OSC-NETLOGO: A TOOL FOR EXPLORING THE SONIFICATION OF COMPLEX SYSTEMS USING NETLOGO

Rodrigo F. Cadiz, Marco Colasso

Center for Research in Audio Technologies
Music Institute, Pontificia Universidad Catolica de Chile
rcadiz@uc.cl marco.colasso@gmail.com

ABSTRACT

In this article, we introduce OSC-NETLOGO, a tool that allows the creation of sonic phenomena by taking advantage of NetLogo's power for designing and building models of complex systems. NetLogo is a multi-agent programming language and modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems that evolve dynamically over time. We provide two examples taken from NetLogo's library of models. These examples provide evidence for the capabilities and potential of NetLogo as a sound synthesis tool. We hope that this tool could be of aid in future efforts of creating new complex sounds and interesting musical material.

1. INTRODUCTION

NetLogo is a multi-agent programming language and modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems evolving over time [4]. NetLogo comes from the Logo family of programming languages [3] and has expanded the original Logo concept in a number of ways.

NetLogo allows modelers to give instructions to hundreds or thousands of independent agents all operating concurrently, which is something esential for modeling complex systems. This makes it possible to explore connections between micro-level behaviors of individuals and macro-level patterns that emerge from their interactions. NetLogo enables users to open simulations and play with them, exploring their behavior under various conditions by manipulating several graphical objects suchs as sliders or buttons. NetLogo is also an authoring environment that enables users to create their own models, and try them on the fly.

Acording to the NetLogo website [8], NetLogo is being used to build an endless variety of simulations. Members of the NetLogo community have turned turtles into molecules, wolves, buyers, sellers, bees, tribespeople, birds, worms, voters, passengers, metals, bacteria, cars, robots, neutrons, magnets, planets, shepherds, lovers, ants, muscles, networkers, and more. Patches have been made into trees, walls, terrain, waterways, housing, plant cells, cancer cells, farmland, sky, desks, fur, sand, etc. [4].

Turtle agents and patches are useful to visualize and study mathematical abstractions, specially non-linear dynamical systems, to display behavior in graphical ways, to make art and even play games. NetLogo comes with a very big library of models, which covers topics such as cellular automata, genetic algorithms, positive and negative feedback, evolution and genetic drift, population dynamics, networks, markets, chaos theory, swarming behavior, and molecular physics. All of these models share core concepts such as complex systems, self-organization and emergence.

Although NetLogo is very powerful for modeling and handling complex systems data, and provides some basic audio functionality through MIDI, it lacks more serious digital audio generation and processing capabilities. However, one good thing about NetLogo is that is provides an API for programmers to develop extensions to the program in Java. We took advantage of this feature and developed OSC-NETLOGO, an Open Sound Control (OSC) [9] extension to Netlogo using the JavaOSC library [2]. This extension allows users to directly map any parameter of a NetLogo patch into any sound processing environment such as Max/MSP, Pd or SuperCollider using OSC.

This article is structured as follows. In section 2 we describe the NetLogo application in more detail, including its history, API. In section 3 we describe the netlogo-osc extension, including its installation and usage. Then, in section 4 we provide examples of mappings of two NetLogo models into sound synthesis using Pd. Finally, in section 5 we discuss the main findings and conclusions of our work.

2. NETLOGO

NetLogo is a cross-platform standalone application written in Java. It has been being developed for more than ten years, which assures that NetLogo is a mature product that is stable and fast. It is freeware, anyone can download it for free and build models without restriction. It also comes with extensive documentation and tutorials and a large collection of sample models, created both by NetLogo developers and the general community of users [4].

2.1. History

As a language, NetLogo adds agents and concurrency to Logo. Logo, as originally developed by Seymour Papert and Wally Feurzeig in 1968, is derived from Lisp, but has a friendlier syntax [5]. The primary objetive of Logo was to provide a tool to teach programming to perfectly typical primary school students, an intention which it successfully fulfilled [1]. In consequence, Logo was designed as a programming language usable by children as well as adults and is still popular today for that purpose.

