REPLICATION OF SUGARSCAPE USING MASON

by

Anthony J. Bigbee
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Arts
Interdisciplinary Studies

Committee:

_____ Director

_____

_____

_____

_____ Department Chairperson

_____ Dean, College of
Arts and Sciences

Date: _____ Spring Semester 2005
George Mason University
Fairfax, VA

Replication of Sugarscape Using MASON

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Arts at George Mason University

By

Anthony J. Bigbee
Bachelor of Science
United States Naval Academy, 1989

Director: Claudio Cioffi-Revilla, Professor
Center for Social Complexity

Spring Semester 2005
George Mason University
Fairfax, VA

DEDICATION

This thesis is dedicated to my wife, Lee. It would have been impossible to finish this research without her intellectual and practical contributions and support.

ACKNOWLEDGEMENTS

I wish to thank my entire family for their encouragement. I am also grateful for the deep participation and knowledge of my committee. I thank Professor Claudio Cioffi-Revilla for his willingness to direct this research, constant encouragement, ability to keep it on track, and desire to share his keen insight into the social science world. I thank Professor Sean Luke for providing powerful and immediate feedback on the optimal use of MASON at all hours as well as other thought-provoking discussions. To Professor Ann Palkovich I am grateful for sage advice that steered me down this path several years ago. I thank Dr. Joshua Epstein for being generous with his time and his encouragement during moments when the road to completion appeared closed. Cindy Roberts of the Center of Social Complexity provided outstanding administrative support and saved me time at crucial points during the research. Finally, I thank Audrey Kelaher for her proactive monitoring of my progress and unerring support during my participation in the MAIS the program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

REPLICATION OF SUGARSCAPE USING MASON

Anthony J. Bigbee, M.A.

George Mason University, 2005

Thesis Director: Dr. Claudio Cioffi-Revilla

This thesis describes an effort to replicate Sugarscape, a classic social sciences model, using MASON, a new agent-based modeling and simulation toolkit. Agent-based modeling is a new method in the social sciences and there are few published replications of classic models. Sugarscape simulation outcomes documented in Growing Artificial Societies (Epstein and Axtell, 1996) were replicated, although significant differences were identified. These differences were due to either implementation errors or missing information in the original description of Sugarscape. MASON itself proved to be mature enough to implement Sugarscape, providing robust infrastructure for core needs such as scheduling. As part of this research, MASON was extended to provide automated parameter sweeping capabilities and automated data structure logging. In the process of replicating a simple, but not simplistic model, many barriers and practical challenges arose. As a result, this thesis provides a list of potential heuristics, techniques, and ideas for new tools to help prevent researcher errors and enable researchers to more deeply understand cause and effect in their computational creations.

1. Introduction and Motivation

In the last two decades, social scientists have begun using computational agent-based modeling techniques and advocating them as powerful, scientific tools. Proponents of agent-based model investigation argue that it can advance scientific knowledge and that it provides a variety of advantages over other kinds of scientific tools (Gilbert and Troitzsch, 1999). These advantages include:

- "[I]mposing precision on the theory, testing theories for contradictions…[but] the main advantage of computer simulations in the social domain are connected to the insights they can offer regarding emergence and dynamics." (Nowak and Lewenstein 1996, p. 256)
- Computer simulations of complex social phenomena, or complex adaptive systems, as potential flight simulators for policy exploration (Holland, 1995).
- Visual presentations of simulation dynamics and results, including visualization of social behavior otherwise difficult or impossible to observe. "It is important to stress the importance of visualization…Often just visual inspection may be sufficient to study emergent phenomena such as patterns being formed during the system's evolution." (Nowak and Lewenstein 1996, p. 257)
- Better understanding of macro effects in models as they relate to unintended consequences in social action (Hegselmann and Flache, 1998, p. 20).
- Unparalleled ability to render social objects and agents – social entities in general – with procedural code, in ways that are notoriously intractable using classical mathematical methods.

While the core idea of agent – or individual-based models – is at least several decades old (Schelling, 1978) – the emergence of Sugarscape as a way to investigate complex social phenomena using computer-based simulations signaled a new wave of research – "We view artificial societies as *laboratories*, where we attempt to 'grow' certain social structures in the computer – or *in silico* – the aim being to discover fundamental local or micro mechanisms that are sufficient to *generate* the macroscopic social structures and collective behaviors of interest." (Epstein and Axtell, 1996, p. 4)

Choosing to use agent-based modeling, however, does not eliminate challenging aspects of computational modeling, such as avoiding mistaking computational artifacts for valid outcomes. Nor does it mean that computationally valid models automatically prove or provide insight into real world social phenomena. It is for these reasons, and that agent-based modeling in the social sciences is a young discipline, that replication of major models has value in advancing scientific understanding. Creativity and novelty that generate projects like Sugarscape are part of scientific progress, but demonstration that these models can be replicated would provide a greater focus on the domain constructs being examined rather than the computational facets. Replication may also

afford different ways of looking at the original model and expose the model to a wider audience. Overall, replication is a fundamental part of science.

Finally, as new modeling and simulation toolkits emerge, they need to be validated themselves and to evaluated to understand how the support both computational modeling needs and simulation and analysis needs.

Goals and Purpose

The purpose of this research was to attempt to replicate the Sugarscape model and simulation outcomes as described in *Growing Artificial Societies* (*GAS*).

Sugarscape is regarded as a classic agent-based model and contemporary simulation toolkits usually only have a very simple replication of a few core rules and 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 replicability.

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 lightweight 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 newness of software engineering and the newness of agent-based modeling in social science. Better tools and technique for lowering the barriers to entry by social scientists are desirable outcomes.

Main Results

General replication of the Sugarscape model was achieved, although significant discrepancies remain unresolved, particularly for population-size based outcomes. It is undetermined if these discrepancies are a result of incorrect implementation, missing elements in the original parameter descriptions, or other factors. MASON was mature enough and provided core capabilities sufficient to generally replicate Sugarscape; it was extended in a few ways to enhance the process of conducting simulations, such as automated parameter sweeping infrastructure and entity states logging. During the process of replication, obstacles emerged such as numerous researcher induced errors and the difficulty of definitely linking relationships between source code and emergent behavior. Suggestions and techniques for reducing those obstacles are provided in the summary section.

2. Methodology

This section reviews the Sugarscape model and the MASON agent-based modeling and simulation toolkit and how I went about replicating Sugarscape using MASON.

Agent-based Modeling and Sugarscape

Epstein and Axtell (1996) offer a framework – Sugarscape – for agent-based modeling and simulation that revolves around the following elements:

- *Agents* – the "people" of artificial societies. Agents have internal states, behavioral rules, and interact with the environment. Some states, such as economic preferences, wealth, cultural identity, and health can change as agents move and interact with the environment and other agents.
- *Environment* – Spaces or a medium separate from agents, "on which agents operate and with which they interact." There are landscapes of renewable resources and more abstract structures such as communication networks. The space is a two-dimensional lattice called a "sugarscape" and realized as a torus. (p. 22)
- *Rules* – behavioral drivers for agents and sites of the environment. Sites/cells of the environment can be coupled via rules. In general, there are agent-environment-rules, environment-environment rules, and agent-agent rules that govern interactions such as mating, combat, or trade.

Epstein and Axtell (1996, p. 6) 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."

There are differences between theoretical and empirical models. Epstein and Axtell do not direct compare Sugarscape to any specific social milieu – although they compare qualitative outcomes with empirical observations from various fields like economics and epidemiology. Doran points out that, "Unlike agent-based models, artificial societies are, in essence, models without a specific target system [i.e., theoretical models] and it has been argued that this type of modeling permits the study of societies and their processes in the abstract (Epstein and Axtell 1996; Doran 1997)." (Doran, 2000a, p. 17) Doran also states "processual regularities within computer-based societies" are discovered (rather than validity of the model relative to a target system), "then they become available for use in a variety of application contexts." (2000b, p. 17) The Artificial Anasazi Project may blur the line between Doran's separation of artificial societies and agent-based models:

> The project was created to provide an empirical, 'real-world' evaluation of the principles and procedures embodied in the Sugarscape model and to explore the ways in which bottom-up, agent-based computer simulations can illuminate human behavior in a real-world setting. (Dean et al., 2000, p. 180).

Besides the Anasazi model, to date there have been fewer empirical ABMs. Liverani and Parisi (1998) and Parisi (1998) have developed a model for the rise and fall of the Assyrian empire, Malkov (2004) has developed a model for the emergence of the Silk Road network across Inner Asia, and Huang et al. (2005) simulate the dynamics of SARS transmission in Singapore, Taipei, and Toronto.

From a software engineering perspective, the Sugarscape framework rests on an 'object-oriented' approach (Epstein and Axtell 1996, p. 5). This approach consists of:

- *Instance variables* representing agents' internal states (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 does contain a short section on OO techniques used and considered. Polymorphism is not discussed, but the authors state inheritance was considered for the agents class. This design approach was not used due to "efficiency considerations …. In total, each agent has over 100 methods." (Epstein and Axtell 1996, p. 180) By comparison, the agent class in MASON Sugarscape has approximately 32 methods, although only 75-80% of all Sugarscape rules were implemented.

OO methods are supposed to have a variety of benefits regarding design efficiency, clarity, and code reuse. One disadvantage to OO schemes is that OO semantics and syntax may make models more difficult to understand and change for researchers with less or minimal software-engineering training. Complex behavioral dependencies between instance variables, methods, and classes can result significant work when evolving models, and software implementers must balance performance, code readability, and code evolvability.

Sugarscape Rules

As rules are the drivers for model dynamics, Table 1 summarizes the major rules described in *GAS* (Epstein and Axtell, Appendix B, p. 182-185).

Table 1. Major rules in the classic Sugarscape mode
(Epstein & Axtell 1996, p. 182–185)

Symbol	**Name**
G_{α}	Sugarscape growback
M	Agent movement
$R_{[a,b]}$	Agent replacement
$S_{\alpha\beta\gamma}$	Seasonal growback
$P_{\pi,x}$	Pollution formation
D_{α}	Pollution diffusion
S	Agent mating
I	Agent inheritance
None	Agent cultural transmission
None	Group membership
K	Agent culture
C_{α}	Agent combat
T	Agent trade
L_{dr}	Agent credit
none	Agent immune response
none	Disease transmission
E	Agent disease processes

Rules in Table 1 are the primary behavior drivers for agents and environment sites, and are important for understand simulation outcomes. Each rule is reviewed below, with the purpose of identifying major features, assumptions, and logic, particularly where social phenomena are involved.

Agent culture (rule **K**) is a rule which exemplifies social and computational aspects of Sugarscape. It is a combination of 'agent cultural transmission' and 'agent group membership' rules given immediately above. The agent group membership rule is stated as "Agents are defined to be members of the blue group when 0s outnumber 1s on their tag strings, and members of the Red group in the opposite case." Agent cultural transmission (a rule without a symbol) is defined as:

- Select a neighboring agent at random;
- Select a tag randomly;
- If the neighbor agrees with the agent at that tag position, no change is made; if they disagree, the neighbor's tag is flipped to agree with the agent's tag;
- Repeat for all neighbors.

