Alternatively, an agentset can be used to construct a list through the primitive `of`, e.g.:

```
set turtle_ages [ age ] of turtles
```

In this case the list `turtle_ages` contains the values of the `age` variable of each turtles.

Although lists are immutable, new lists can be created on the basis of existing lists, again using the `set` primitive.

```
set example_list replace-item 2 example_list 25
```

The '`2`' in the example above represents the index of the list, i.e. the position of the item which is to be replaced with the value '`25`'. Similar, to most programming languages the indexing in NetLogo starts with zero, i.e., the index of the first element of the list is 0, the second: 1, the third: 2, etc.

The for-loop mentioned above allows the user to perform an action on each element of the list. If we would like to inspect the content of the `example_list` after it was altered we can use:

```
foreach example_list show
```

and the result should be (if you executed the previous example):

```
10
25
30
40
50
```

We can also use `foreach` in conjunction with the arrow reporter to iterate through a list. For example

```
(foreach example_list example_list
  [ [a b] -> show word "the sum is: " (a + b) ])
```

Which tells NetLogo to use example_list as input and add corresponding integers together (e.g., 10+10) and report that in the command line.

The result should be (if you executed the previous example):

```
20
50
60
80
100
```

The examples above are intended as a general reference only. We will guide the reader through the process of building a simulation in NetLogo and discuss the code elements in a more comprehensive manner in the tutorial.

1.6. NetLogo resources

There are many freely-available learning resources for ABM and NetLogo on the Internet. NetLogo documentation (NetLogo 2018), which includes tutorials, a programming guide and a full dictionary of NetLogo primitives is usually the first port of call for any technical inquiries. However, there are many other ABM- and NetLogo- dedicated websites, blogs, code repositories and user groups. **simulatingcomplexity.wordpress.com** run by the authors of this tutorial is a specialized blog on archaeological applications of computational modeling and complexity science. The Special Interest Group in Complex Systems Simulation holds an annual beginner workshop in NetLogo as well as sessions and roundtables at the Computer Application and Quantitative Methods in Archaeology - CAA conference (https://caa-international.org/), while the European Social Simulation Association - ESSA (http://www.essa.eu.org/) organizes a week-long summer school in social simulation as a satellite to its annual conference. In addition, the Complex Systems Society annual conference - CSS (https://cssociety.org/) usually features at least one session (satellite) dedicated to archaeology. There are other, domain specific training courses available.

There are a number of university-level textbooks which use NetLogo to show the principles of complex systems modeling in ecology (Railsback and Grimm 2011), geography (O'Sullivan and Perry 2014), social science (Gilbert and Troitzsch 2005; Miller and Page 2007) and economy (Hamill and Gilbert 2016).

Not all simulations need to be written from scratch. NetLogo comes with a large and growing library of models. With over two hundred agent-based models the Models Library (accessible through the user interface of NetLogo) contains many examples, which, although developed for other disciplines from mathematics and physics to ecology and transport, could be adapted to archaeological research. For instance, epidemiological models simulating the spread of a disease in human populations under different conditions share many characteristics with theoretical models of the diffusion of innovations. In addition, many authors working on archaeological case studies share their model's code after publication in the OpenABM (https://www.comses.net/) repository and increasingly also on GitHub (https://github.com/).

1.7. When to switch to a different platform

Given how easy it is to develop agent-based models in NetLogo and how popular it is among social scientists and in other disciplines, you may ask yourself: 'why would anyone use any other software?'

Like every tool, NetLogo has its tradeoffs and it is important to know its 'weaknesses' in order to minimize their impact on one's research. The three main limitations of NetLogo are: 1. The high level of the programming language, 2. The lack of some of the standard software development tools, and 3. The performance.

The high level of the programming language is the exact reason why it is not difficult to learn to code in NetLogo. You 'ask turtles' to 'forward 1'. It's easy to write, it's easy to read, but what exactly does it do? The simplicity of coding comes at a price of not having a full control over the code. For example, it is all too common to (erroneously) assume that the primitive 'forward 1' makes all turtles move to the next patch ahead. It looks so obvious that hardly anyone checks the documentation to see whether this is actually the case (it is not). But these checks should be done. NetLogo documentation is extensive, thorough and openly available. It is important to constantly test the code during development to be sure it is doing exactly what one thinks it is doing. Unfortunately, NetLogo lacks

some of the standard software development tools, for example, for testing code. This is a weakness that we hope will be soon addressed by its developers, but in the meantime it is crucial that the modeler runs a wide variety of tests to minimize the risk of code errors. It is commonly done using the 'print' primitive (e.g., by asking one of the turtles to print out the results of calculations it is performing and checking that they are within the expected ranges) or by running 'special' scenarios (e.g., with no agents or only one agent to check that the algorithms function correctly). Also, the 'inspect' and 'watch' functions as well as the 'pen-down' primitive may come very handy in checking the code.