From Logo, Netlogo inherits the widely known *turtle*. In traditional Logo, the programmer controls a single turtle; a NetLogo model can have thousands of them [4]. According to [5], although there is no single agreed upon standard for the Logo language, NetLogo shares enough syntax, vocabulary, and features with other Logos to earn the Logo name.

2.2. Structure

The NetLogo world is made up of programmable agents, which are beings that can follow instructions. Agents can interact with each other and perform multiple tasks concurrently. In NetLogo, there are four types of agents: turtles, patches, links, and the observer.

Turtles are agents that move around in the world. The world is two dimensional and is divided up into a grid of *patches*. Each patch is a square piece of ”ground” over which turtles can move. *Links* are agents that connect two turtles. The *observer* does not have a location, it is looking out over the world of turtles and patches.

In NetLogo, commands and reporters tell agents what to do. A *command* is an action for an agent to carry out, resulting in some effect. A *reporter* is instructions for computing a value, which the agent then reports to whoever asked it. Commands and reporters built into NetLogo are called *primitives*. Commands and reporters defined by the user are called *procedures*.

Many NetLogo models have a once button that calls a procedure called *setup* and a forever button that calls a procedure called *go*. In NetLogo, it is possible to specify which agents are to run each command. If nothing is specified, the code is run by the observer.

2.3. Extensions API

NetLogo is a multi-agent modeling language and environment that continually strives toward a central design goal (shared with the original Logo programming language) to be both *low threshold* (easy for beginners to learn) and *no ceiling* (such that experts do not feel limited) [4]. A key aspect of the no ceiling goal is the extensibility of the language.

The NetLogo Extensions API provides facilities for programmers to extend the NetLogo language by creating user-defined language primitives. NetLogo extensions may be written in Java or any other language compatible with the JVM. One of the built-in extensions to NetLogo allows agents make sounds and music using Java’s MIDI capabilities. However, this extension does not allow to send data in a more flexible way and with a better resolution, being this the most important factor for the need of an OSC extension.

3. OSC-NETLOGO: AN OSC EXTENSION FOR NETLOGO

The OSC-NETLOGO Extension for NetLogo adds primitives to NetLogo that allow users to send data from NetLogo models via OSC to third party applications. Turtles, Breeds, Links, Patches and any other variables from a NetLogo patch can be routed to a compatible external hardware or software, via Ethernet.

3.1. Usage

The OSC-NETLOGO Extension including example Pd patches and audio files can be downloaded from `http://www.rodrigocadiz.com/osc-netlogo`. It can be installed in two different ways:

- Put the osc folder into the Extensions folder in the same location where the NetLogo application is installed.
- Put the osc folder into the current model folder.

After installing the OSC extension, in order to use it, it is mandatory to add the following line to the top of the procedures tab: `extensions [osc]`

3.2. Primitives

`osc:port-out`

This primitive should be used in the setup of the model. The user can set the IP (String) and port number (integer) of the receiving host, such as

```
(osc:port-out "169.254.183.129 " 10000)
```

If `port-out` is not defined, the extension will use the default parameters, ip: localHost and port: 57110.

`osc:send-agent` This primitive receives as input a string with the name of the OSC tag, followed by the name of an agentset of the model, and names of default or defined variables for this agentset, and sends them out.

The syntax for this primitive is `osc:send-agent "tagName" agentsetName var1 var2 var3 ...`

Examples:

```
(osc:send-agent "myTurtles" turtles "xcor"
"size")
(osc:send-agent "breeds" cars "XCOR" "age")
(osc:send-agent "links" blue-links "weight")
(osc:send-agent "patches" patch 2 2 "pcolor"
"pxcor")
```

`osc:send-variable variableName`

This primitive receives as input the name of any variable of the model and sends it out. Example:

```
(osc:send-variables "decays" decays)
```

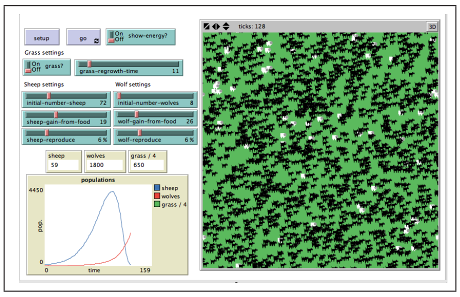
4. EXAMPLES

We selected two models from NetLogo model's library and generated audio mappings of some of the parameters of the models using osc-netlogo. The selected models were: Wolf Sheep Predation and Pursuit.

