SYSTEMIC RISK IN ARTIFICIAL WORLDS, USING A NEW TOOL IN THE ABM PERSPECTIVE

PIETRO TERNA

Department of Economics and Public Finance, University of Torino, Corso Unione Sovietica 218bis, 10134 Torino, Italy

terna@econ.unito.it

We propose SLAPP, or Swarm-Like Agent Protocol in Python, as a simplified application of the original Swarm protocol, choosing Python as a simultaneously simple and complete object-oriented framework. With SLAPP we develop two test models in the Agent-Based Models (ABM) perspective, building an artificial world related to the actual important issue of interbank payment and liquidity.

1 A few notes on agents and complexity

Following Ostrom (1988), and to some extent, Gilbert and Terna (2000), in social science, we traditionally build models as simplified representations of reality in two ways: (i) verbal argumentation and (ii) mathematical equations, typically with statistics and econometrics. The first way (i) is absolutely flexible and adaptable, as in the case of a historical book reporting an analysis of past events, but mere descriptions and discussion, by their nature, preclude tests and verifications of hypotheses. In contrast, the second way (ii) allows for computations and verifications, but suffers from severe limitations in flexibility and adaptation, especially with respect to how agents are expected to operate in the model and when accounting for their heterogeneity and interactions.

There is a third way to build models, (iii) computer simulation, mainly if agent-based. Computer simulation can combine the extreme flexibility of a computer code where we can create agents who act, make choices, and react to the choices of other agents and to modification of their environment – and its intrinsic computability. This allows us to use the descriptive capabilities of verbal argumentation and the ability to calculate the effects of different situations and hypotheses together. From this perspective, the computer program is a form of mathematics. In addition, we can generate time series from our models and analyze them employing statistics and econometrics.

However, reality is intrinsically agent-based, not equation-based (for a short, but illuminating discussion of this consideration, see Weinberg (2002) in his review of Wolfram's book, A New Kind of Science). At first glance, this is a strong criticism. Why reproduce social structures in an agent-based way, following (iii), when science applies (ii) to describe, explain, and forecast reality, which is, per se, too complicated to be understood?

The first reply is that we can, with agent-based models and simulation, produce artifacts of actual systems and "play" with them, i.e., showing consequences of perfectly known ex-ante hypotheses and agent behavioral designs and interactions; then we can apply statistics and econometrics to the outcomes of the simulation and compare the results with those obtained by applying the same tests to actual data. In this view, simulation models act as a sort of magnifying glass that may be used to better understand reality.

Considering the analysis of agent-based simulation model as a source of knowledge, there is another "third way view" of these kinds of tools. Consider Axelrod and Tesfatsion (2005):

> Simulation in general, and ABM in particular, is a third way of doing science in addition to deduction and induction. Scientists use deduction to derive theorems from assumptions, and induction to find patterns in empirical data. Simulation, like deduction, starts with a set of explicit assumptions. But unlike deduction, simulation does not prove theorems with generality. Instead, simulation generates data suitable for analysis by induction. Nevertheless, unlike typical induction, the simulated data come from a rigorously specified set of assumptions regarding an actual or proposed system of interest rather than direct measurements of the real world. Consequently, simulation differs from standard deduction and induction in both its implementation and its goals. Simulation permits increased understanding of systems through controlled computational experiments.

The considerations above act in a way similar to abduction, or inference to the best explanation, where one chooses the hypotheses that, if true, give the best explanation of the actual evidence. Note that in the ABM perspective, the hypotheses are also related to the rule that determines the behavior of the agent.

The second reply is that, relying on Anderson (1972), we know that complexity arises when agents or parts of a whole act and interact and the quantity of involved agent is relevant. Furthermore, following Villani (2006, p.51), "Complex systems are systems whose complete characterization involves more than one level of description." To manage complexity, one needs to build models of agents. As a stylized example, consider ants and an ant-hill: Two levels need to be studied simultaneously to understand the (emergent) dynamic of the ant-hill based on the (simple) behaviors of the ants.

