Replication of Sugarscape Using MASON[1]

Anthony Bigbee, Claudio Cioffi-Revilla, Sean Luke

Center for Social Complexity and Evolutionary Computation Laboratory
George Mason University, Fairfax, VA 22030 USA ccioffi@gmu.edu

1 Introduction

The purpose of this research was to replicate the Sugarscape model (Eptstein and Axtell 1996) and simulation outcomes as described in *Growing Artificial Societies* (*GAS*). Sugarscape is a classic agent-based model and contemporary simulation toolkits usually only have a very simple replication of a few core rules. There is scant evidence of significant replication of the rules and simulation outcomes; code supplied with Repast, Swarm, and NetLogo implement a minority of the rules in Sugarscape. In particular, the standard Repast distribution only implements **Growback**, **Movement**, and **Replacement**. Sugarscape implementations in these toolkits are clearly provided only as basic demonstrations of how well-known social models might be implemented, rather than complete achievements of scientific replication.

A major goal included assessing the maturity of the new MASON toolkit to replicate Sugarscape. MASON (Multiagent Simulator of Neighborhoods) "is a fast discrete-event multiagent simulation library core in Java, designed to be the foundation for large custom-purpose Java simulations, and also to provide more than enough functionality for many light-

---

[1] Presented at The 4th International Workshop on Agent-based Approaches in Economic and Social Complex Systems (AESCS 2005), Pacific-Asian Association for Agent-based Approach in Social Sciences (PAAAA), Tokyo Institute of Technology, Tokyo, July 9–13, 2005. The authors thank Joshua Epstein and Robert Axtell for comments on Bigbee (2005).

weight simulation needs." (Luke et al. 2005). Since MASON was designed to be a tool for social science research, among other uses, replication of one of the most recognized agent-based social science models would demonstrate its maturity and usability for its intended purpose.

Replication of well-known models is also important given the relative novelty of agent-based modeling in social science. Better tools and technique for lowering barriers to entry by social scientists are desirable outcomes.

2 Approach

Epstein and Axtell (1996) offer a framework – Sugarscape – for agent-based modeling and simulation that revolves around the following elements: agents, environment, rules. Epstein and Axtell state that the defining feature of the Sugarscape/artificial society model is that "fundamental social structures and group behaviors emerge from the interaction of individual agents operating on artificial environments under rules that place only bounded demands on each agent's information and computational capacity." Computationally, Sugarscape rests on an 'object-oriented' approach consisting of:

Instance variables representing agents' internal states or attributes (such as sex, age, wealth);
Methods for agents' rules of behavior (such as eating, trading, combat);
Encapsulation of agents internal states and rules to facilitate agent-based model construction.

Details regarding object-oriented (OO) techniques in Sugarscape are generally omitted from *GAS*. Appendix A contains a short section on OO techniques used and considered. Polymorphism is not discussed and inheritance was considered but was not used due to "efficiency considerations .... In total, each agent has over 100 methods." By comparison, the single agent class in MASON Sugarscape has approximately 32 methods, although only 75-80% of all Sugarscape rules were implemented. A prototype implementation of Sugarscape using ASCAPE appears to employ polymorphism and inheritance.

2.1 MASON

Although the MASON distribution includes a variety of graphics, utility, and other supporting infrastructure and examples, discussion of MASON in this paper focuses on timing and scheduling. Controlling and determining what behaviors are executed and when are critical aspects of simulation. In addition to overt requirements – such as an agent having to scan its surroundings and move to a site before it can harvest resources – synchronous and asynchronous interactions occur between entities. A necessary, but not sufficient, requirement is that actions are executed in time exactly as desired—this requires a precise scheduling mechanism.

2.1.1 Scheduling and Time

The MASON Version 10 Schedule class sequences and executes objects that implement the `Steppable` interface. Classes implementing `Steppable` have a `Step` method for the schedule to initiate behavior/execution. A primary construct in `Schedule` is order, a collection of executable objects that enable a deterministic sequence of execution. For each time step, all order zero objects are executed first, followed by order one, and so forth. Implementation of rules in specific orders is a key aspect of this replication and orders for agent and environment rules are currently specified in the Sugarscape class as constants. These constants could easily be redesigned as parameters in the primary configuration file for more flexible experimentation.

