Sayama *Complex Adaptive Systems Modeling* 2013, **1**:2
http://www.casmodeling.com/content/1/1/2

Complex Adaptive Systems Modeling
a SpringerOpen Journal

METHODOLOGY Open Access

PyCX: a Python-based simulation code repository for complex systems education

Hiroki Sayama

Correspondence:
sayama@binghamton.edu
Collective Dynamics of Complex Systems Research Group, Departments of Bioengineering & Systems Science and Industrial Engineering, Binghamton University, State University of New York, Binghamton, NY, USA

Abstract

We introduce PyCX, an online repository of simple, crude, easy-to-understand sample codes for various complex systems simulation, including iterative maps, cellular automata, dynamical networks and agent-based models. All the sample codes were written in plain Python, a general-purpose programming language widely used in industry as well as in academia, so that students can gain practical skills for both complex systems simulation and computer programming simultaneously. The core philosophy of PyCX is on the simplicity, readability, generalizability and pedagogical values of simulation codes. PyCX has been used in instructions of complex systems modeling at several places with successful outcomes.

Keywords: PyCX, Python, Complex systems simulation, Education, Iterative maps, Cellular automata, Dynamical networks, Agent-based models

Background

Until nearly the end of the last century, dynamic simulations of complex systems—such as cellular automata and agent-based models—were only available to researchers who had sufficient technical skills to develop and operate their own simulation software. At that time, there were very few general-purpose simulation software packages available (e.g., (Hiebeler 1994; Wuensche 1994)), and those packages were rather hard to program, unless one had a computer science background. The lack of general-purpose simulation software easily accessible for non-computer scientists was a major limiting factor for the growth of complex systems science, given the highly interdisciplinary nature of the field.

Over the last decade, several easy-to-use complex systems modeling and simulation software packages have been developed and become widely used for scientific research, including NetLogo (Tisue and Wilensky 2004), Repast (Collier 2003), Mason (Luke et al. 2004) (for agent-based models) and Golly (Trevorrow et al. 2005) (for cellular automata). They have been playing a critical role in making complex systems modeling and simulation available to researchers outside computer science. A number of publications used these software packages as key research tools, and increasingly many online tutorials and sample simulation models are becoming publicly available.

However, such existing software has several problems when used for teaching complex systems modeling and simulation in higher education settings. These are all real issues we have faced in classrooms and other educational settings over the last several years.

Firstly, and most importantly for college students, learning languages or libraries specific to particular simulation software used only in academia would not help the students advance their general technical skills. Because most students will eventually build their careers outside complex systems science, they usually want to learn something generalizable and marketable (even though they want to study complex systems science and they do appreciate its concepts and values).

Secondly, even for those who actively work on complex systems research, choices of preferred software vary greatly from discipline to discipline, and therefore it is quite difficult to come up with a single commonly agreeable choice of software useful for everyone. This is particularly problematic when one has to teach a diverse group of students, which is not uncommon in complex systems education.

Thirdly, details of model assumptions and implementations in pre-built simulation software are often hidden from the user, such as algorithms of collision detection, time allocation and state updating schemes. As we all know, such microscopic details can and do influence macroscopic behavior of the model, especially in complex systems simulations.

Finally, using existing simulation software necessarily puts unrecognized limitations to the user's *creativity* in complex systems research, because the model assumptions and analytical methods are influenced significantly by what is available in the provided software. This is a fundamental issue that could hamper the advance of complex systems science, since any breakthroughs will be achieved only by creating entirely novel ways of modeling and/or analyzing complex systems that were not done before.

These issues in using existing simulation software for complex systems education leads to the following very challenging riddle:

Which computational tool is best for teaching complex systems modeling and simulation, offering students generalizable, marketable skills, being accessible and useful for everyone across disciplines, maintaining transparency in details, and imposing no implicit limit to the modeling and analysis capabilities?