We can also imagine building models based on multiple layers of agents [1], with the agents of each layer composing in a collective sense the more complicated agents of the upper stratum.

This interpretation of the agent-based paradigm, which is consistent with the way this kind of model is used in this work, corresponds to the "second use - partially soluble models: Artificial agents as complementary to mathematical theorizing" and to the "third use - models ostensibly intractable or provably insoluble: Agent computing as a substitute for analysis" considered in Axtell (2000). The Axtell's first use occurs "when models can be formulated and completely solved: Agent models as classical simulation."

[1] Being each layer a *swarm*, which is also the name of the first standardized tool used to build this kind of models, i.e., Swarm, from Santa Fe Institute (Minar et al., 1996)

The first use quoted above is mainly related to Monte Carlo simulations and the verification of numerical solutions to equation models. The second use relates to the cases of existing equilibria which can be incomputable, not attainable by bounded rational agents, known only for simple network configurations, or less interesting than transition phases, fluctuations, and extreme events. The third use is related to intractable models (my addendum to Axtell's considerations) when we believe that agents should be able to develop self-generated behavioral rules. This is the case here.

However, agent-based simulation models have severe weaknesses, primarily arising from:

(a) The difficulty of fully understand them without studying the program used to run the simulation;
(b) The necessity of carefully checking computer code to prevent generation of inaccurate results from mere coding errors. Epstein and Axtell (1994) pointed out, in their seminal paper, that it is necessary to develop new ways to control software and avoid bugs. In addition, due to the object-oriented structure that is intrinsic to agent-based programs, it is also possible to create a class of internal agents charged with observing the behavior of the actual agents of the simulation and reporting anomalies. Anomalies can be interesting to analyze and do not necessarily always arise from errors, but it is necessary to carefully explore this possibility. For example, if an accounting procedure produces strange results, the users search for an error in the procedure; if a simulation program produces anomalous results, the user may have discovered an interesting finding; however, in this case, it is also necessary to determine whether the finding is actually valid, and not the product of a coding error;
(c) The difficulty of systematically exploring the entire set of possible hypotheses in order to infer the best explanation, in accordance with the previously discussed practice of abductive reasoning. This is mainly due to the inclusion of behavioral rules for the agents within the hypotheses, which produces a space of possibilities that is difficult if not impossible to explore completely.

The difficulty of communicating the results, which is described in (a), can be overcome by the diffusion of standardized tools to develop agent simulation models and by the introduction of a protocol to be applied to those tools. The first example, introduced in the mid-1990s, is Swarm (www.swarm.org), a project that started within the Santa Fe Institute, but then grew independently. Swarm is not a program in the classic sense, but a library of functions to build agent-based computer models. More specifically, it is a library of particular functions that are useful in the handling of a collection of agents, populating spaces with agents, or organizing events in time. Swarm is appropriately a milestone in simulation, thanks to the protocol suggested for using those functions, initially combining them with codes written in Objective C (a language combining C and a subset of SmallTalk) and, subsequently, in Java. The

Swarm development team's original purpose, which was to create a *lingua franca* for agent-based model development, has only been partially achieved if one considers only the library of functions. With modern languages such as Python, a large part of the Swarm library is now unnecessary due to the facilities offered by the language itself. On the contrary, when considering the protocol aspect of the project, Swarm has been highly successful, being that protocol intrinsically the basis of several recent tools. For interesting considerations for the use of Python in agent-based programming, refer to Isaac (2008) and for an application of the Swarm protocol to Python, see SLAPP, which is introduced here.

Many other tools have been built upon the Swarm legacy, such as Repast, Ascape, JAS, and now SLAPP. Different protocols are used by important tools, such as NetLogo and StarLogo. StarLogo TNG, a recent innovative version of StarLogo, is programmed by moving small shaped cards which fit together to build a running code. A second important innovation of StarLogo TNG was the production of animations that are very similar to animations in video games. This was done because video games are typically easily understood.

