

IMAGINARY OR ACTUAL ARTIFICIAL WORLDS USING A NEW TOOL IN THE ABM PERSPECTIVE

PIETRO TERNA

*Department of Economic and financial sciences, 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 both an artificial world related to an imaginary situation with stylized chameleons and an artificial world related to the actual important issue of interbank payment and liquidity.

1 From a “classical” protocol to a new tool

1.1. *The Swarm protocol*

The Swarm protocol dates from the mid-1990s, so in some way it is a “classical” reference in the relatively young world of the agent-based simulation, mainly for social sciences. The Swarm project (www.swarm.org), born at Santa Fe Institute, has been developed with an emphasis on three key points [1]: (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 [2], 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 [3], to the Java version of Swarm, and by myself [4], 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.

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

The SLAPP project ^a 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://cneurocv.s.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 run 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 (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

^a 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>.

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.

2 Creating an imaginary artificial world: the surprising behavior of learning chameleons

2.1 *The structure of the chameleon world*

A test model created with SLAPP involves intelligent chameleons. We have chameleons of three colors: red, green and blue. When two chameleons of different colors meet, they both change their color, assuming the third one. If all chameleons change to be the same color, we have a steady-state situation. This case is possible, although rare. Even so, what if the chameleons of a given color want to conserve their identity? On the other hand, what if they strongly want to change it?

With the on-line version of the chameleon model,^b we can see the chameleons moving randomly in their space. The chameleons can (i) move in a random way or (ii) refer to a *runner* mind (nine neural networks able to evaluate the moves from the point of view of the runners, used together) to avoid changing their color, or (iii) refer to a *chaser* mind (nine neural networks able to evaluate the moves from the point of view of the chasers) to look for contacts and so to change their color, if they want to change their color. As an example, if we tell a specific type of chameleons (i.e., the red ones) to be conservative, adopting the rules generated by a reinforcement learning process to avoid contacts, they become capable of increasing in number with the strategy of decreasing their movement when staying in zones free from chameleons of other colors, and getting close to subjects with their own color.

^b The project, built in SLAPP, has been implemented also in NetLogo, relatively to the on-line visual representation of the results: with NetLogo you can easily produce an applet to be directly run in a browser; NetLogo is at <http://ccl.northwestern.edu/netlogo/>. The model is at <http://eco83.econ.unito.it/terna/chameleons/chameleons.html>, where you can find also an animation with voice explanations.

I thank Riccardo Taormina, a former student of mine, for developing this kind of application with great involvement and creativity. Many thanks also to Marco Lamieri, a former Ph.D. student of mine, for introducing the powerful chameleon idea.

2.2 Reinforcement learning

We use reinforcement learning [5] to develop intelligent behavior in our chameleons. Rewards and punishments come from the experience made in the past by chameleons while acting.

The evaluation of the rewards is quite simple. We consider here only a 3×3 space, as in Figure 1, with nine potential moves, accounting also for staying in the initial patch. The red chameleon in the central position of the internal 3×3 square in Figure 2 has three enemies around it; moving to the north in the center of the green square, it would have only two (visible) enemies, with a +1 reward; the reward can also be negative. Here we always consider two steps jointly and sum up their rewards, without the use of a discount rate. The double-step analysis compensates for the highly bounded rationality strategy applied both to the knowledge of the world (limited in each step to a 5×5 space) and the evaluation of the rewards, in a 3×3 area.

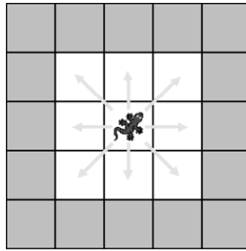


Figure 1. The nine possible moves of the agent (staying in place and moving to one of the eight adjacent squares).

We have nine neural networks, one for each of the potential moves in the 3×3 space, with 25 input elements (the 5×5 space), ten hidden elements and a unique output, which is a guess of the reward that can be gained by choosing each one of the nine valid moves in the presence of each specific situation in the 5×5 space. Neural networks are used to summarize the results of the described trial and error process, avoiding the requirement of dealing with huge tables reporting the data upon the reward of each potential choice in each potential situation.

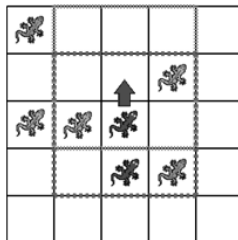


Figure 2. Two subsequent spatial situations, always considered from the center of the two internal 3×3 squares.