Obviously, there would be no single best answer to this kind of question. In what follows, we present a case of our humble attempt to give our own answer to it, hoping that some readers may find it helpful for solving their unique challenges in complex systems education.

PyCX

Through several years of experience in complex systems education, we have come to realize that using a simple general-purpose computer programming language *itself* as a complex systems modeling platform is our current best solution to address most, if not all, of the educational challenges discussed above. By definition, general-purpose computer programming languages are universal and can offer unlimited opportunity of modeling with all the details clearly spelled out in front of the user's eyes.

Identifying a programming language that would be easily accessible and useful in a wide variety of disciplines had been difficult even a decade ago.[a] Fortunately, several easy-to-use programming languages have recently emerged and become very popular in various scientific and industrial communities, including Python and R. For our educational needs, we chose Python, a programming language now widely used in industries as well as in

academia, so that students can gain practical skills for both complex systems simulation and computer programming simultaneously.

Using the Python language itself as a modeling and simulation platform, we have developed "PyCX", an online repository of simple, crude, easy-to-understand sample codes for various complex systems simulation.[b] The target audiences of PyCX are researchers and students who are interested in developing their own complex systems simulation software using a general-purpose programming language but do not have much experience in computer programming. We carefully designed the sample codes so that our audience can understand, modify, create and visualize dynamic complex systems simulations relatively easily.

The core philosophy of PyCX is therefore placed on the simplicity, readability, generalizability and pedagogical values of simulation codes. This is often achieved even at the cost of computational speed, efficiency or maintainability. For example: (1) every PyCX sample code is written as a single .py file, which is a plain text file, without being split into multiple separate files; (2) all the dynamic simulations follow the same scheme consisting of three parts (initialization, visualization and updating); (3) we do not use the object-oriented programming paradigm because it is sometimes difficult for non-computer scientists to grasp; and (4) we do use global variables frequently to make the code more intuitive and readable. These choices were intentionally made based on our experience in teaching complex systems modeling and simulation to non-computer scientists and their feedback.

A simple example of PyCX sample codes ("abm-randomwalk.py", an agent-based simulation of particles moving in independent random walk in a two-dimensional open space) is shown below. For a dynamic simulation like this one, the user needs to define three functions (init for initialization, draw for visualization, and step for updating) and then call pycxsimulator.GUI().start() to run the simulation.

```
import matplotlib
matplotlib.use('TkAgg')

import pylab as PL
import random as RD
import scipy as SP

RD.seed()

populationSize = 100
noiseLevel = 1

def init():
    global time, agents

    time = 0

    agents = []
    for i in xrange(populationSize):
```

```
        newAgent = [RD.gauss(0, 1), RD.gauss(0, 1)]
        agents.append(newAgent)

def draw():
    PL.cla()
    x = [ag[0] for ag in agents]
    y = [ag[1] for ag in agents]
    PL.plot(x, y, 'bo')
    PL.axis('scaled')
    PL.axis([-100, 100, -100, 100])
    PL.title('t = ' + str(time))

def step():
    global time, agents

    time += 1

    for ag in agents:
        ag[0] += RD.gauss(0, noiseLevel)
        ag[1] += RD.gauss(0, noiseLevel)

import pycxsimulator
pycxsimulator.GUI().start(func=[init,draw,step])
```

There are a few limitations in using Python for complex systems education. The first limitation is that the software installation and maintenance is a little more difficult than other more sophisticated software environment.[c] We believe, though, that the long-term educational benefit from learning a general-purpose programming language will outweigh the short-term learning difficulty for the majority of students.

The second limitation is that it is relatively difficult to build an interactive GUI (Graphical User Interface) in Python. To alleviate this problem, we provide a simple universal GUI and a coding template for dynamic simulations. The user can define just three functions (for initialization, visualization and updating) in the "realtime-simulation-template.py" file to implement a dynamic simulation, and then the "pycxsimulator.py" module will automatically generate a minimalistic GUI with three buttons ("Run/Pause", "Step Once" and "Reset"; Figure 1). These two files must be placed within the same folder.