We can deal with the second weakness introduced in (b), i.e., the risk of using code with "bugs" that corrupt the results, both when employing the standard tools reported here (but this is in some way insufficient) and duplicating the code using two independent tools programmed by two different scholars. The result is never the same, due mainly to the use of random numbers when determining sequences. However, if the emergent phenomena are substantially similar in both constructions, we can be reasonably sure that the results are not the product of coding errors. This significant amount of work is suggested mainly for important and critical applications.

The third weakness described in (c), i.e., the difficulty of exploring the whole set of possible hypotheses (including the behavioral rules of the agents, where the full rationality and perfect information hypotheses are only one of the possible choices and not plausible) is determined by the uncontrolled dimension of the space of possibilities. This space of possibilities, when analyzed in a detailed way, is necessary for computations where no black box is allowed, although it generates an unmanageable set of possible paths. This is precisely why this paper proposes the use of neural networks to generate behavioral rules in an atomatic way, or, in other words, the reinforcement of learning to extract the same rules from experience. In effect, this paper seeks to introduce strategies for going from the wide search of hypotheses about behavior to a procedure to calculate artificially generated, but plausible, rules.

Generating behavioral rules to achieve the capability of emulating cognition is a step that is both highly difficult and challenging. Consider Sun (2006, p. 17):

> What makes computational social simulation, especially computational cognitive social simulation (based on detailed models of cognitive agents),

> different from the long line of social theories and models (such as utility theory and game theory) includes the fact that it enables us to engage with observations and data more directly and test various factors more thoroughly. In analogy with the physical sciences (…), good social theories may be produced on the basis of matching theories with good observations and data. Cognitive agent based computational social simulation enables us to gather, organize, and interpret such observations and data with cognition in mind. Thus, it appears to be a solid means of developing social–cognitive theories.

Finally, with regard to the presentation of the results, the interaction between artificial agents and actual people, and the tools quoted above, it is useful to consider use of artificial on line worlds, such as Second Life (Bainbridge 2007).

The motivation for using ABM techniques with learning capabilities in this work is to explore and discover the consequences of self-generated behavioral schemes applied to an actual case of complex interaction (like the case of the diffusion of innovation and ideas in which many counterintuitive consequences of planned action occur in reality).

Let us conclude with Lave and March (1975, p.10): "The best way to learn about model building is to do it."

2 From a "classical" protocol to a new tool

2.1 *The Swarm protocol*

As seen above, the Swarm protocol is a "classical" reference in the relatively young world of the agent-based simulation, mainly for social sciences. The Swarm project, born at Santa Fe Institute, has been developed with an emphasis on three key points (Minar et al., 1996): (i) Swarm defines a structure for simulations, a framework within which models are built; (ii) the core commitment is to a discrete-event simulation of multiple agents using an object-oriented representation; (iii) to these basic choices Swarm adds the concept of the "swarm," a collection of agents with a schedule of activity.

The "swarm" proposal was the main innovation coming from the Swarm project, diffused as a library of function together with a protocol to use them. Building the (iii) item required a significant effort and time in code development by the Swarm team; now using Python we can attain the same result quite easily and quickly.

To approach the SWARM protocol via a clear and rigorous presentation it is possible refer to the original SimpleBug tutorial (Langton, 1996?), developed using the Objective-C programming tool (built on C and Smalltalk, www.smalltalk.org) by Chris Langton and the Swarm development team; the tutorial also has explanatory texts in the README files of the main folder and of the internal subfolders). The same contents have also been adapted by Staelin (2000), to the Java version of Swarm, and by myself (Terna, 2007), to create a Python implementation, exclusively related to the protocol and not to the libraries. Note that the Swarm original libraries are less important, anyway, using a modern object-oriented language. The SWARM protocol can be considered as a meta-*lingua franca* to be used in agent-based simulation models.

2.2 *The Swarm-Like Agent Protocol in Python (SLAPP)*

The SLAPP project[2] has the goal of offering to scholars interested in agent-based models a set of programming examples that can be easily understood in all its details and adapted to other applications.

Why Python? Quoting from its main web page: "Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days."