Finally, the performance issue. NetLogo is not a tool for highly optimized simulations. Also, it has no code parallelization capacity or any other support for HPC (High Performance Computing) so even a supercomputer may not be able to save you. If you need to run your simulation 300,000 times, NetLogo is not going to cut it. This limits, for example, the parameter ranges one can test or the number of runs performed to deal with the stochasticity of the model. It is common that the modeler finds themself (usually about 6 months before the end of a project) with a model that takes 20 minutes to run from start to finish. If they need to run scenarios with 4 parameters, and each one has 10 values that need to be tested, plus each run needs to be repeated 10 times because of stochasticity then we are talking about three and a half years before the results come in and let's hope that the code does not have a bug and needs to be run again. However, there are tools that can help optimize the code, in particular, the profiler (https://simulatingcomplexity.wordpress.com/2015/03/23/netlogo-profiler/)

On the other hand, it is much easier to implement a working simulation developed in NetLogo in one of the fast programming languages, such as Python or Java, than to develop it there from scratch. Thus, it is not unusual for archaeologists to start their ABM adventure with NetLogo and then move to other programming languages and simulation platforms while still using NetLogo to prototype their models. As one of the reviewers of this paper noted: "I would not make the argument that learning Netlogo will make you an ABM expert over all platforms. It is a good start though." We hope that this set of tutorials will help you kick start such a journey.

To cite this document:

Romanowska, Iza, Stefani Crabtree, Benjamin Davies, and Kathryn Harris
2019 "Agent-based Modeling for Archaeologists. A step-by-step guide for using agent-based modeling in archaeological research (Part I of III)." Advances in Archaeological Practice 7 (2).

References cited

Abar, Sameera, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare
2017 "Agent Based Modeling and Simulation Tools: A Review of the State-of-Art Software." *Computer Science Review* 24: 13-33. Elsevier Inc.: 13–33. doi:10.1016/j.cosrev.2017.03.001.

Crabtree, Stefani, Kathryn Harris, Benjamin Davies, Iza Romanowska
2019 "Outreach in Archaeology with Agent-based modeling. A step-by-step guide for using agent-based modeling in archaeological research (Part III of III)." Advances in Archaeological Practice 7 (2).

Davies, Benjamin, Iza Romanowska, Kathryn Harris, Stefani Crabtree
2019 "Combining Geographic Information Systems and Agent-Based Models in Archaeology: A step-by-step guide for using agent-based modeling in archaeological research (Part II of III)." Advances in Archaeological Practice 7 (2).

Davies, Benjamin, and Iza Romanowska

2018 “An Emergent Community? Agent Based Modelers in Archaeology.” *The SAA Archaeological Record* 18(2): 27–32.

Gilbert, Nigel G., and Klaus G. Troitzsch
2005 *Simulation for the Social Scientist*. Open University Press, Maidenhead.

Hamill, Lynne, and Nigel Gilbert
2016 *Agent-based modeling in economics*. Wiley, Chichester.

Miller, John H., and Scott E. Page
2007 *Complexity in Social Worlds*. Princeton University Press, Princeton.

O’Sullivan, David, and George Perry
2013 *Spatial Simulation: Exploring Pattern and Process*. Wiley-Blackwell, Chichester.

OpenABM
2014 Open Agent Based Modeling Consortium. A node in the CoMSES Network. https://www.openabm.org

Railsback, Steven F., and Volker Grimm
2011 *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, Princeton.

Romanowska, Iza, Stefani Crabtree, Benjamin Davies, and Kathryn Harris
2019 “Agent-based Modeling for Archaeologists. A step-by-step guide for using agent-based modeling in archaeological research (Part I of III).” Advances in Archaeological Practice 7 (2).

Wilensky, Uri
1999 “NetLogo.” Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston. https://ccl.northwestern.edu/netlogo/, accessed February 2, 2016.