Implemented Sample Codes

We have implemented a number of sample codes for typical complex systems simulations, including iterative maps, cellular automata, dynamical networks and agent-based models. All of those codes were concisely written in plain Python. They are freely available from the project website.[b] To run those sample codes, the user will need the following software modules, which are also available for free and widely used for scientific computation in academia and industries:

- Python 2.7[d]
- NumPy and SciPy for Python 2.7[e]

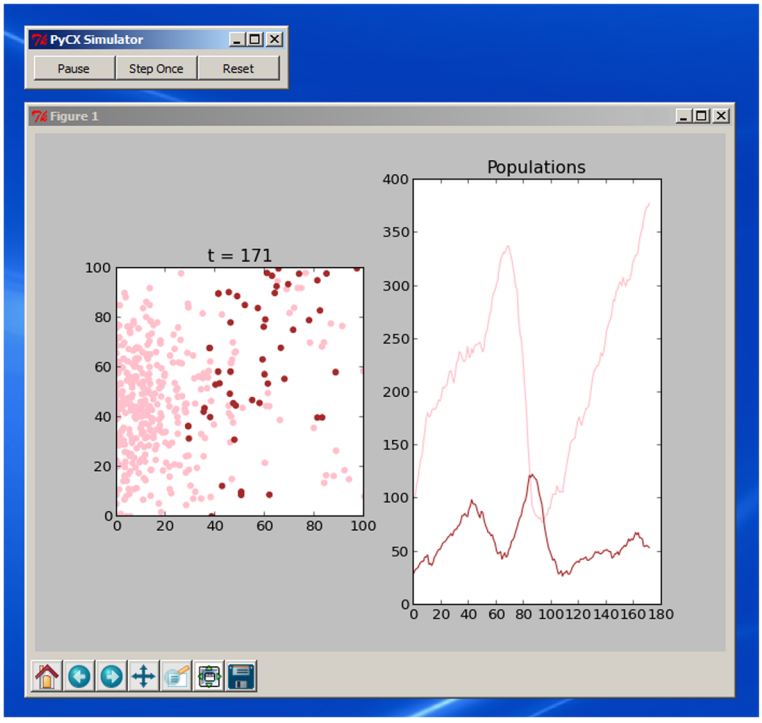

Figure 1 Minimal GUI (Graphical User Interface) generated by the "pycxsimulator.py" module. A three-button minimal GUI window is displayed at the top, together with the large main window showing the simulation result that is dynamically updated in real time. The main window also comes with several matplotlib-default GUI buttons for zooming and saving images, etc. (bottom). Simulations of cellular automata, dynamical networks and agent-based models are to be operated using this simple GUI.

- matplotlib for Python 2.7[f]
- NetworkX for Python 2.7 (this module is needed for dynamical network simulations only) (Hagberg et al. 2008)[g]

In most operating systems, running a PyCX sample code is just double-clicking on the file. Below is a selected list of simulations included in the most recent version of PyCX (version 0.2):

- Basic dynamical systems (logistic map, cobweb plot, bifurcation diagram, Lotka-Volterra equations; Figure 2)
- Cellular automata (local majority rule, droplet rule, Game of Life, Turing pattern formation, excitable media, host-pathogen interaction, forest fire; Figure 3)

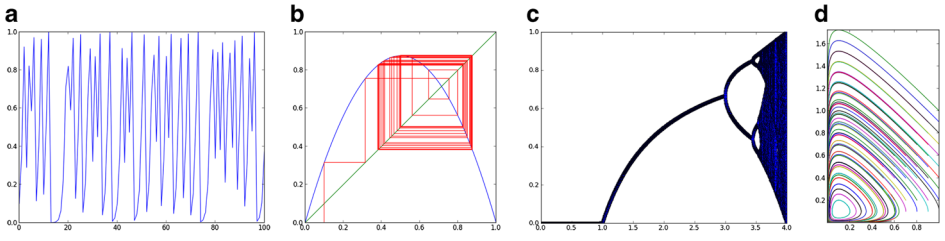

Figure 2 Screen shots of basic dynamical systems in PyCX. (**a**) Logistic map, (**b**) cobweb plot, (**c**) bifurcation diagram, (**d**) Lotka-Volterra equations.