Python can be valuably used to build models with a transparent code; Python does not hide parts, have "magic" features nor have obscure libraries. Finally, we want to use the openness of Python to: (i) connect it to the R statistical system (R is at http://cran.r-project.org/; Python is connected to R via the rpy library, at http://rpy.sourceforge.net/); (ii) go from OpenOffice (Calc, Writer, …) to Python and vice versa (via the Python-UNO bridge, incorporated in OOo); (iii) do symbolic calculations in Python (via http://code.google.com/p/sympy/); and (iv) use Social Network Analysis from Python, with tools like the Igraph library (http://cneurocvs.rmki.kfki.hu/igraph/), the libsna library (http://www.libsna.org/), and the pySNA code (http://www.menslibera.com.tr/pysna/).

The main characteristics of the code reproducing the Swarm protocol in Python are introduced step by step via the on line tutorial referenced above. Summarizing:

- SimpleBug – We use a unique agent, running the time by the way of a *for* cycle, without object-oriented programming.
- SimpleClassBug - We run the time by the way of a *for* cycle, now with object-oriented programming to create and to use a unique agent as an instance of a class; in this way, it is quite simple to add other agents as instances of the same class.
- SimpleClassManyBugs - We run the time by the way of a *for* cycle, with object-oriented programming to create many agents as instances of a class; all the agents are included in a collection; we can interact with the collection as a whole.
- SimpleSwarmModelBugs - We run the time following a *schedule* of events based on a simulated clock; the schedule can be managed in a dynamic way

[2] Python is at www.python.org. You can obtain the SLAPP tutorial files and the related examples at: http://eco83.econ.unito.it/terna/slapp.

(events can change sequences of events). With object-oriented programming we create families of agents as instances of classes, within a special class, the model class, that can be considered as a the experimental layer of our program.

- SimpleSwarmObserverBugs – As above, but we now have the model and all its items (agents, schedule, clock) included in a top layer of the application, that we name "observer". The observer runs the model and uses a schedule to apply tools like graphical representations, report generations, etc. The clock of the observer can be different from that of the model, which allow us to watch the simulation results with a granularity different from that of the simulated events.

In the object-oriented programming perspective the starting module generates the observer as an instance of the observer class. The observer creates: (i) the reporting tools, (ii) one or more models as instances of the class model and (iii) finally the schedule coordinating the top-level actions. Each model creates (iv) the instances of the agents and (v) the schedule controlling their behavior.

3 Recreating an actual world in an artificial way: interbank payments and systemic risk

3.1 *The payment problem*

We shift to focus on the concrete aspects of an actual banking system, recreating the interaction of two institutions (a payment system and a market for short-term liquidity) to investigate interest rate dynamics in the presence of delays in interbank movements. [3]

The problem is a crucial one because delays in payments can generate liquidity shortages that, in the presence of unexpected negative operational or financial shocks, can produce huge domino effects (Arciero et al., 2009). Here, we use agent-based simulation as a magnifying glass to understand reality.

3.2 *Two parallel systems*

We have two parallel and highly connected institutions: the RTGS (Real Time Gross Settlement payment system) and the eMID (electronic Market of Interbank Deposit). Considering the flow of interbank payments settled via the first institution, we simulate delays in payments and examine the emergent interest rate dynamics in the

[3] I am deeply grateful to Claudia Biancotti and Leandro D'Aurizio, of the Economic and Financial Statistics Department of the Bank of Italy, and to Luca Arciero and Claudio Impenna, of the Payment System Oversight Office of the Bank of Italy, for having involved me in considering this important problem. The model is a contribution that I hope can be useful in identifying some aspects of the problem in a complexity perspective. You can obtain the code of the model by e-mailing the author.

eMID. In this kind of market the interest rate is the price. A few microstructures of the market should be investigated and understood.