Epstein and Axtell have modeled a simple representation for beliefs and belief changes via the culture rules. There are no belief strengths nor are there explicit means for agents to have beliefs about social structures or institutions – standardized modes of behavior – as such. These culture rules, however, may in fact embody some of the important aspects of agent knowledge about institutions and related behavior since agent group membership does effect agent behavior through the combat. Within the core rule

set, cultural tags have no significance or meaning for agents other than group membership and combat. Via an extension to the **Trade** rule (**T**), however, **Culture** does directly effect agent behavior – "here we let economic preferences vary according to the state of an agent's cultural tags" (1996, p. 124) One simulation effect of this extension to rule **T** and **Sexual Reproduction** (**S**) is "to produce so much variation in price that equilibrium seems lost forever." (1996, p. 126) Although this rule extension does not have explicit notation, its impact on agent behavior is significant as trade effects movement, reproduction, and resource consumption. At the most macro level of description, this culture rule extension effects agent-agent and agent-environment dynamics.

A primary focus in social modeling is understanding how internal states that are not physical ones – material holdings, age, other biological factors – affect behavior with other agents and the environment. This why it is important to understand the linkages that **Culture** and other cognitive rules have to do with observable behavior.

Although not listed in Appendix B, Epstein and Axtell (1996, p. 79-80) describe an additional rule which links culture and social and environment behavior.

> "Each agent keeps track of the five agents it has encountered who are nearest it culturally; these are its friends. Each time an agent encounters a new neighbor the agent determines how close they are culturally and, if the neighbor is closer than any of the agent's five friends, the neighbor displaces one of them."

Epstein and Axtell typically describe simulation runs or results by the rules that are operative during those runs. In the case of the "Network of Friends" run (1996, p. 81), rules $\mathbf{G_1}$, **M**, and **K** (only these three rules are operating) are used to generate statistics like average closeness of best friends and average closeness of all friends. Additionally, visual presentation depicts spatial patterns of agents and their friendship states via connecting lines.

Epstein and Axtell explore the dynamics of wealth distribution – a critical area of social science research in which to demonstrate the power of generative social science – and they used an inequality measure called a Gini to understand the relationships between local rules and emergent structures (1996, p. 32-37). In particular, they point out that initial symmetrical (uniform) distributions always change to highly skewed distributions and that these results are constant for a wide range of agent and environment specifications (1996, p. 33). Epstein and Axtell point out adjusting local rules – such as inheritance and taxation – to determine if the same global patterns emerge "is one of the most powerful features of artificial societies." (1996, 37)

Since publication of their primary description of Sugarscape, Axtell and Epstein have developed some extensions or variations to their approach. Epstein (2000), for example, has developed a rule or update procedure called "Best Reply to Adaptive Sample Evidence." Epstein's goal was to explore agents' changing norms and the relationship between cognitive effort (agent computation), norm strength, and norm-based behavior. This model has very few rules and does not use the Sugarscape framework or rule set in a comprehensive way, such as the simulations described under rules $\{\mathbf{G_1}\}$, **M**, **S**, **K**, **T**, $\mathbf{L_{10,10}}$, **E** (Epstein and Axtell, 1996, p. 154-155).

In terms of institutional forms and social structures in general, Epstein and Axtell (1996, p. 164-165) describe a long term vision of agent-based models – "Ultimately, one would like to see if wage labor, firms, elaborate production hierarchies, and various forms of specialization (division of labor) can be made to emerge in agent-based models. But, in the near term we would be happy with far more modest results."

Wealth Distribution Constructs in Sugarscape

The classic Sugarscape reports metrics such as the Gini coefficient, which is used to describe wealth distribution in social systems. Since Gini coefficients and Lorenz curves are not typical statistics supplied in with agent-based modeling toolkits, this section presents a short review of these constructs.

Kleiber and Kotz (2003, p. 30) define the Gini coefficient as "twice the area between the Lorenz curve and the 'equality line'":

$$G = 1 - 2\int_0^1 L(u)du$$

And the Lorenz curve (L) maybe be calculated and plotted from empirical data using

$$L\binom{k}{n} = \frac{\sum_{i=1}^{k} x_{i:n}}{\sum_{i=1}^{n} x_{i:n}}, k = 0, 1, ..., n,$$

where $x_{i:n}$ denotes the *i*th smallest income, and n = the number of households. Figure 1 shows a Lorenz curve and the derived Gini coefficient. The more concave the curve, the greater the coefficient.

Kleiber and Kotz (2003) provide an extensive treatment of these and numerous other properties of wealth distributions, including alternative measures and approximations.

Kleiber and Kotz (2003, p. 30) believe that interest in Gini coefficients as a primary measure of income inequality has become nearly obsessive and "is an unhealthy and possibly misleading development." The UNDP (2004) reports a composite index called the Human Development Index (HDI); this index makes use of GDP statistics for nations but does not include Gini coefficients.

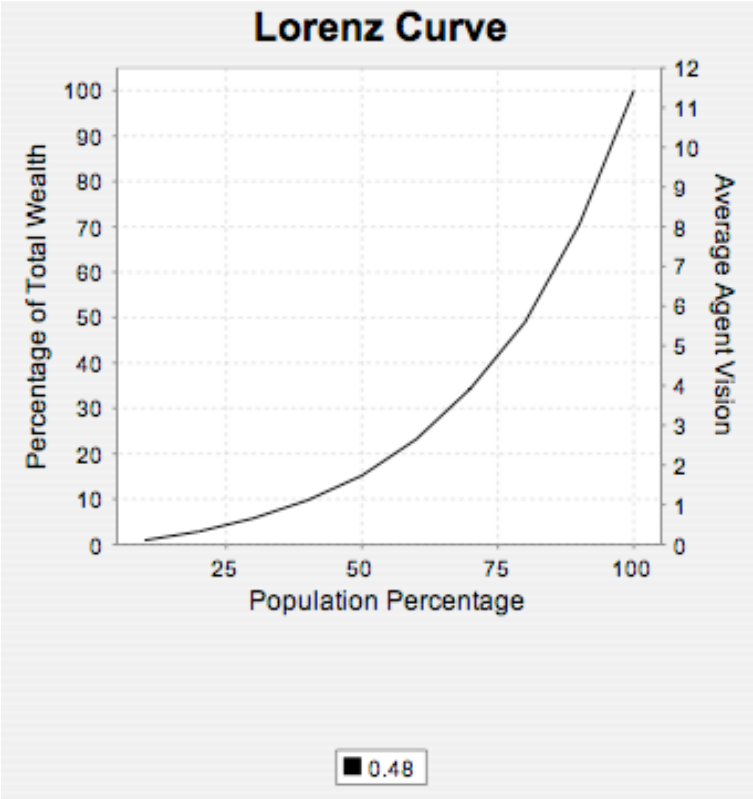


Figure 1. Example Lorenz curve

The MASON Multi-Agent Simulation Toolkit

Since the publication of *Growing Artificial Societies* in 1996, several multi-agent or agent-based modeling toolkits have emerged or matured and MASON is one of the newest available. Luke et al. (2004, p. 1-2) describe MASON as a "fast, easily extendable, discrete-event multi-agent simulation toolkit in Java" in the tradition of SWARM, Repast, and other agent-based modeling toolkits. Their design goals were to build a toolkit easily extended by "an experienced Java programmer", visualization capabilities separate from models, results independent of platform, simulation checkpointing to disk files and simulation restoration with or without visualization, support for up to 1 million agents without visualization, efficient support for visualizing large numbers of agents, and easy to embed in other existing libraries "including having multiple instantiations of the system co-resident in memory."

Luke et al. (2004, p. 2) describe three potential design goals that MASON developers have not pursued. These goals are: support for parallelizing a single simulation across a network of multiple systems/CPUs, built-in features for domains such as social systems or robots, and memory efficiency.

Researchers have used the core MASON toolkit and extensions to implement models in a variety of domains, including models that have become de rigueur. One model that has not yet been implemented using MASON is Sugarscape. Simple versions of Sugarscape have been implemented in SWARM and RePast; TeamBots, Repast, and ECJ have influenced the design of MASON (Luke et al., 2004, p. 2). MASON was chosen to implement Sugarscape because it was designed to be fast, has explicit infrastructure for scheduling, spatial data structures, random number generation (using `ec.util.MersenneTwisterFast`), graphical visualization, and affords easy integration with free charting libraries such as JFreeChart.

Replication Methodology

Implementation of Sugarscape using MASON proceeded via review of each rule's behavior, algorithm, and simulation outcomes as described in *GAS*. Review of the outcomes included identifying what kinds of statistics and chart output were necessary. To the maximum extent possible, MASON classes were used to implement core model functionality and graphical and statistical graphs functionality.

Agent and environment rule development was an iterative process. Once basic behavior was established by inspection of the visual display during simulation and via diagnostic output, simulation outcomes – figures and animations – were studied to recreate animation frames and statistical charts, and further verify correct rule implementation. One technique that was not used during software development was 'unit testing' or writing explicit testing and verification routines prior to source code being written. This technique would probably have significantly reduced human produced throughout the process and saved time.

In the latter phases of the research, I contacted both authors and asked questions regarding assumptions, implementation aspects, and parameters for simulation runs. The authors were able to identify a major incorrect assumption of mine and to clarify some of the parameter settings and specific behavior aspects (i.e., will an agent stay on its current site if the sugar there is greater than any other site within vision). I also had direct access to the primary designer of MASON and was able to ask very specific questions about performance and usage of specific classes.

3. Results

This section presents overall replications goals, results, including implementation techniques, new infrastructure added to MASON, rules and Sugarscape outcomes selected for replication.

Replication Goals

The overall objectives of the replication efforts were to demonstrate general behavioral congruence with the classic model and to replicate aggregate qualitative outcomes as documented in various animations and tables. This effort entailed creating an integrated model, replicating selected environment and agent rules, and creating supporting infrastructure for graphs output and simulation parameter sweeps.

Model

The model structure consists of four primary elements: agent rules, environment rules, an environment, and visual/GUI components. Rules may be thought of as behaviors that are associated with each agent or environment resource entity, or behaviors that are associated with entities as a collection or aggregate. The class diagram below depicts the non-graphical primary classes (Figure 2).