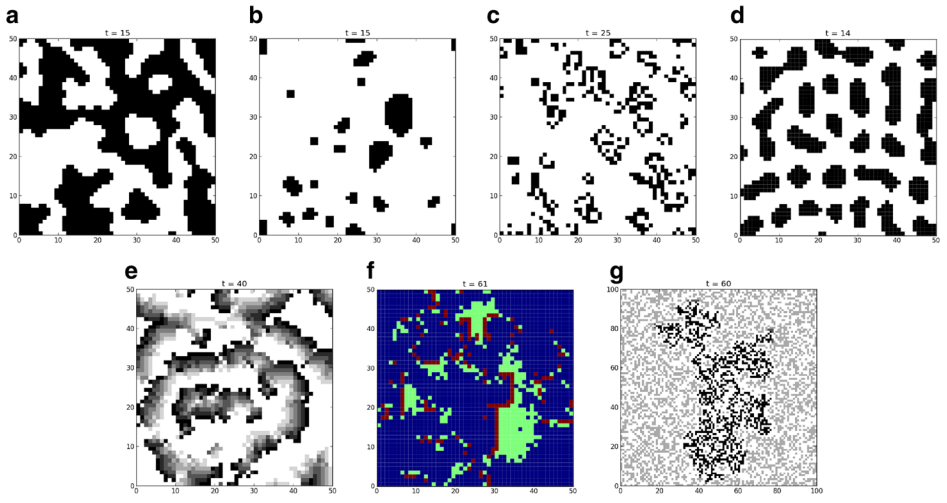

Figure 3 Screen shots of cellular automata in PyCX. (**a**) Local majority rule, (**b**) droplet rule, (**c**) Game of Life, (**d**) Turing pattern formation, (**e**) excitable media, (**f**) host-pathogen interaction, (**g**) forest fire.

- Dynamical networks (basic network construction and analysis, network growth by preferential attachment, local majority rule on a network, synchronization on a network based on the Kuramoto model (Strogatz 2000), disease spreading on a network, cascade of failures on a network; Figure 4)
- Agent-based models (random walk of particles, diffusion-limited aggregation, predator-prey ecosystem, garbage collection by ants (Resnick 1997), aggregation of ants via pheromone-based communication; Figure 5)

Actual use in the classrooms

PyCX has been used in instructions of complex systems modeling and simulation at several occasions, including:

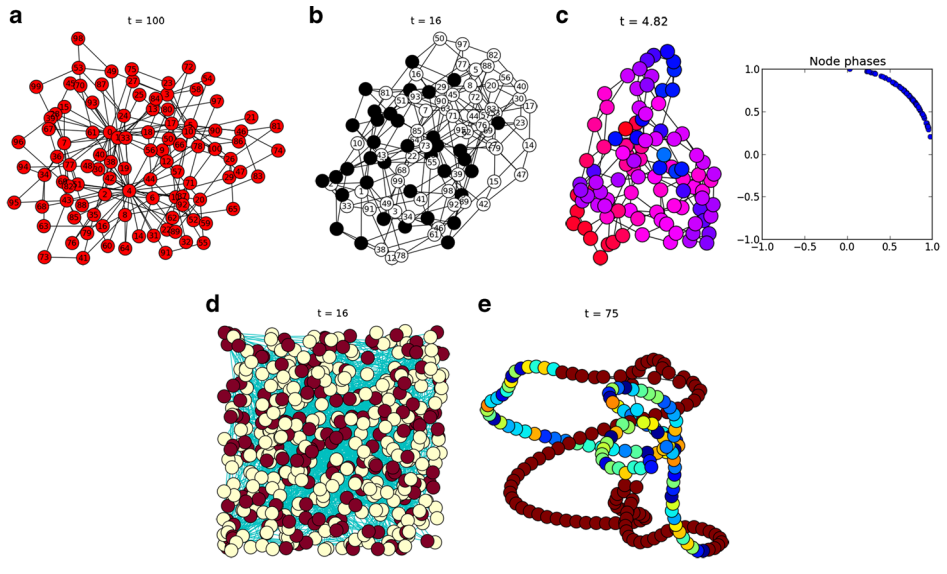

Figure 4 Screen shots of dynamical networks in PyCX. (**a**) Network growth by preferential attachment, (**b**) local majority rule on a network, (**c**) synchronization on a network based on the Kuramoto model (Strogatz 2000), (**d**) disease spreading on a network, (**e**) cascade of failures on a network.