In Figure 1 we present a modified representation of the standard sequence diagram of the UML (Unified Modeling Language, www.uml.org) formalism, introducing time as the first actor or user in the sequence. Events come from a time schedule to our simulated environment; the treasurers of the banks, acting via the RTGS system, with given probabilities bid prices, to buy liquidity in the eMID, or ask prices, to sell liquidity in the same market. Bid and ask probabilities can be different. The simple mechanism of bidding or asking on a probabilistic basis (if and only if a payment has to be done or has been received, as in Figure 1), will be integrated – in future developments - with an evaluation of the balance of the movements in a given time period.

The different sequences of the events (with their parallel or immediate diffusion, as in Figure 1) generate different lists of proposals into the double-action market we are studying. Proposals are reported in logs: the log of the bid proposals, according to decreasing prices (first listed: bid with the highest price); and the log of the ask proposals, according to increasing prices (first listed: ask with the lowest price). “Parallel” means that we are considering an actual situation in which all the treasurers are making the same choice at practically the same time.

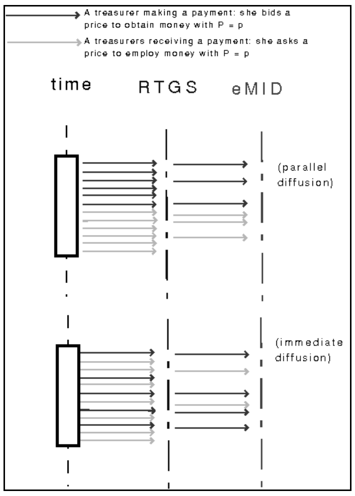
A treasurer making a payment: she bids a price to obtain money with P = p
A treasurers receiving a payment: she asks a price to employ money with P = p
time
RTGS
eMID
(parallel diffusion)
(immediate diffusion)

Figure 1. Events to RTGS and from RTGS to eMid, in two different ways: a “quasi UML” representation of a sequence diagram.

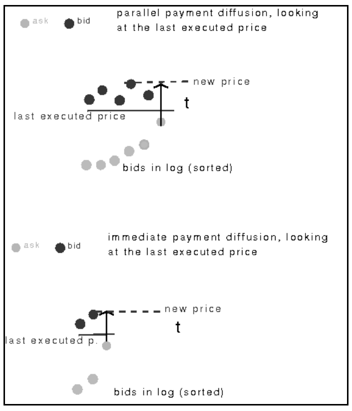
ask
bid
parallel payment diffusion, looking at the last executed price
new price
t
last executed price
bids in log (sorted)
ask
bid
immediate payment diffusion, looking at the last executed price
new price
t
last executed p.
bids in log (sorted)

Figure 2. Looking at the last executed price, both in a parallel and in an immediate diffusion scheme.

In Figure 2 we discuss how a new price is proposed to the market when we look at the last executed price as a reference point, placing a price below it to get a ask position easily matched. In this case, both the case of parallel proposal and that of immediate diffusion are figuring out close expected situations. On the other hand: (i) this is not the case in the logs of the unmatched proposals, with ask prices greater than bid prices; (ii) the behavior of a market maker, not present here, is based on positive ask minus bid price differences. Other potential microstructures have to be investigated.

In Figure 3, a new price is proposed to the market looking at the best proposal in the opposite log as a reference point, placing a price below it to get an ask position easily matched. The cases of parallel proposal and that of immediate diffusion are now producing quite different effects.

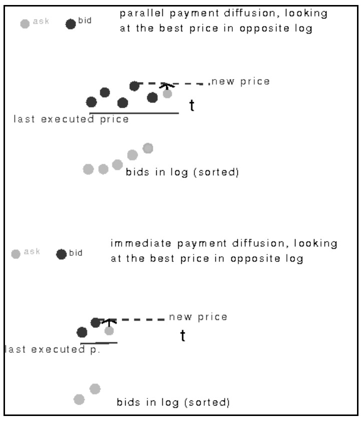
ask
bid
parallel payment diffusion, looking at the best price in opposite log
new price
t
last executed price
bids in log (sorted)
ask
bid
immediate payment diffusion, looking at the best price in opposite log
new price
t
last executed p.
bids in log (sorted)

Figure 3. Looking at the best proposal in the opposite market log, both in a parallel and in an immediate diffusion scheme..