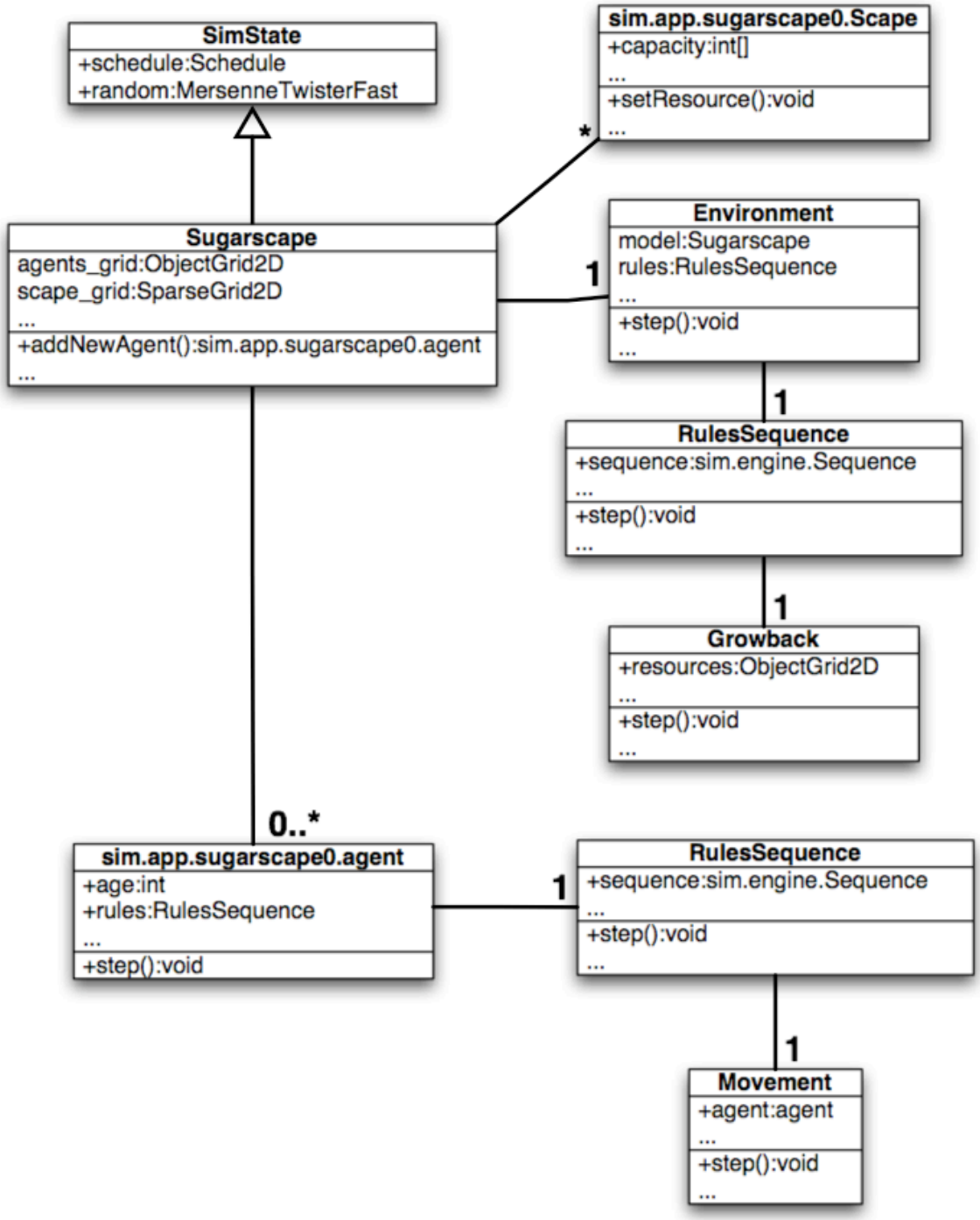


Figure 2. Class diagram of primary MASON Sugarscape classes

Scheduling and Timing

Controlling and determining what behaviors are executed and when is a critical aspect 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 scheduling mechanism that provides precise control.

The MASON Version 8 `Schedule` class is responsible for sequencing and executing 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`. An order is a collection of executable objects that enable a deterministic sequence of execution. For each time step, all order zero objects are executed first, order one objects are executed next, and so forth. Implementation of rules in specific orders is a key aspect of this replication, and the orders for rules in the general model is as follows:

0 environment rules

1 agents rules

2 open

3 seasons

4 statistics and logging

5 open

6 text histogram chart

In this listing, "open" indicates the order is not used and is simply a result of evolving the model and experimenting with MASON. Although `seasons` is a variation of the **Growback** environmental rule in the classic Sugarscape model, it is treated as a separate rule and is executed in order 3. Given the current MASON Sugarscape model, it could easily be incorporated into the environment rules. Orders for agent and environment rules are specified in the Sugarscape class as constants, with all simulations results described in the Simulation Infrastructure section. These constants could easily be redesigned as parameters in the primary configuration file for more flexible experimentation. Finally, the text histogram chart class uses order 6; this diagnostic tool is further described in the Rules section.

`Seasons`, in addition to statistics, charts and logging, uses another MASON class – `MultiStep`. This enables `Steppables` – objects that implement the `Steppable` interface – to be executed less than once every time step but with a regular periodicity. In the example below, a `MultiStep` object is instantiated to execute the `season_changer` instance every `season_rate` time steps:

```
MultiStep season_multi = new MultiStep(season_changer,
season_rate, true);
```

An alternative to `MultiStep` is to use a `Schedule.setRepeating` method that has an interval. This has a performance advantage over `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 agents do not determine which of their rules are executed, nor in what sequence. Instead, a new class named `RulesSequence` was created to 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. Sequences are used instead of adding the rule objects directly to the schedule as direct schedule usage has order *n* entities x *m* rules time complexity, and `Sequence` only has order *m* rules time complexity.

Figure 3 presents a UML sequence diagram to illustrate how the example classes relate in time when two agent rules – **Movement** and **Biological** – and one environment rule – **Growback** – are active.

To provide for a flexible means of experimentation, a primary configuration file specifies rule execution order during runtime. Using the reflection facilities of the Java language, initialization code translates text names for rules classes into object instantiations. Abbreviations are also specified in the configuration file, providing a quick way to specify desired rule sequences without having to recompile source code. The example below shows an extract from a configuration file specifying agent rule classes, two environment rules classes, abbreviations for each, and execution order:

```
agent_rule1=Movement,M
agent_rule2=Biological,B
agent_rule3=Reproduction,S
agent_rule4=Culture_K,K
agent_rule5=Trade,T
agent_rule6=Pollution,P
environment_rule1=DiffuserPollution,D
environment_rule2=Growback,G
#rules will be executed in order listed
agent_sequence_rules=M,T,B
```

While most rules in Sugarscape involve sequential execution of behavior for each agent or site, there are rules that involve cellular automata (CA) type synchronous treatment of all instances of a class. **Pollution Diffusion** (**D**) is an example of an

environment rule in which the states of all sites are updated synchronously and the rule does not invoke any MASON scheduling machinery.

Sequence diagrams are useful for helping to understand interactions in time, but many agent-based models have discrete space aspects. As an example, interactions in space and time are illustrated in the diagram below (Figure 4). This diagram provides insight into the local effects of harvesting when a) sites grow back during the harvesting time step, and b) sites do not grow back until one full time step after the time step during which harvest occurred.

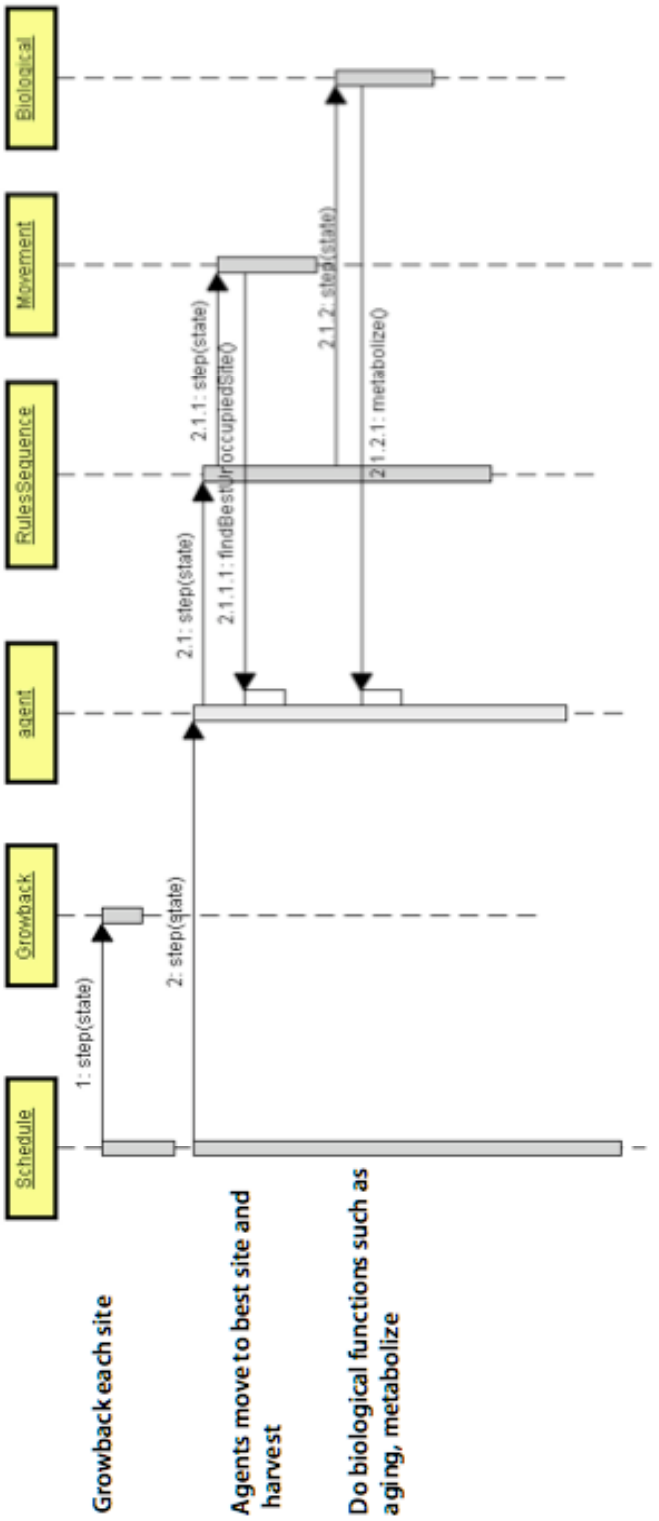


Figure 3. Sequence diagram for primary classes in MASON Sugarscape

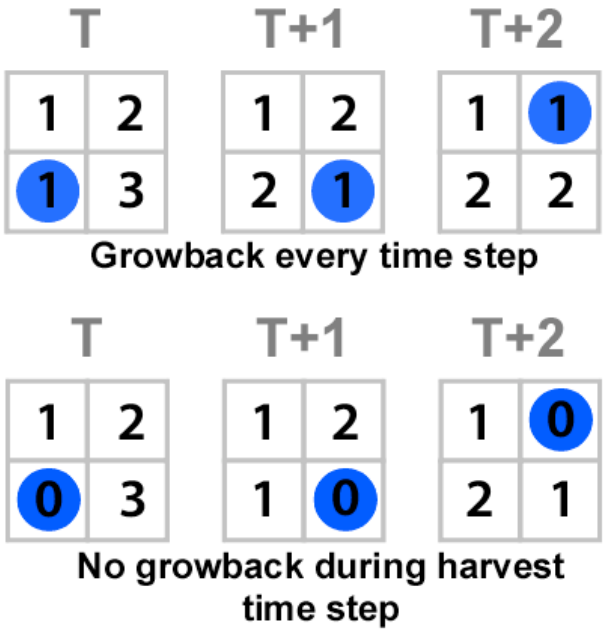


Figure 4. An illustration of **Movement** and **Growback** rules effects under two different harvesting-growback constraints

An agent is represented as a blue circle. At the end of time T, the agent has moved to a site, and harvested the sugar there. For this scenario, the order in the schedule is agents are executed first, then sites. In the top row, the agent sees that the site with 3 units is the nearest and highest level sugar site and moves there during T+1. Since the sites execute in a later order, the new site regenerates to 1 at the end of T+1. At the end of T+2, the agent has moved up one cell and harvests that site, which regenerates to 1. The agent could have gone back to its original time T site (but the upper site was randomly selected). In the second row, the modified **Growback** rule is such that a site can not regenerate *during* a time step, which is why at the end of T, T+1, and T+2, occupied sites have level zero. One can see that the local effect with this rule change is that the agent, in the shown spatial configuration, will not 'backtrack' and that sites where agents have been are one level less in the top row than the bottom row *even when not occupied by the agent.* The persistence and global impact of this effect is mitigated when population density is low – i.e., less competition for resources – and due to the random selection of equidistant sites having equal resources, and finally by a combination of these two factors. The high density starting agent block of Animation II-6 (see Simulation Methodology and Results section below) and the emergent wave phenomenon graphically illustrate the global effects of growback prohibition during a harvest period.