`Seasons`, in addition to statistics, charts and logging, uses another MASON class – `MultiStep` to enabling `Steppables` – to be executed less than once every time step but with a regular periodicity. `Schedule.setRepeating` method that has an interval has better performance than `Multistep` when orders have multiple `Steppables`

Within each order, the `Steppable` entities are executed once in a random sequence, and the sequence is randomized every time step. The other critical timing mechanism used in this implementation is that agent instances themselves do not determine which of their rules are executed, nor in what sequence. Instead, a member instance named `RulesSequence` wrap `Sequence`, a MASON class for holding a static sequence of `Steppables`. This static sequence contains a collection of rules, each of which is invoked in turn. A benefit of using `Sequence` is that the `step()` method in the agents or environment objects are small and simple, primarily calling `step()` for its `RuleSequence`. The `RuleSequence` instance, in turn, calls each rule in the original order specified

when the rules were added during initialization. `Sequence` only has order *m* rules time complexity as opposed to order *n* entities x *m* rules complexity in direct schedule usage .

To provide for a flexible experimentation, a primary configuration file specifies rule execution order during runtime. Using *reflection* facilities of the Java language, initialization code translates text names for rules classes into object instantiations. This allows quick way to specify desired rule sequences without source code recompilation.

Finally, there are rules that involve cellular automata (CA) type synchronous treatment of all instances of a class (i.e. all entities of one type). **Pollution Diffusion (D)** is an environment rule in which the states of all sites synchronously update without using MASON scheduling machinery.

2.2 Space-time Interactions

Many agent-based models have discrete space aspects that UML sequence diagrams cannot visualize. As an example, interactions in space and time are illustrated in Figure 1. This diagram provides insight into the local effects of agent harvesting when a) sites grow back during the harvesting time step, and b) sites are restricted from growing back until one full time step after the time step during which harvest occurred. The blue circle depicts the location of the agent at each time step. Bigbee (2005) has an extended discussion on this phenomenon and emergent behavior.

2.3 Rules Implemented

Table 1 describes rules implemented from the classic model and whether the rule can be added, removed, or reordered simply by editing the `agent_rules_sequence` or `environment_rules_sequence` lines in the runtime configuration file.

Total source lines of code are approximately 3500 as counted by the SLOCCOUNT tool (Wheeler 2005). Total source lines of code for the rules is 934, with the ratio of rules code to other code being approximately 1:3. The non-rules source lines of code involve model initializing, graphs and statistics, logging, and parameter sweeping. The total SLOC count for MASON version 8 itself, not including the supplied demonstration applications, is approximately 18000. This includes many classes not used by MASON Sugarscape, including other portrayals and 3-D visualization infrastructure.

3 Results

Table 2 documents simulation outcomes and rule sets investigated as part of replication. Qualitative and other criteria are described in the table, such as whether aggregate statistics and shapes of graphs were matched, as well as an overall level of replication achieved—exact, general, or partial. The overall pattern of replication outcomes is that the outcomes documented in *GAS* were generally replicated in MASON Sugarscape simulations. The most successful replications occurred for outcomes that did not involve movement/welfare-dependent agent survival; these outcomes included **Culture**, **Pollution Diffusion**, **Seasons**, and other spatial phenomena. Bigbee (2005) provides detailed discussion of each outcome and issues in replicating.

4 Summary

The research yielded partial replication of the Sugarscape outcomes described in GAS. A major lesson from this research is the difficulty in understand and constructing simulation models that appear simple yet have complex emergent behavior.

Software engineering is a young field, although tools and techniques have emerged to support error/bug reduction, automated testing, and faster development. While OO-based software has been touted as an effective way of constructing software applications, OO methodologies do not eliminate cognitive error nor complexity in development. Close examination of the psychology of software development is beyond the scope of this thesis, but attention to the biases and heuristics literature and cognitive errors literature might yield some techniques and suggest tools appropriate for effective development of agent-based models.

Bigbee (2005) offers a list of lessons learned and 10 heuristics and ideas to promote successful replication. Social scientists creating or replicating agent-based models face a variety of challenges encountered in other fields and much work remains to be done to lower barriers to good science in this field.

References

Bigbee A (2005) Replication of Sugarscape Using MASON. Unpublished master's thesis, George Mason University, Fairfax, VA.

Densmore O (2005) Sugarscape. Retrieved April 1, 2005, from http://backspaces.net/Models/sugarscape.html

Doran J (2000) Questions in the Methodology of Artificial Societies. In. Suleiman R, Troitzsch K, Gilbert N (eds), *Tools and Techniques for Social Science Simulation*. : Physica-Verlag, Heidelberg

Epstein J, Axtell R (1996) *Growing Artificial Societies: Social Science from the Bottom Up*. Brookings Institution Press, Washington, D.C.