3.3 *A case of simulated behavior*

We show here an interesting case of the dynamics emerging from this simulation environment, that occurs when the diffusion of the payment into the RTGS system is parallel and the operators look at the last executed price in the eMID. The run reported in Figure 4 shows a non-trivial behavior of the interest rate. The dynamic is here magnified due to the dimension chosen for micro-movement in bids and asks. In these five days, we have a huge movement of this time series, as a consequence of significant delays in interbank payments. The simulation runs step by step, but we account for breaks in the time to reproduce the end of each day (i.e., cleaning all the positions, etc.).

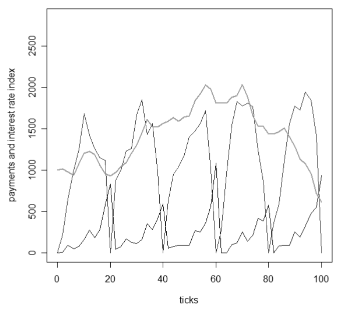

Figure 4. Interest price dynamic (upper line), stock of due payments (intermediate line) and flow of the received payments (lower line) in case of relevant delay in payments (with a uniform random distribution between 0 and the 90% of the available time until the time break). Time breaks at 20, 40, … (end of a day).

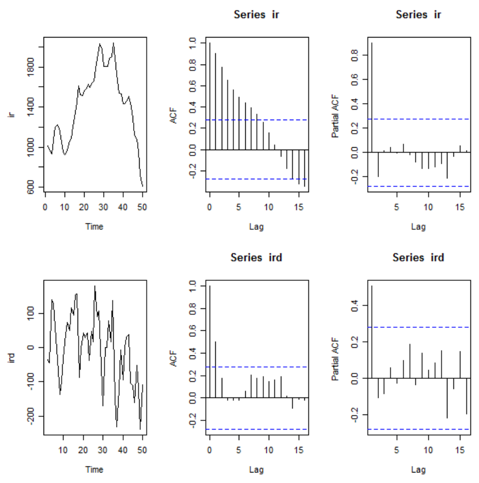

Figure 5. The autocorrelation analysis of the interest rate data of 4 (with the presence of delays in interbank payments). First row: raw data; lagged correlations among data; the same, but as partial correlations. Second row: data first differences with lag 1; their lagged correlations; their lagged partial correlations.

Elaborating the interest rate series with the standard AR (autoregressive) and MA (moving average) technique, directly connecting SLAPP to R as seen above, we find in the graphs of the second row in Figure 5 a typical AR(1) model.

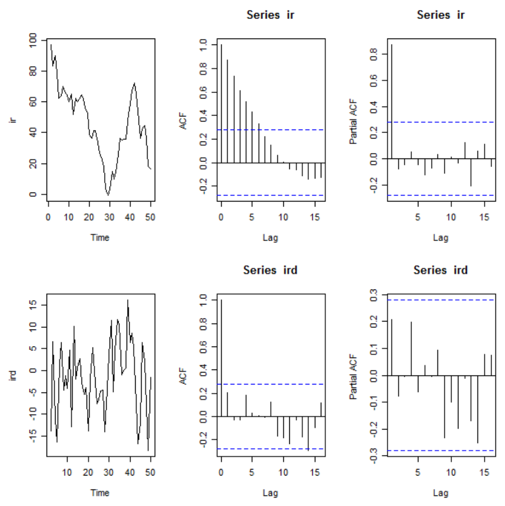

Figure 6. The autocorrelation analysis of the interest rate data in a case of absence of delays in interbank payments). First row: raw data; lagged correlations among data; the same, but as partial correlations. Second row: data first differences with lag 1; their lagged correlations; their lagged partial correlations.

On the contrary, in a situation like that of Figure 6, with data coming from a simulation run in which no payment delays occur, we find a random-walk dynamic in the interest rate first differences (first graph of the second row), without any correlation evidence.