Timing issues are crucial in discrete event simulations. The `Schedule` class affords precise and easy control, although this class does not remove the burden of researchers clearly understanding the overt and subtle effects of timing.

Data Structures

The two primary data structures hold the two types of entities in the model: agents and site-resources. The original Sugarscape model contained agents and sites on a lattice – a discrete grid-like space. MASON has provisions for a variety of spatial data structures, including several types of grids, and each grid type has different computational performance characteristics. The two primary spatial data structures in the model class are shown in Figure 3 above.

To hold agents, this model uses the `SparseGrid2D` class. This class contains a variety of methods for searching for and storing objects. It also contains support for toroidal constraints, which address the original Sugarscape space design. `SparseGrid2D` is efficient at determining object locations and searching through all objects and is recommended for use when the number of objects is significantly less than the number of discrete locations (Luke et al., 2005b). Sugarscape has a 50 by 50 grid, often with less than 500 total agents existing in that space, i.e., less than a 1:5 ratio of agents to sites.

To hold resource sites – sugar and/or spice – the model uses `ObjectGrid2D`. This class also contains many methods for searching and storing objects. Although this class is convenient space for a model in which there are one or more objects at all lattice intersections, there are performance implications for associated graphical rendering classes. Other utility classes heavily used in the model are `MutableInt2D, Int2D`, and `Bag`. The two `Int2D` classes are used to represented coordinates for agents and sites, and `Bag` is a general purpose resizable data structure that offers the flexibility of the Java `Collections` classes and some of the performance of arrays.

The `sim.app.sugarscape0.agent` and `Scape` classes use a variety of standard Java primitives to represent internal states, as well as the aforementioned MASON utility classes. To facilitate agent-environment rule interactions, many internal state variables are declared public.

Rules

The table below describes selected 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. The Simulation Methodology and Results section describes results from simulation runs using each rule and combinations of rules as described throughout *GAS*.

Appendix 3 contains a list of classes sorted by SLOC (not including comments or white space). Total source lines of code are approximately 3500, with approximately 100 lines being optional diagnostic print out lines. 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,

instantiating agents, their rules, and sites, and simulation management infrastructure such as 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. SLOC statistics were generated by the SLOCCOUNT tool (Wheeler, 2005).

Table 2. Sugarscape rules implemented in MASON

Symbol	Name	Sequenceable	SLOC
$\mathbf{G_{\alpha}}$	Sugarscape growback	Yes	42
M	Agent movement	Yes	323
$\mathbf{R_{[a,b]}}$	Agent replacement	No	17
$\mathbf{S_{\alpha\beta\gamma}}$	Seasonal growback	Yes	16
$\mathbf{P_{\pi,x}}$	Pollution formation	Yes	18
$\mathbf{D_{\alpha}}$	Pollution diffusion	Yes	56
S	Agent mating	Yes	163
None	Agent cultural transmission	Yes	128
None	Group membership	Yes	
K	Agent culture	Yes	
T	Agent trade	Yes	177

Source lines of code statistics do not tell a complete story about the nature of each rule. Difficulty of implementation and testing varies with the nature and amount of calculations being performed, numbers of conditional, and other factors. Additionally, since code for each rule may be spread across classes, the SLOC count for the rules does not have a 1:1 correspondence to SLOC counts in Appendix 3. Multithreaded code may present significant complexity, but the MASON Sugarscape codebase is single-threaded except for `Logger`.

The following Sugarscape rules were not implemented: **Inheritance** (non-sexual reproduction), **vertical cultural transmission, Combat, Foresight, Credit, Disease processes**. While functioning forms of each could be implemented, time limits in the research and the fact that all preceding rules outcomes' were not exactly replicated resulted in stopping implementation at rule **T**.

The remainder of this section briefly reviews all rules and describes implementation requirements, techniques, and challenges for each. The purpose of this section is not to critique the social science and conceptual aspects of the original Sugarscape model, but to examine how replication was achieved using MASON and difficulties that arose. Appendix 2 provides pseudocode for each rule.

G_α Sugarscape growback

Growback is a core rule in this model and is only other means besides **Trade** that agents can acquire life-sustaining resources. In *GAS* Chapters 2 and 3, **Growback** concerns just one resource, sugar. Chapter 4 includes a second resource, spice, and all sites in the space contain some levels of both sugar and spice.

Since each site is independent of other sites, the order in which the sites regenerate during each time step is unimportant. **G**, as implemented, simply contains two loops that process the raw array of `Scape` (site) objects directly accessible from `ObjectGrid2D`. The subscript indicates the number of units of grow back per time step.

M Agent movement

Although seemingly a simple rule, agent movement is complex because its behavior includes biological processes, such as metabolic processing, visually scanning for resources and other agents, and welfare estimation. Given the variety of functions performed and the presence of the **Movement** rule in nearly every simulation and rule combination, it became apparent during this research that both 'correct' and efficient execution of movement was critical. The Simulation Methodology and Results section addresses this issue in greater detail and generalizes computational modeling issues from experience in code development and simulation testing involving **Movement**.

To enhance code readability and development, this rule was split into two separate rules – **Movement** and **Biological**. **Movement** encompasses vision scanning, actual movement from one site to another, and consumption of resources acquired. **Biological** encompasses metabolic processes, aging, and death.

There are various points in the model and in rules where behavior must be random – there have been criticisms of results obtained in other computational efforts due to some kind of unintentional bias rather than random behavior. In movement, when resources of equal magnitude and equal distance are found, one site must be randomly selected. Among sites having equal levels of sugar (and or spice), the selected site is the closest one – this is an intentional bias.

$R_{[a,b]}$ Agent replacement

To keep a population constant, **Replacement** controls minimum and maximum ages that agent can live before being replaced. This rule is currently implemented as part of the **Biological** rule – upon death, the **Biological** rule checks to see if a global `boolean` variable indicates whether replacement occurs. If the replacement rule is active, the agent is removed from the `SparseGrid2D` instance, and from the schedule via the `Stoppable` instance obtained from schedule when the agent was first added. In this research, simulation runs of 2500 steps (described in the results description for *GAS*, Figure III-2) resulted in 30000 total agents executed – 700-1000 were alive at any given time.

Finally, a method within the main model class is invoked to instantiate a new agent and add it to the schedule for execution starting at `Schedule.time()+1`. It would be

simple to reconstruct this rule as a `Steppable` rule like **Movement** or **Biological** by checking to see if an agent's death occurred in the current time step. The difference would be that all dead agents would be replaced at once – i.e., after the execution of the other agent rules for all other agents – rather than during the execution of each agent's rule set; the qualitative differences generated by these implementation in simulation outcomes was not explored.

$S_{\alpha\beta\gamma}$ Seasonal growback

This is a simple rule that involves changing model parameters regarding growback rates. A global model variable describes the season in each hemisphere and the **Growback** rule simply checks the model variable as it loops through each site – sites are instantiated with an instance variable representing their hemisphere.

In this implementation, seasons always occur, but season effects are nil when the configuration file specifies values of 1 for α (summer grow back rate), 1 for β (winter grow back rate), or γ (season change rate) is set to a very large number that exceeds the number of time steps for any given simulation run. This rule was the first to take advantage of the `MultiStep` class in that change in seasons rate is used as the frequency at which the schedule executes the rule.

P Pollution formation

Agents generate pollution as they harvest and metabolize resources. The rule was implemented as an agent rule that can be sequenced into the `RulesSequence` class, and it simply invokes a method in `sim.app.sugarscape0.agent` to generate pollution: the `step()` method is shown below:

```
public void step(SimState state) {
       agent.doPollution();
}
```

While this makes the source code in the agent class a little bit longer – it does the computational work of adding the pollution to the site occupied by the agent – it makes evolution of the overall model easier, does not require a `boolean` global variable like **Replacement**, and is more understandable. Users can also easily remove or add **Pollution** formation to the agent rule sequence in the configuration file. In short, **Pollution** formation is implemented the way **Replacement** could and should be implemented.

D_{α} Pollution diffusion

This environment rule complements **Pollution** formation, where α determines the time steps between diffusion execution. The implementation incorporates a cellular automata style mechanism where by each site is affected by its 4-way neighbor sites, but all the sites change at once. Since the data structure for all sites includes a two-dimensional object array, executing this rule is efficient.

S Agent mating

Reproduction via sexual mating is one of the more complex rules in Sugarscape. It involves checking to see if the agent has enough resources to reproduce, checking for potential partners among its 4-way neighbors, checking for a free site on which to place a child agent, simple genetic inheritance from parents, creation of the new child agent, and reducing parents' resource holdings due to child production.

The authors deliberately excluded kinship taboos constraints on mating desiring to keep the construct and resulting code simple. This theme of simplicity is one of the most important themes in the *GAS* and is discussed in Section 4.

Agent cultural transmission
Group membership
K Agent culture

The collection of rules dealing with culture are implemented as a single sequenceable rule, with supporting methods and infrastructure in `sim.app.sugarscape0.agent`. As in the classic model, the number of culture tags or bins is specified at runtime rather than hard-coded into the model. This rule is one of the rules added to a `RulesSequence` instance for each agent. Unlike the CA-style approach of **Pollution Diffusion**, each agent executes **K** such that culture effects are nondeterministic at a local level. An agent that was a neighbor to the agent executing its rules may not be a neighbor if that neighboring agent moves to a site where it can no longer see the first agent. The execution order of each agent for each time period is random, further contributing to the non-determinism of local culture dynamics via pseudo-parallelism.

T Agent trade

This rule may be the most complex rule in Sugarscape. It requires multiple resources types in the resource space and a multi-commoditywelfare estimation rule, implemented as a Cobb-Douglas utility function (Epstein and Axtell, p. 97). The trade rule itself involves several floating point-based equations. Due to the computationally intensive requirements for welfare calculation, and the fact that trade with one neighbor requires recalculation of welfare to estimate welfare changes for potential trades with other neighbors, an effort was made to optimize performance. The result of the code redesign was that this rule does not appear to consume an inordinate amount of CPU time.