The dimension of the observed space, 5x5, obeys the bounded rationality principle. We could also consider a 7x7 space, but the number of potential situations increases enormously: in the case of the 5x5 space, omitting the central square, we have 2^{24} potential situations. With a 7x7 space, we have 2^{48} possibilities, a number on the order of 3 times 10^{14} . In the 5x5 case, we are able to explore a simplified space like that of Figure 3, with about 17 million of possible states, with symmetric inputs not mandatory to produce symmetric outcomes (the decision system attributed to our agents is intrinsically imperfect).

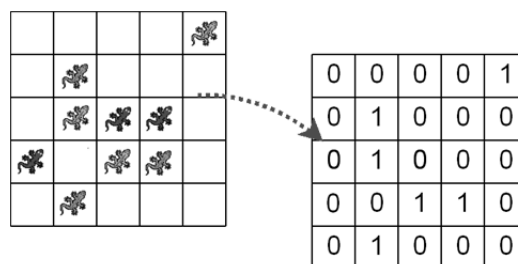


Figure 3. The chameleon in the center of the grid (a red one) creates the matrix corresponding to its observed world. Values of 1 identify the places of the chameleons of colors other than red.

2.3 Results in chameleon behavior

The chameleons can (i) move in a random way, or (ii) refer to a *runner* mind (the nine neural networks able to evaluate the moves from the point of view of the runners, used together) both to avoid contacts and to avoid changing their color, or (iii) refer to a *chaser* mind (the nine neural networks able to evaluate the moves from the point of view of the chasers, used together) to look for contacts and to change their color.

The running model can be found at the address reported in the note introducing the section, with and animation with voice instruction to train users in interacting with the simulation interface. It is possible to reproduce the different behavior of both the running and of the chasing chameleons, remembering that the actions of the chameleons are arising from an automatic learning process.

The model can also be metaphorically interpreted in the following way: an agent diffusing innovation (or political ideas or financial innovative behaviors) can change itself through interaction with other agents. As an example, think about an academic scholar working in a completely isolated context or, on the contrary, interacting with other scholars or with private entrepreneurs to apply the results of her work. On the opposite side, an agent diffusing epidemics modifies the others without changing itself.

3 Recreating an actual world in an artificial way: interbank payments

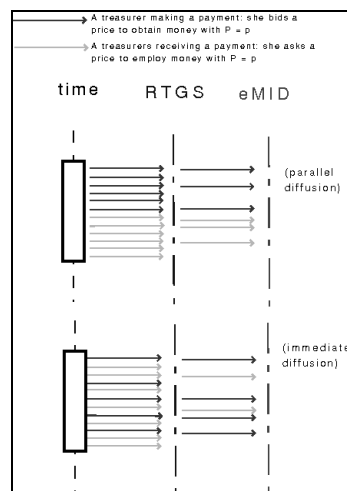
3.1 The payment problem

From the imaginary world of the chameleons, always using SLAPP, 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.^c

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



^c 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.

Figure 4. Events to RTGS and from RTGS to eMid, in two different ways: a “quasi UML” representation of a sequence diagram.

In Figure 4 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 4), will be integrated – in future developments - with an evaluation of the balance of the movements in a given time period.

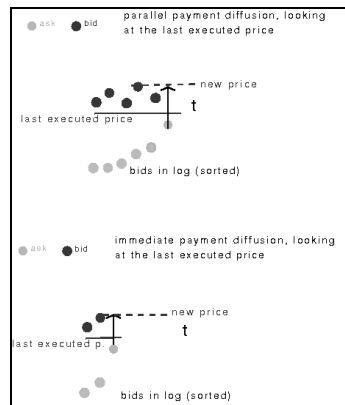


Figure 5. Looking at the last executed price, both in a parallel and in an immediate diffusion scheme.

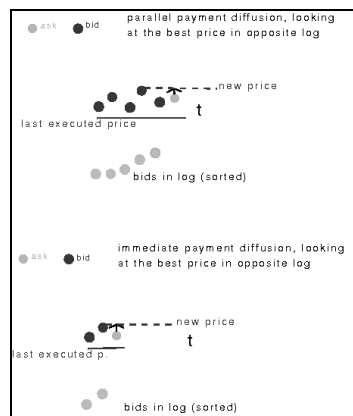


Figure 6. Looking at the best proposal in the opposite market log, both in a parallel and in an immediate diffusion scheme..