This analysis suggests that the strong difference between these two situations is the direct consequence of the presence/absence of the payment delays.

Future developments

There are two promising lines for future developments:

- In terms of SLAPP, the development of the capability of directly probing each agent, the graphical representation of spatial dynamics and of social networks links, and the simplification of the schedule code for event dynamic.
- In terms of the payment system, applying also in this case the reinforcement learning technique, the introduction of a *market maker*, i.e., a subject

continuously asking and bidding a price for selling and buying money, with a spread, assuring liquidity to the market and developing a pricing capability aimed at the containment of liquidity crisis.

References

Anderson, P. W. (1972), More is different. *Science*, 177, 4047, pp. 393–396.

Arciero L., Biancotti C., D'Aurizio L. and Impenna C. (2009), Exploring agent-based methods for the analysis of payment systems: a crisis model for StarLogo TNG. *Journal of Artificial Societies and Social Simulation*, http://jasss.soc.surrey.ac.uk/12/1/2.html.

Axelrod R. and Tesfatsion L. (2005), *A guide for newcomers to agent-based modeling in the social sciences*. In K. L. Judd and L. Tesfatsion (eds.), *Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics*, pp. 1647-1658, Amsterdam, North-Holland. On line at http://www.econ.iastate.edu/tesfatsi/GuidetoABM.pdf

Axtell R. (2000), *Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences*. In Proceedings of the Workshop on Agent Simulation: Applications, Models and Tools, Chicago, Argonne National Laboratory. On line at http://www.brookings.edu/~/media/Files/rc/reports/2000/11technology_axtell/agents.pdf

Bainbridge W.S. (2007), The Scientific Research Potential of Virtual Worlds. *Science*, 317, pp. 472-476.

Gilbert N. and Terna P. (2000), How to build and use agent-based models in social science, *Mind & Society*, 1, 1, pp. 57-72.

Isaac A. G. (2008), Simulating Evolutionary Games: A Python-Based Introduction. *Journal of Artificial Societies and Social Simulation*, 11, 3. http://jasss.soc.surrey.ac.uk/11/3/8.html.

Langton C. and Swarm development team (1996?), Santa Fe Institute, *SimpleBug tutorial*, on line at http://ftp.swarm.org/pub/swarm/apps/objc/sdg/swarmapps-objc-2.2-3.tar.gz.

Lave C. A. and March J. G. (1975), *An introduction to models in the social sciences*. New York, Harper & Row.

Minar N., Burkhart R., Langton C. and Askenazi M. (1996), *The Swarm simulation system: A toolkit for building multi-agent simulations*. Working Paper 96-06-042, Santa Fe Institute, http://www.swarm.org/images/b/bb/MinarEtAl96.pdf

Ostrom, T. (1988), Computer simulation: the third symbol system. *Journal of Experimental Social Psychology*, n. 24, pp. 381-392.

C. J. Staelin (2000), *jSIMPLEBUG, a Swarm tutorial for Java*, (2000), at http://www.cse.nd.edu/courses/cse498j/www/Resources/jsimplebug11.pdf, only text, or http://eco83.econ.unito.it/swarm/materiale/jtutorial/JavaTutorial.zip, text and code.

Sun R. (ed.) (2006), *Prolegomena to Integrating Cognitive Modeling and Social Simulation*. In R. Sun, *Cognition and Multi-Agent Interaction - From Cognitive Modeling to Social Simulation*, Cambridge, Cambridge University Press.

Sutton R. S., Barto A. G. (1998), *Reinforcement Learning: An Introduction*. Cambridge MA, MIT Press.

Terna P. (2007), implementation of the SWARM protocol using Python, at http://eco83.econ.unito.it/terna/slapp.

Villani M. (2006), Networks and Complex Systems. In M. Villani (ed.) (2006), *Educating managers in complexity*, pp. 41-119, Roma, Aracne.

Weinberg S. (2000), Is the Universe a Computer? *The New York Review of Books*, 49, 16. http://www.nybooks.com/articles/15762.