During testing of this rule, simulation results did not match those described in *GAS*. After direct consultation with the *GAS* authors, it was discovered that MASON Sugarscape did not have sugar and spice at every site. Though this is stipulated in *GAS*, it is briefly mentioned only once. Even after necessary changes to the model code were made to address this problem, simulation outcomes still did not exactly match. This resulted in a review of the core **Movement**, **Biological**, and other core infrastructure and a few errors and incorrect implementation aspects. Although it may common sense in

retrospect, core rules require significant care to implement correctly and that adequate testing of the core rules should be as conducted soon as feasible in the overall model development process.

Simulation Infrastructure

MASON lacks infrastructure for managing simulation runs and automatically generating and executing parameter sets; MASON 9 has provisions for parameter loading. Graphical visualizations of agent and resource dynamics are separable aspects of model building and simulation. MASON's design decouples visualization from simulation, but the two may be easily used in conjunction. This section describes work done in creating visual and graphing elements in MASON Sugarscape using MASON infrastructure, as well as new work to enable automatic parameter sweeping.

The classic Sugarscape model embodies two graphical components: portrayal of the agents and resources, and graphing and charting to show current and end of simulation aggregate statistics. MASON has facilities for graphical portrayal and these were used and customized. While MASON does not focus on graphs and charts, the software distribution contains examples for using an open-source Java library, JFreeChart; these examples are generally adequate for researchers not familiar with JFreeChart to incorporate these classes into the MASON GUI environment. More complex charts and graphs may required access to the non-free developers guide and is recommended.

This implementation exploited portrayal classes in MASON associated with each data structure for the agents and the sites. A `SparseGridPortrayal2D` instance was used for the agents in the `SparseGrid2D` instance, and either an `ObjectPortrayal2D` was used for the sites in an `ObjectGrid2D` instance or a `FastObjectGridPortrayal2D` instance. The `ObjectGrid2D` class provides a high potential degree of control over the shape, size, and color of a visual object, while the `FastObjectGridPortrayal2D` class executes 2-3 times faster on Mac OS 10.3 running Java 1.4.2_05, but offers much less control over the visual attributes of the entity. Finally, two new subclasses of `SimplePortrayal2D` were created to render the agents and sites; `SimplePortrayal2D` is meant for this purpose.

JFreeChart offers a wide variety of chart types and features for controlling chart appearances. Charts and data structures to drive the charts are simple to construct or instantiate. Javadoc documentation is available online, but a developer's guide costs a small amount of money. Replication of graphs depicted in *GAS* with JFreeChart was relatively straightforward.

In order to provide real-time, but lightweight statistical presentation capability, the new `Histogram` class was written to produce text character histograms to standard output. This class was used during certain diagnosis phases of rules development. Figure 5 shows an example where the average vision for each bin are labeled at the bottom of the x-axis, counts for each metabolism bin are along the y-axis, and average vision for the agents in each bin is annotated at the top of the graph. The bin labels are currently limited to single digit values.

For the purpose of generating graphs like Figure II-5 in *GAS*, infrastructure was developed for automatically generating all permutations of a parameter set and executing N time steps, M loops of each parameter set – i.e., a parameter sweep. The core of this capability is a recursive method used to generate the entire parameter space. In a text configuration file, one specifies variables of interest, their starting and stopping values, and step size, and the parameter sweeping code automatically generates all possible permutations and execute each parameter set for N time steps, M repetitions. Additionally, a multiple line graph can be generated automatically upon completion of all simulations runs on a configuration by instantiating a simple results visualization class. This graph depicts one independent variable parameterized by another independent variable and the dependent variable will be depicted on the y-axis. The average value of the independent variable for the M repetitions is used as the x-axis value. An example graph is provided in the Animation II-2 section below.

```
------------------------------
         Vision
         5  5  5  9
50            **
45            **
40         ** **
35         ** **
30         ** **
25         ** **
20       ** ** **
15       ** ** **
10       ** ** ** **
5        ** ** ** **
0        ** ** ** **
         2  3  4  5  6
         Metabolism
------------------------------
```

Figure 5. Example output from `Histogram`
A text histogram shows agent metabolism bins on the x-axis, frequency of occurrence on the y-axis, and average agent vision per bin along the top.

A current limitation of this code is that values in the parameter set are used to generate normal distributions, but no control is provided over the type of distribution, nor its shape. Care must be taken to use the same output name in the statistics of the model as one specifies in the results graphing configuration file. Both of these constraints should be addressed in further work.

Parameter sweeping infrastructure affords almost completely automatic exploration of parameter space, although distributing parameter sets across loosely coupled grid nodes would provide significant time savings, especially for large parameter sets and complex, long-running simulations.

New logging facilities were created to support more efficient model development and testing. One simply specifies space data structure class names in the configuration file, and all objects' states – agents, sites – are dynamically and automatically inspected using Java reflection during runtime and output via a separate thread to a file. The only constraint on this capability is that objects' public `get*()` methods are exclusively used to obtain states at the end of each time step. The resulting log files can be extremely large, but are useful diagnosing rare, but important states and state transitions. Section 4 describes additional log file exploitation methods, such as fractal measures generation, that might benefit the modeling and simulation process.

Simulation Methodology and Results

One of the primary goals of this research was to attempt to duplicate the qualitative outcomes of simulation runs as measured by aggregate statistics, emergent visual phenomena, and charts described in *GAS*. This section describes, rule by rule, simulations conducted and how results compared with results presented in *GAS*.

Table 3 provides an overall summary of rules sets investigated and a general assessment regarding how well MASONN Sugarscape simulation runs replicated Sugarscape. If replicating animations was the goal, screenshots matching the first and last panels in each *GAS* animation figure are typically provided. Single screenshots, graphs, and other descriptions comprise other evidence presented for each result listed in Table 3.

Table 3. Summary of Simulation Outcomes and Rule Sets Investigated

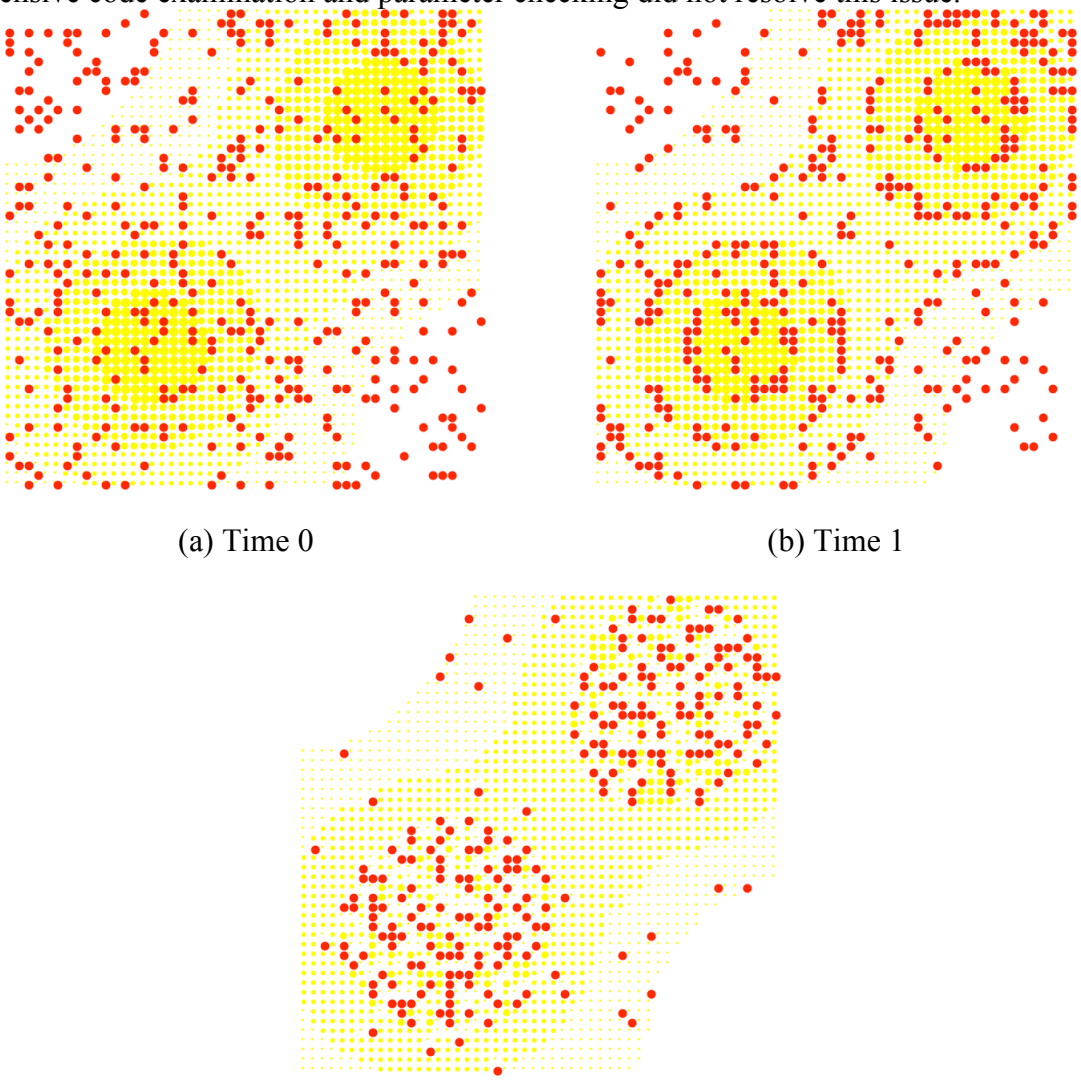
Outcome	Rule Set	Replication Criteria	Replication Achieved
Animation II-2 (p.29)	({G_1}, {M})	Hiving, peak clustering, terrace sticking	Exact
Figure II-5 (p. 31)	({G_1}, {M})	Small positive slopes, equally spaced lines, visually estimated line coordinates	General
Animation II-3 (p. 34)	({G_1},{M,$R_{[60,100]}$})	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)	({$S_{1,8,50}$),{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

The general pattern is that replication was mostly successful, but with some exceptions that indicate the likelihood of an incorrect implementation of the **Movement** rule due to the effects of welfare estimation and spatial dynamics in this rule. This conclusion is derived from evidence described in the rest of this section describing each result.

Animation II-2

The qualitative outcomes of this simulation are entirely visual. *GAS* describes hiving, peak clustering, and initial terrace edge sticking by agents. These phenomena were replicated. Figure 6 shows three screenshots from time 0, time 1, and time 150.

This outcome demonstrates carrying capacity as a function of mean agent vision, parameterized by metabolism, under **G** and **M** rules. A starting population of 500 agents is used. By 500 time steps, the populations become asymptotic. As depicted in Figure 7, MASON Sugarscape simulations generally replicate the graph outcome, although the lines are shifted down by about 20% in each case and the slopes do not match. Lower carrying capacity than *GAS* describes occurs in several of the outcomes simulated, and intensive code examination and parameter checking did not resolve this issue.