4.1. Wolf Sheep Predation

This Wolf Sheep predation model [7] explores the stability of predator-prey ecosystems. Such a system is called unstable if it tends to result in extinction for one or more species involved. In contrast, a system is stable if it tends to maintain itself over time, despite fluctuations in population sizes.

There are two main variations to this model. In the first variation, wolves and sheep wander randomly around the landscape, while the wolves look for sheep to prey on. Each step costs the wolves energy, and they must eat sheep in order to replenish their energy - when they run out of energy they die. To allow the population to continue, each wolf or sheep has a fixed probability of reproducing at each time step. This variation produces interesting population dynamics, but is ultimately unstable.



Grass settings
grass? On
grass-regrowth-time 11
Sheep settings
initial-number-sheep 72
sheep-gain-from-food 19
sheep-reproduce 6 %
Wolf settings
initial-number-wolves 8
wolf-gain-from-food 26
wolf-reproduce 6 %
sheep 59
wolves 1800
grass / 4 650

Figure 1. NetLogo screen for the Wolf Sheep predation model

The second variation includes grass (green) in addition to wolves and sheep. The behavior of the wolves is identical to the first variation, however this time the sheep must eat grass in order to maintain their energy - when they run out of energy they die. Once grass is eaten it will only regrow after a fixed amount of time. This variation is more complex than the first, but it is generally stable.

We have created a sonification of the second variation of this model using our extension, as shown in figures 1 and 2. In this example it is possible to hear how the sheep and wolves populations increase and decrease over time. The sheep sound is a simple oscillator that changes its frequency with the sheep count. The sound of the wolves consists of two oscillators with frequencies that change depending on the amount of wolves present in the model.

In figure 3, it is possible to clearly hear how the two populations change in time. Compare the curves in the sonogram to those ones shown in figure 1.

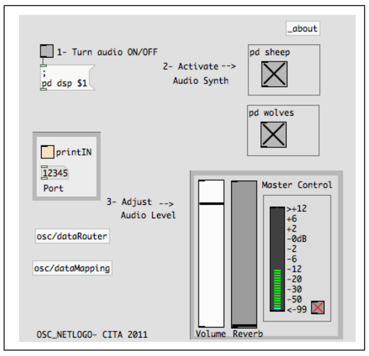


Figure 2. Pure data patch for the Wolf Sheep predation model

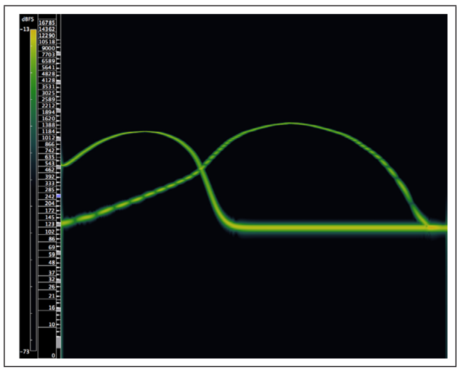
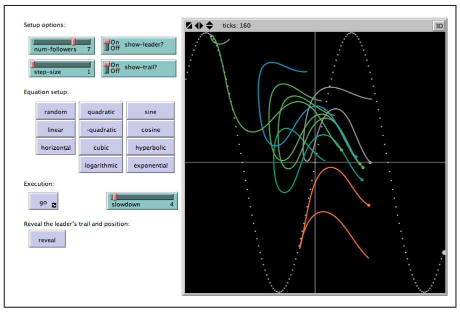


Figure 3. Sonogram excerpt for the Wolf Sheep predation model

4.2. Pursuit

In the Pursuit model [6] there is one leader turtle and a group of follower turtles. It is possible to show the leader's path or hide it and try to guess the path it's moving along. The idea of the model is that by watching the followers it is possible to obtain clues to the leader's path. In the sonic implementation of this model, the idea is to guess the leader's formula by listening to the followers.