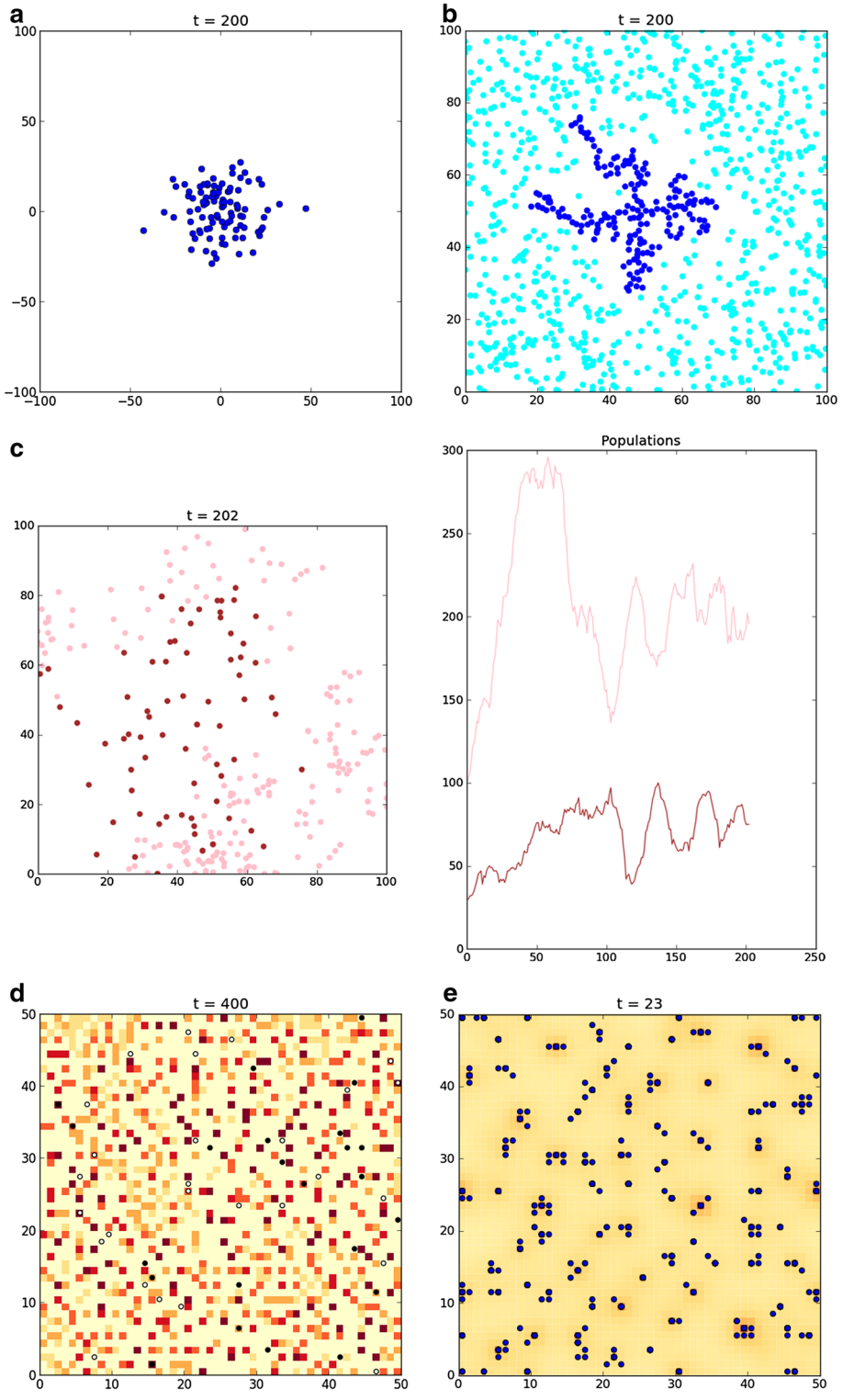

Figure 5 Screen shots of agent-based models in PyCX. (**a**) Random walk of particles, (**b**) diffusion-limited aggregation, (**c**) predator-prey ecosystem, (**d**) garbage collection by ants (Resnick 1997), (**e**) aggregation of ants via pheromone-based communication.

- Binghamton University Graduate Courses
  - BME-510: Modeling Complex Biological Systems (2009, 2010)
  - SSIE-518X/523: Collective Dynamics of Complex Systems (2011-)
  - BME-523X/523: Dynamics of Complex Networks (2012-)
- New England Complex Systems Institute Summer/Winter Schools[h]
  - CX 202: Complex Systems Modeling and Networks (2008-)
  - CX 102: Computer Programming and Complex Systems (2010-)
- NWO (Netherlands Organisation for Scientific Research) Complexity Winter School (2011)[i]
- NetSci High Summer Workshop (2012)[j]

In a typical curriculum, we ask students to bring in their own laptops to the classroom, and spend the first three to six hours for software installation and general introduction to Python, covering basics of its syntax and data structures. The rest of the curriculum can be custom-designed based on the students' interests and needs, by combining appropriate simulation sample codes as curricular units.

While each curricular unit can be taught in various ways, we have found the following instruction format most effective so far:

1. Describe the key concepts of the phenomenon being modeled, as well as the basic model assumptions.
2. Run the simulation sample code, show the results and have a brief discussion on the observations.
3. Open the code in an editor[k] and give a line-by-line walk-through, explaining how the model is implemented in detail and addressing any technical questions as needed.
4. Have a couple of in-class exercises that require students to understand and then modify the code to implement some model variations.
5. Summarize the learning experience of the curricular unit and have open Q&A's and/or try further model extensions.

In this format, each unit can be covered within 30 minutes to one hour, depending on how much details will be discussed. The basic goal of these activities is to make students feel comfortable in investigating into sample codes of their interest and then actively modifying them as a template for their own simulation models.

A systematic evaluation of PyCX's educational impacts is beyond the scope of this short article. However, we have gained several tangible outcomes, including:

- 100% "thumbs-up" user rating scores on SourceForge.net [l]
- Many highly positive testimonials from those who took courses with PyCX or who used PyCX (some testimonials are available on the project website)
- Several publications of papers on Python-based computer simulation by students and faculty who took courses with PyCX (Hao et al. 2010; Mischen 2010; Yamanoi and Sayama 2012)

Conclusion

We have presented PyCX, an online repository of sample codes of complex systems simulations designed primarily for higher education. It uses the Python programming language itself as a modeling and simulation platform, aiming at overcoming several limitations that arose from using existing complex systems simulation software for educational purposes.