(a) Time 0 (b) Time 1

(c) Time 150

Figure 6. Emergent spatial behavior in Animation II-2

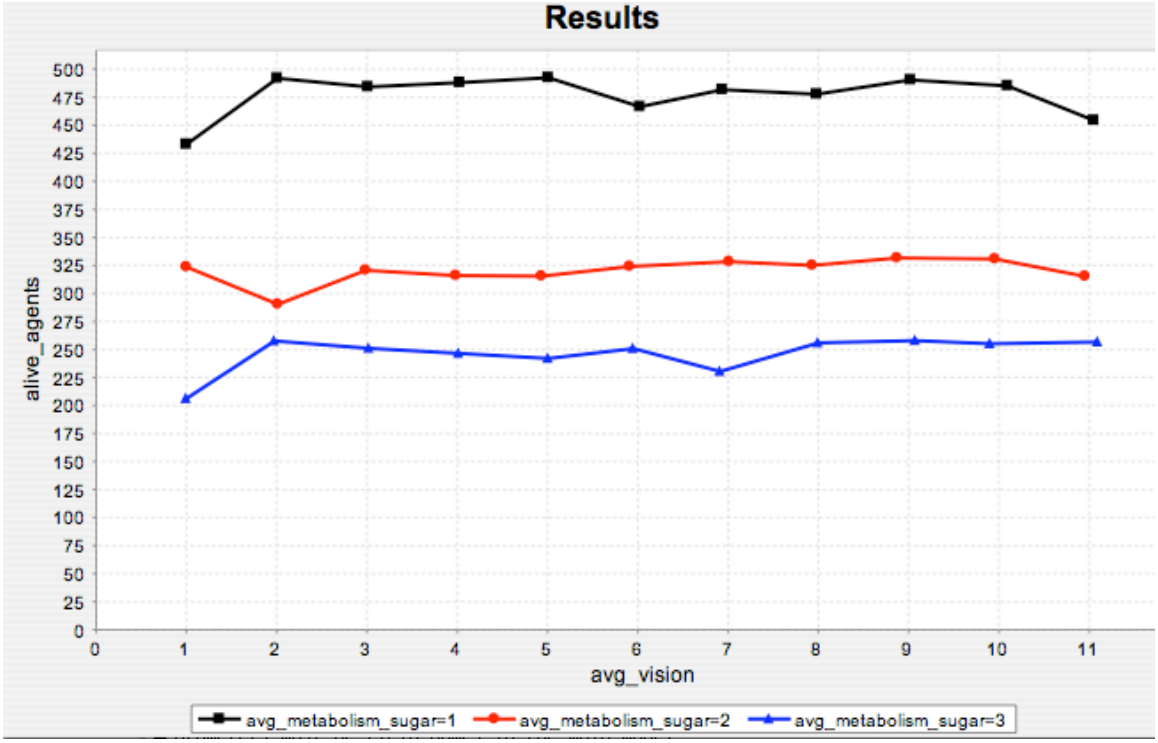
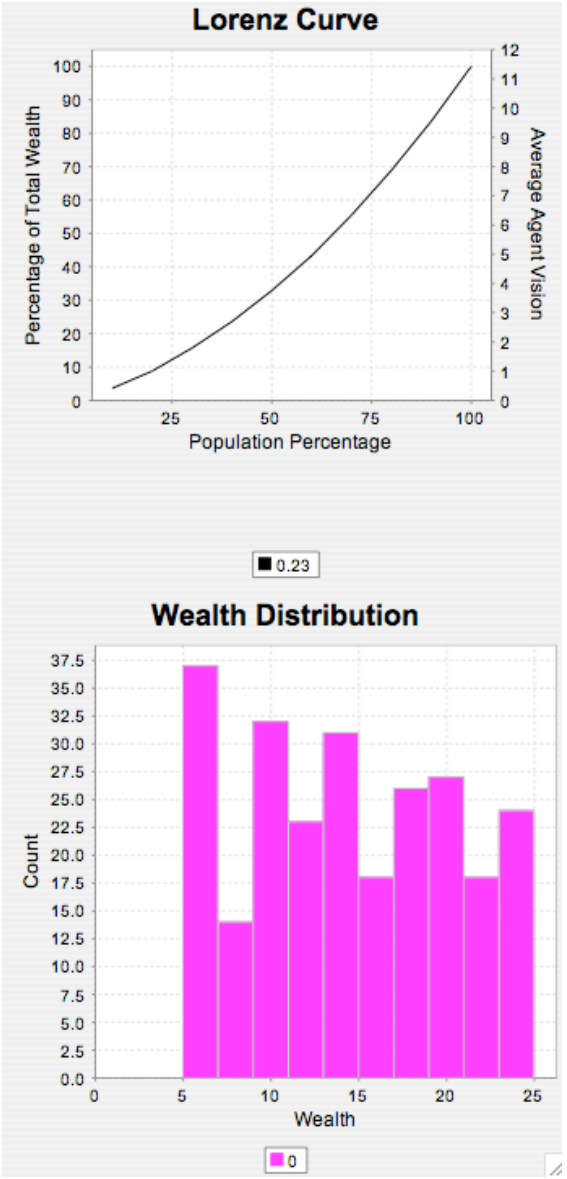


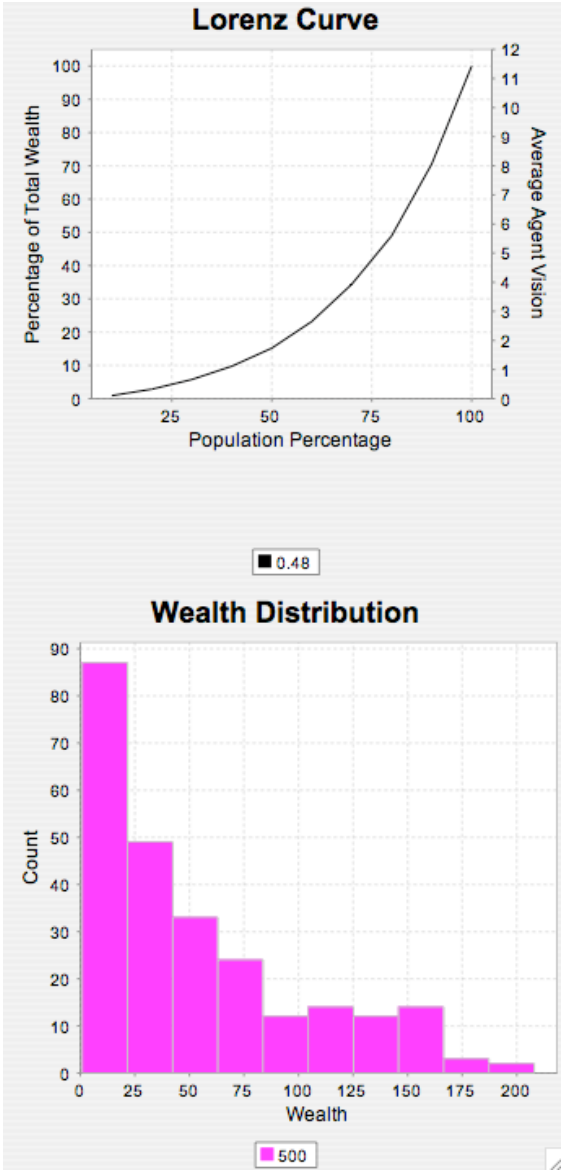
Figure 7. Carrying capacity as a function of average agent vision and average metabolism

Animation II-3

In this outcome, agents are replaced between the ages of 60 to 100 years (time steps) via rule **R**. The qualitative outcome in this animation is a starting symmetrical (normal) distribution of wealth, followed by an eventual Pareto-like distribution where most agents are poor and a few are wealthy. As depicted in Figure 8 for time steps 0 and 500, simulation results generally replicate the Pareto distribution, although the final frame in Animation II-3 features a maximum wealth bin somewhat greater than the maximum wealth bin in MASON Sugarscape. The time step for the final frame in Animation II-3 is not specified. The legend in most figures presented in this thesis from MASON Sugarscape simulations describe the time step.



(a) Step 0



(b) Step 500

Figure 8. Agent wealth distribution produced by Animation II-3

Animation II-4

Lorenz curves and Gini coefficients are presented under **G**, **M**, and **R** rules. MASON Sugarscape simulations, as depicted in Figure 9 below, match to a high degree this outcome. The legend, in this case, indicates the Gini coefficient, which is a

numerical analog to the Lorenz curve. The 0.48 value is very close to the reported end value in Animation II-4 of 0.5.

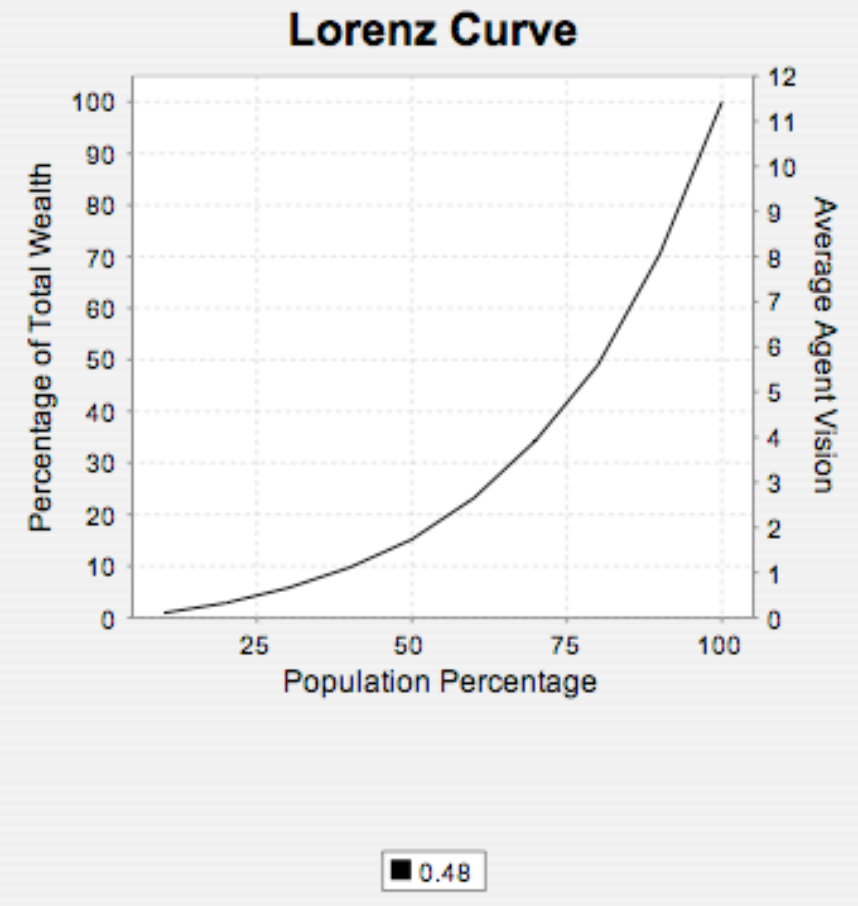


Figure 9. Lorenz curve and Gini coefficient at time step 500 for Animation II-4

Animation II-6

By starting the agents in a block configuration, and increasing maximum vision to 10 (minimum vision is unspecified), waves emerge in a diagonal direction. The significance of this outcome is that it is a demonstration of emergent behavior. None of the individual agents can move in a diagonal direction, but ensembles of these agents appear to move diagonally.

This phenomenon could not be replicated in MASON Sugarscape without delaying the growback for a site that had its sugar harvested for a full time period. Ordinarily, sites growback during a time period even if an agent is on the site and has harvested all the sugar. The line of code that controls this behavior is in the **Movement** rule.

```
      s.time_since_last_regen[a] = 0;  //change to 1 for
normal growback behavior
```

This modification was the simplest change I could imagine that successfully generated the wave phenomenon.