The different sequences of the events (with their parallel or immediate diffusion, as in Figure 4) 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.

In Figure 5 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 an 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 6, 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.

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 7 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.).

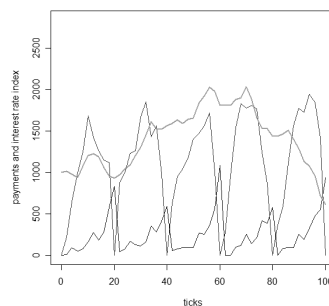


Figure 7. Interest price dynamic (upper line), stock of due payments (intermediate line) and flow of the received payments (lower line) in case of relevant delay in payments (with a uniform random distribution between 0 and the 90% of the available time until the time break). Time breaks at 20, 40, ... (end of a day).

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 8 a typical AR(1) model. On the contrary, in a situation like that of Figure 9, 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.

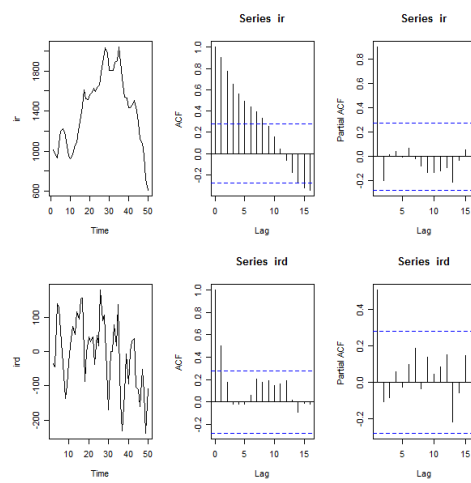


Figure 8. The autocorrelation analysis of the interest rate data of Figure 7 (with the presence of delays in interbank payments). First row: raw data; lagged correlations among data; the same, but as partial correlations. Second row: data first differences with lag 1; their lagged correlations; their lagged partial correlations.

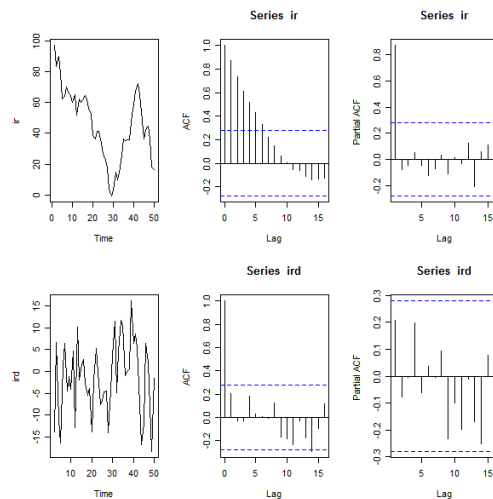


Figure 9. The autocorrelation analysis of the interest rate data in a case of absence of delays in interbank payments). First row: raw data; lagged correlations among data; the same, but as partial correlations. Second row: data first differences with lag 1; their lagged correlations; their lagged partial correlations.

4 Future developments

There are three 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 chameleons, the introduction of communication capabilities, mainly via information left in the space, to search for the emergence of social coordination and of social capital.
- 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

1. N. Minar, R. Burkhart, C. Langton, and M. Askenazi, *The Swarm simulation system: A toolkit for building multi-agent simulations*. WP 96-06-042, Santa Fe Institute, Santa Fe (1996), <http://www.swarm.org/images/b/bb/MinarEtAl96.pdf>.
2. C. Langton and Swarm development team, Santa Fe Institute, *SimpleBug tutorial*, on line at <http://ftp.swarm.org/pub/swarm/apps/objc/sdg/swarmapps-objc-2.2-3.tar.gz>, (1996?).
3. C. J. Staelin, *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 (2000).
4. P. Terna, implementation of the SWARM protocol using Python, at <http://eco83.econ.unito.it/terna/slapp>, (2007).
5. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge MA, MIT Press (1998).
6. L. Arciero, C. Biancotti, L. D'Aurizio and C. Impenna, Exploring agent-based methods for the analysis of payment systems: a crisis model for StarLogo TNG. *Journal of Artificial Societies and Social Simulation*, forthcoming, (2008).