PyCX has both new and old aspects. Its technical aspect is new because it is based on a rapidly growing programming language Python and its powerful add-on modules such as NetworkX for network modeling. In the meantime, its core philosophy remains very old, advocating that everyone should be writing codes of his or her own simulation models in order to explore highly unique, novel modeling frameworks and analysis methods, just like what complex systems researchers used to do in the twentieth century. We believe that, by raising the level of basic computer programming literacy among various disciplines through PyCX and other similar projects, the complex systems science community will be able to remain highly creative and innovative in its exploratory endeavors.

The PyCX project is still continuously evolving. We would highly appreciate comments, suggestions and contributions from researchers and educators who are interested in participating in it.

Endnotes

[a]Note that C, C++ and Java were too technical for most of non-computer scientists. Other commercial languages such as MATLAB and Mathematica were relatively easy to code, but neither freely available nor widely accepted across disciplines.
[b]http://pycx.sf.net/
[c]But this problem can be remedied by using a pre-packaged Python software suite such as Enthought (http://www.enthought.com/).
[d]http://python.org/
[e]http://scipy.org/
[f]http://matplotlib.org/
[g]http://networkx.lanl.gov/
[h]http://www.necsi.edu/education/school.html
[i]http://www.nwo.nl/en/research-and-results/programmes/complexity
[j]http://www.bu.edu/networks/
[k]We usually use IDLE, Python's default text editor, for our instructions.
[l]http://sourceforge.net/projects/pycx/reviews/ (as of November 19, 2012)

Competing Interests
The author declares that he has no competing interests.

Acknowledgements
We thank Chun Wong for his invaluable contribution in developing the GUI for real-time dynamic simulations.

Received: 22 November 2012 Accepted: 10 December 2012 Published: 13 March 2013

References

Collier, N: **Repast: An extensible framework for agent simulation.** 2003. http://repast.sf.net/.
Hagberg, AA, Schult DA, Swart PJ: **Exploring network structure, dynamics, and function using NetworkX.** In *Proceedings of the 7th Python in Science Conference*. Edited by Varoquaux, G, Vaught T, Millman J; 2008:11–15.
Hao, C, Gupta A, Paranjape R: **Pooling unshared information: building expertise and social ties in decision-making groups.**, Paper presented at *the 70th Academy of Management Annual Meeting*, Session # 1727: Decision Making in Organizations. Montreal, Canada; 2010.

Hiebeler, D: **The Swarm simulation system and individual-based modeling.** In *Proceedings of the Decision Support 2001: Advanced Technology for Natural Resource Management*. Toronto; 1994. http://www.santafe.edu/media/workingpapers/94-11-065.pdf.

Luke, S, Cioffi-Revilla C, Panait L, Sullivan K: **Mason: A new multi-agent simulation toolkit.** In *Proceedings of the 2004 SwarmFest Workshop* (Vol. 8); 2004.

Mischen, PA: **Information sharing and knowledge sharing in interorganizational networks.**, Paper presented at *the Computational Social Science Society Annual Conference*. Tempe, AZ; 2010.

Resnick, M: *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: MIT Press; 1997.

Strogatz, SH: **From Kuramoto to Crawford: exploring the onset of synchronization in populations of coupled oscillators.** *Physica D* 2000, **143:**1–20.

Tisue, S, Wilensky U: **NetLogo: A simple environment for modeling complexity.** In *Proceedings of the Fifth International Conference on Complex Systems*. Boston, MA; 2004:16–21.

Trevorrow, A, Rokicki T, et al.: **Golly Game of Life simulator.** 2005. http://golly.sf.net/.

Wuensche, A: **Discrete dynamics lab (DDLab) – Software and Manual.** 1994. http://ddlab.com/.

Yamanoi, J, Sayama H: **Post-merger cultural integration from a social network perspective: A computational modeling approach.** *Comput Math Organ Theory* 2012. in press.

doi:10.1186/2194-3206-1-2
Cite this article as: Sayama: PyCX: a Python-based simulation code repository for complex systems education. *Complex Adaptive Systems Modeling* 2013 **1**:2.