The space-time diagram (Figure 4) and following discussion in Section 3.2.1 detail why this change enables the wave phenomenon; preventing growback during a harvest step prevents an agent from moving 'backwards' until two time steps later. Finally, the waves were more coherent and similar to Animation II-6 if minimum vision was increased to 5 and maximum vision increased to 15.

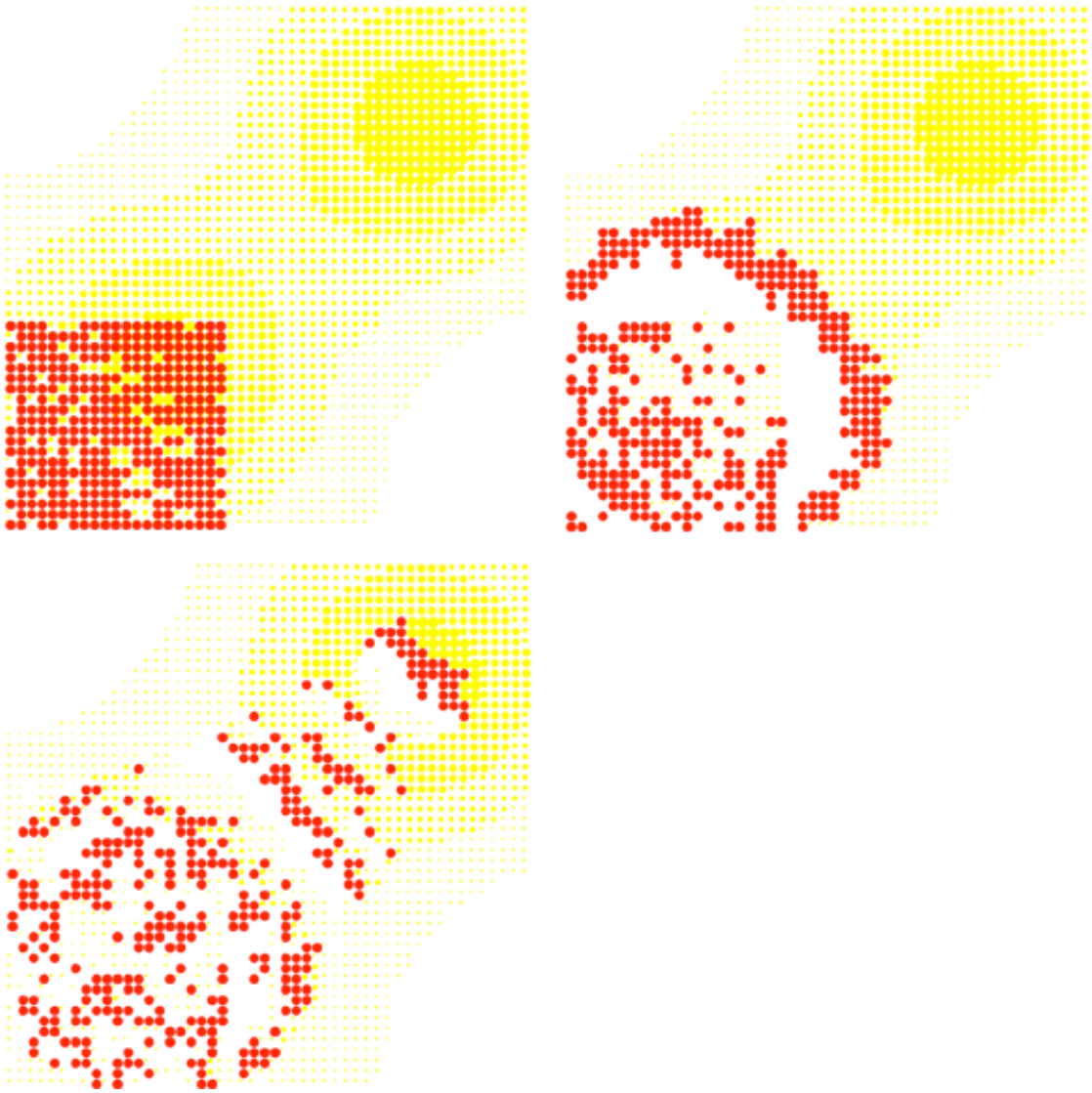


Figure 10. Emergent, migrating waves generated in Animation II-6

Animation II-7

This outcome involves periodic migrations, with seasons changing every 50 time steps, and growback rates in the summer of 1 unit per time step in the summer and 8 time steps per unit in winter. Figure 11 accurately reflects migration patterns described in Animation II-7. Panel 1 represents time step 0, panel 2 represents time step 4, and panel 3 represents time step 53, a few steps after the first season change.

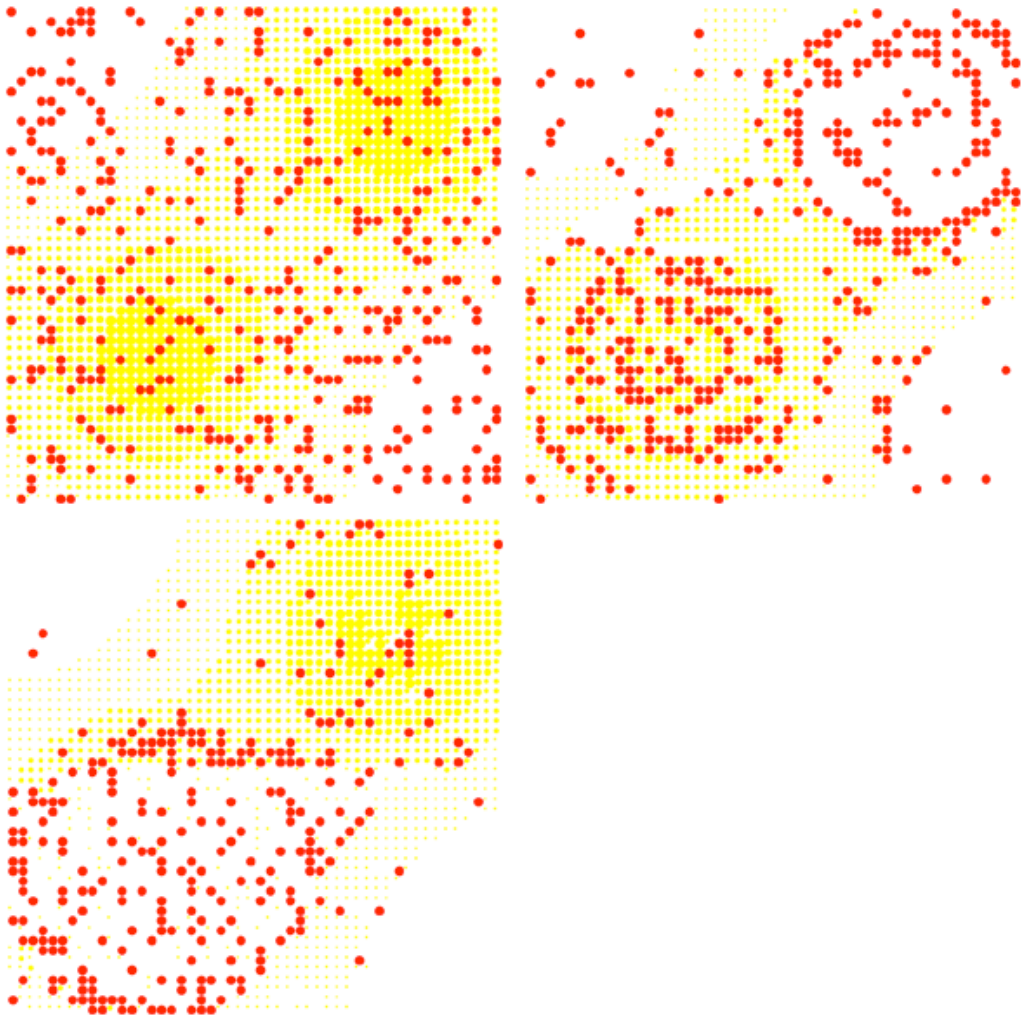


Figure 11. Seasonal effects generated in Animation II-7

Animation II-8

This outcome incorporates two new rules – **P** and **D**. The primary features of this animation are that once **D** becomes active, agents have no choice but to return to high sugar sites regardless of the pollution levels. Eventually pollution spreads to all sites due to diffusion. Figure 12 represents time steps 0, 54 (pollution is generated starting at time step 50), 105 (diffusion starts at time step 100), and 332. MASON Sugarscape output differs in that agents in the later time steps stick to the outer edges of each sugar terrace. This is probably due to the same movement issue identified earlier. In the final panel, one can see that pollution has spread over the entire landscape, with only a two small areas somewhat less pollution than the others. Those areas are where sugar levels are either zero or one.

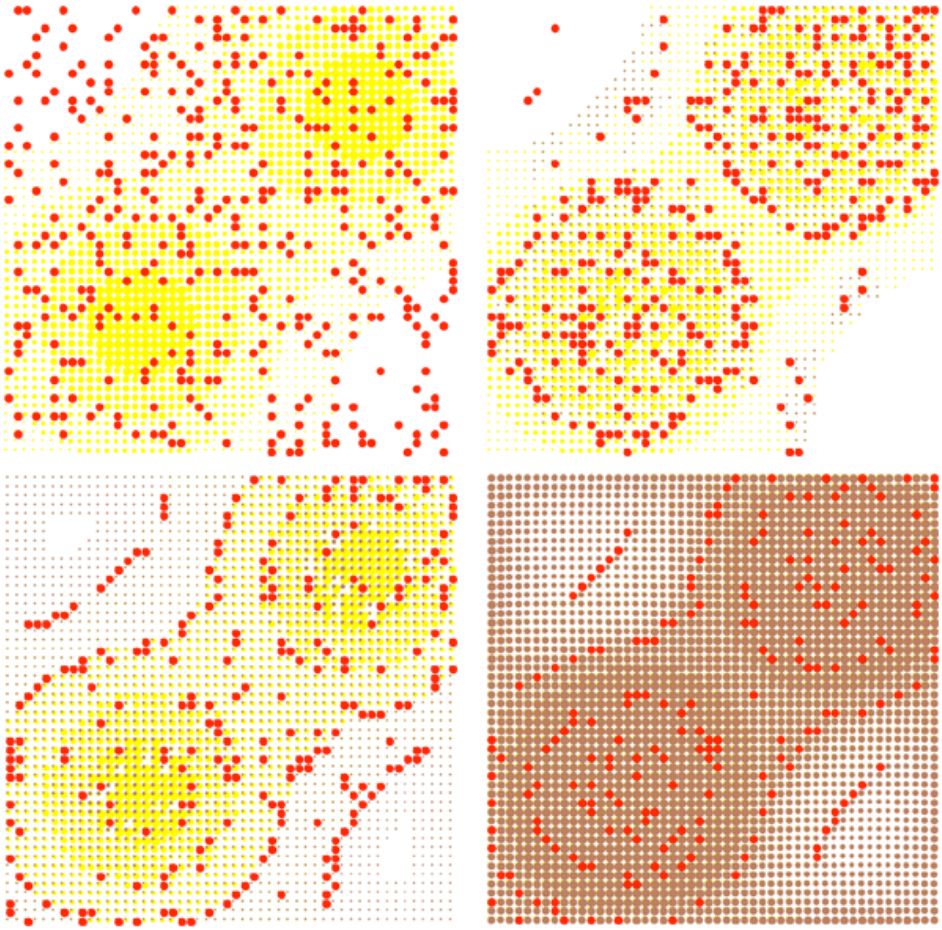


Figure 12. Pollution generation and pollution diffusion effects from Animation II-8

Figure III-1

This figure demonstrates a roughly stable population over a significant time period – 2500 time steps when the **S** is introduced. Agents are not replaced when they die, but have an opportunity during part of their lives to reproduce if they have enough resources. Figure III-1 depicts a graph of agent population over time using the default fertility parameters on page 57. Figure 13 shows how this stable population is also achieved and generally matches the graph in Figure III-1.