Hegselmann, R, Flache A (1998, June) Understanding Complex Social Dynamics: A Plea for Cellular Automata Based Modeling. *J Artificial Societies and Social Simulation* 3, Retrieved August 1, 2004, from http://www.soc.surrey.ac.uk/JASSS/1/3/1.html

Huberman B, Hogg T (1988) The Behavior of Computational Ecologies. In B. Huberman (Ed.), *The Ecology of Computation*. Amsterdam: North-Holland.

Kleiber C, Kotz S (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley, Hoboken, NJ

Kliemt H (1996) Simulation and Rational Practice. In R. Hegselmann, U. Mueller (Eds.), *Modelling and simulation in the social sciences from a philosophy of science point of view*. Kluwer, Dordrecht

Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G (2005) MASON: A Multi-Agent Simulation Environment. *Simulation* 81: 517-527

Nowak A, Lewenstein M (1996) Modeling social change with cellular automata. In R. Hegselmann, U. Mueller (Eds.), *Modelling and simulation in the social sciences from a philosophy of science point of view*. Kluwer, Dordrecht

Wheeler D SLOCCount. Retrieved April 1, 2005, from http://www.dwheeler.com/sloccount/

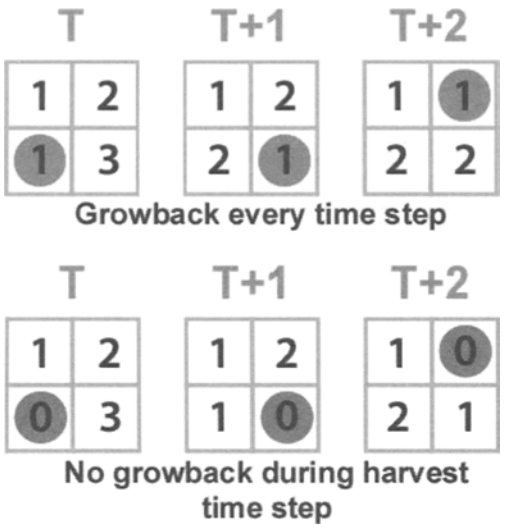

Fig. 1. Space time interactions under two Growback rules

Table 1. Implemented Rules

Symbol	Name	SLOC
G_{α}	Sugarscape growback	42
M	Agent movement	323
$R_{[a,b]}$	Agent replacement	17
$S_{\alpha\beta\gamma}$	Seasonal growback	16
$P_{\pi,x}$	Pollution formation	18
D_{α}	Pollution diffusion	56
S	Agent mating	163
None	Agent cultural transmission	
None	Group membership	128
K	Agent culture	
T	Agent trade	177

Table 2. Replication Outcomes

Outcome	Rule Set	Replication Criteria	Replication Achieved
Animation II-2 (p.29)	({G_1}, {M})	Hiving, peak clustering, terrace sticking	Exact
Figure II-5 (p. 31)	({G_1}, {M})	Small positive slopes, equally spaced lines, visually estimated line coordinates	General
Animation II-3 (p. 34)	({G_1},{M,$R_{[60,100]}$})	Pareto distribution, maximum wealth bin	General
Animation II-4 (p. 38)	({G_1},{M,$R_{[60,100]}$})	Gini coefficient evolution	General
Animation II-6 (p. 43)	({G_1}, {M})	Visual wave phenomenon	General
Animation II-7 (p. 46)	({$_{S1,8,50}$),{M})	Seasonal clustering	Exact
Animation II-8 (p. 49)	({G_1,D_1}, {M,P_{11}})	Migration patterns	Partial
Figure III-1 (p. 58)	({G_1}, {M,S})	Stable time series	General
Animation III-1 (p. 58)	({G_1}, {M,S})	Approximate stationary age distribution	General
Figure III-2 (p. 63)	({G_1}, {M,S})	Diverging Vision, Metabolism	General
Figure III-3 (p. 64)	({G_1}, {M,S})	Small amplitude oscillations	General
Figure III-4 (p. 65)	({G_1}, {M,S})	Large amplitude oscillations	Partial
Figure III-5 (p. 66)	({G_1}, {M,S})	Severe population swings, extinction	General
Animation III-6 (p. 75)	({G_1}, {M,K})	Homogenous population	Exact
Figure III-8 (p. 77)	({G_1}, {M,K})	Time series extremes, random group convergence	Exact
Animation IV-1 (p. 100)	({G_1}, {M})	Peak hopping, small population	General
Figure IV-4 (p. 110)	({G_1}, {M,T})	Significant trade volumes over time	Partial