The leader moves along a path according to a preselected formula, such as $y = x^2$ and starts at the left edge of the world. The leader always moves from left to right by one unit increments along the x-axis. The leader's y coordinate is based on the selected formula and the current x coordinate. Each follower turns to face the leader, then moves forward by a fixed amount.



num-followers 7
step-size 1
slowdown 4

Figure 4. NetLogo screen for the Pursuit model

In this example, shown in figures 4 and 5, it is possible to listen to either the leader or the followers. The *y* coordinate of each turtle controls the frequency of one sine wave oscillator, while the *x* coordinate is mapped to stereo panning. It is also possible to change the minimum and maximum frequencies in the `osc/dataMapping` box and set different frequency ranges for each breed as well.

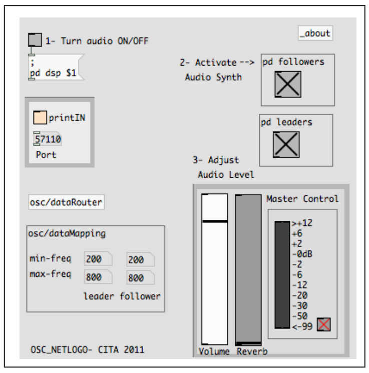


Figure 5. Pure data patch for the Pursuit model

In the sonogram of this example, shown in figure 6, the different pursuits and different paths of the leader and followers can be clearly observed. Also, the sonic shape that each oscillator produces can be audibly mapped to the equations governing the behavior of the leader.

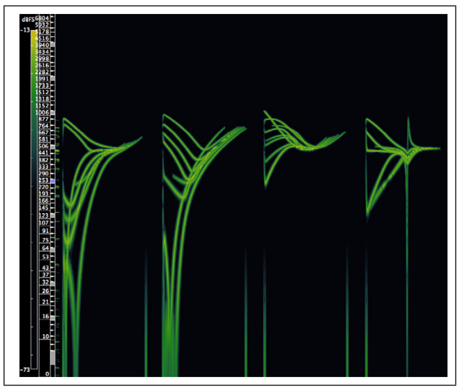


Figure 6. Sonogram excerpt for the Pursuit model

5. CONCLUSIONS

We have developed OSC-NETLOGO, a NetLogo extension tool that allows to create very complex sonic phenomena by taking advantage of NetLogo's power for designing and building models of complex systems.The extension is very simple to install and use and gives the possibilty of mapping any variable of a NetLogo model into an OSC-enabled audio synthesis engine.

We have provided two examples taken from the available models at the NetLogo's library of models. These examples provide evidence for the capabilities and potential of NetLogo as a sound generating and processing tool. The behavior of complex systems is something that is very appealing from a musical standpoint, and we hope that this tool could be of aid in the efforts of creating new complex sounds and interesting musical material.

6. ACKNOWLEDGEMENTS

This research was funded by Fondecyt Grant #11090193, Conicyt, Government of Chile.

7. REFERENCES

[1] A. Hechmer, A. Tindale, and G. Tzanetakis, "Logorhythms: Introductory audio programming for computer musicians in a functional language paradigm," in *Frontiers in Education Conference, 36th Annual*, San Diego, CA, 2007.

[2] Illposed, *Java OSC*. http://www.illposed.com/software/javaosc.html, 2006.

[3] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1980.

[4] S. Tisue and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in *International Conference on Complex Systems*, Boston, MA, 2004.

[5] ——, "Netlogo: Design and implementation of a multi-agent modeling language," in *SwarmFest*, Ann Arbor, MI, 2004.

[6] U. Wilensky, *NetLogo Pursuit model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.: http://ccl.northwestern.edu/netlogo/models/Pursuit, 1998.

[7] ——, *NetLogo Wolf Sheep Predation model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.: http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation, 1998.

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

[9] M. Wright and A. Freed, "Open sound control: a new protocol for communicating with sound synthesizers," in *Proceedings of the International Computer Music Conference*, Thessaloniki, Greece, 1997.