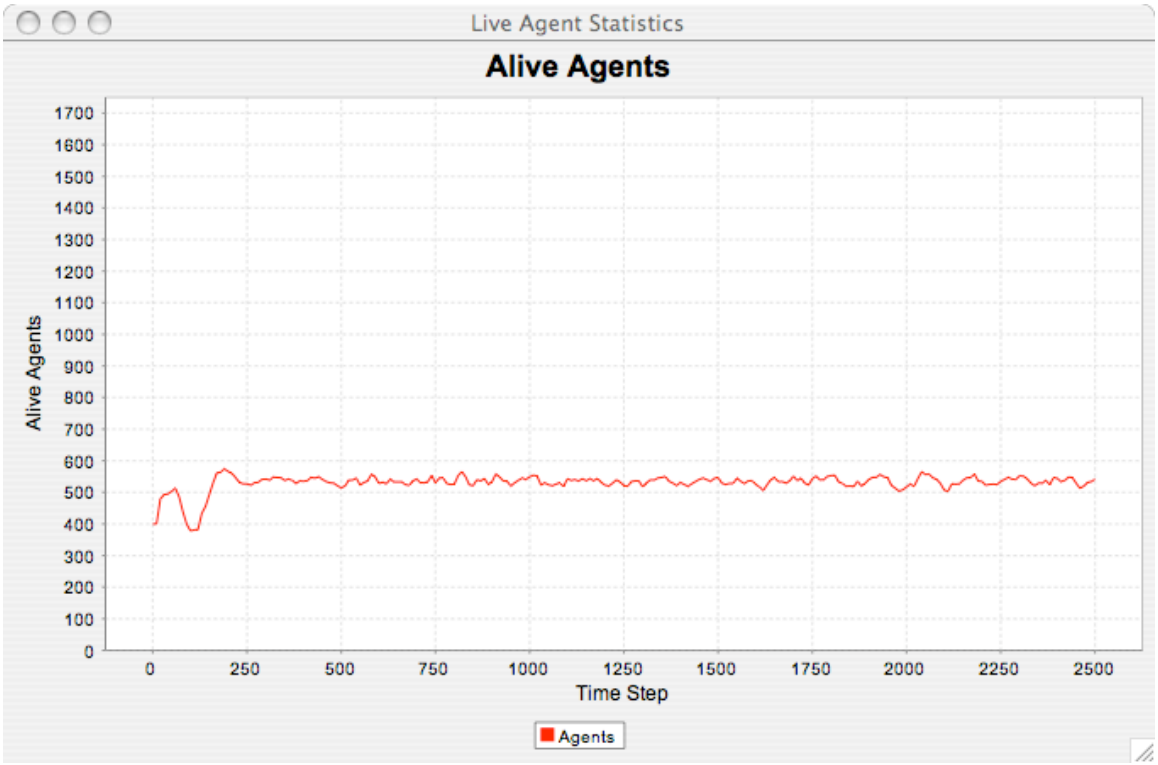


Figure 13. Agent population over time generated via Figure III-1 simulation outcome

Animation III-1

This animation shows the evolution of age distribution over time using age bins of 10 years, up to the maximum possible age of 100. The outcome of this animation is simply an approximately stationary distribution. Figure 14, representing distributions at time steps 100 and 500, generally replicates this outcome. Although child gender and other gender-based social phenomena are not an explicit part of the model, an interesting extension would be to incorporate a simple behavioral rule and graph population pyramids used by demographers.

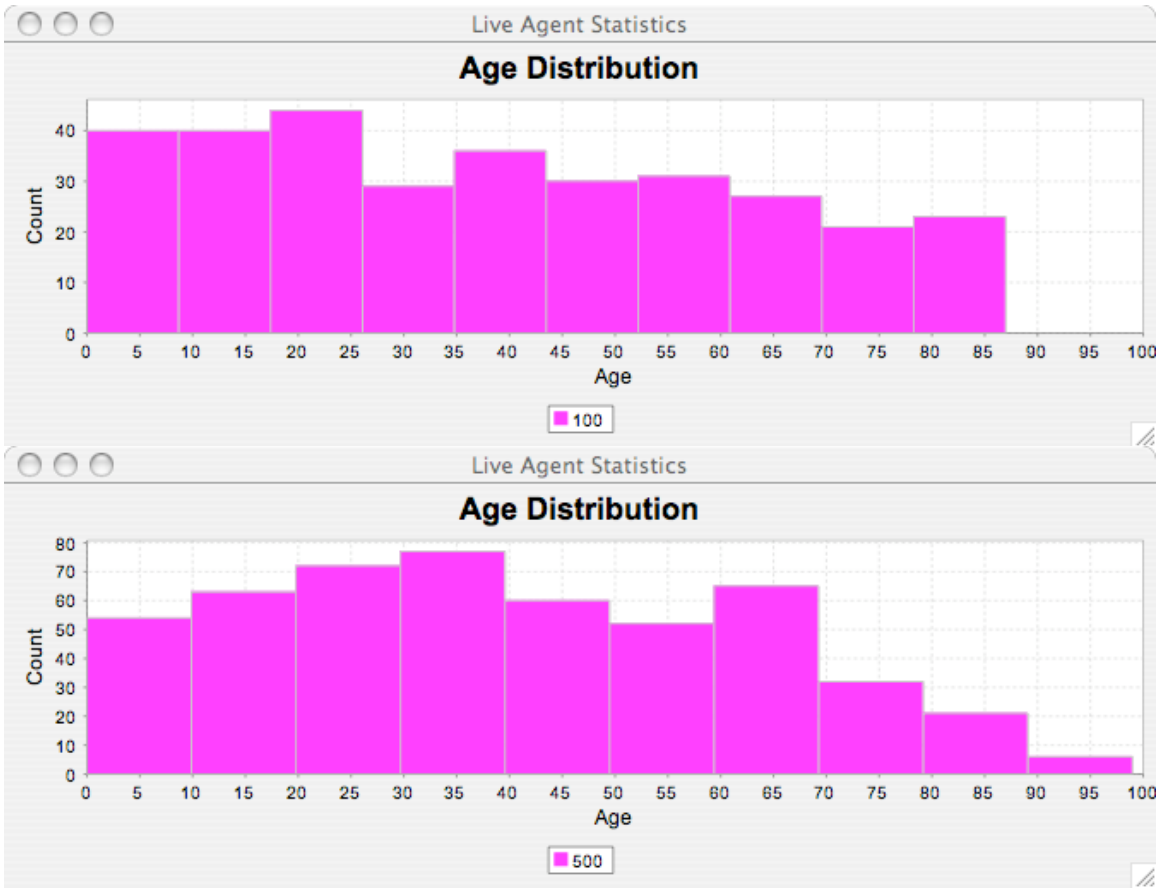


Figure 14. Age distribution at time 100 and time 500 generated by Animation III-1.

Figure III-2

The outcome in this figure is that mean agent vision and mean agent metabolism diverge, with metabolism converging on 1.0 and mean agent vision slowly increasing beyond 5.0 after approximately 900 time steps. Figure 15 generally replicates the shape and slopes of both lines.

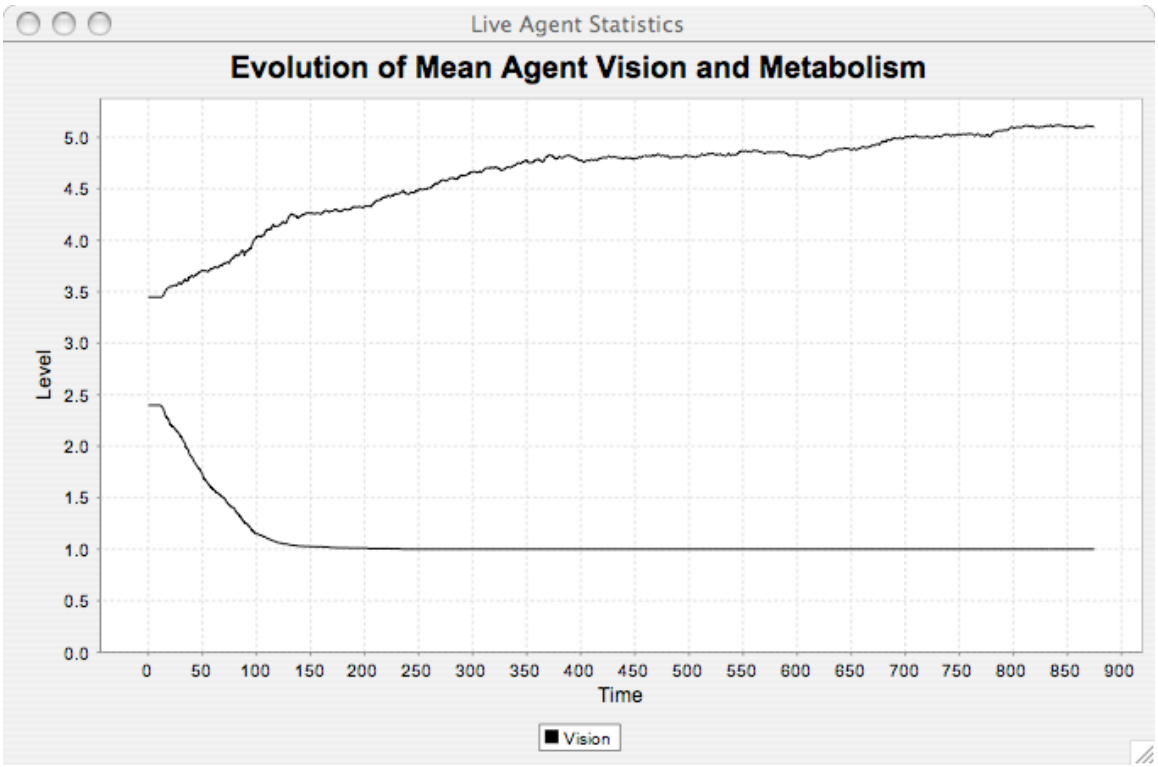


Figure 15. Mean agent vision (top line) and metabolism (bottom line) over time from Figure III-2

Figure III-3

This outcome is another experiment with the effects of **Sex**. In this simulation, female fertility ends ten years earlier (between 30 and 40), and male fertility ends ten years earlier than in Figure III-1. The outcome is fairly regular oscillations, population never falling below 250, and periods of approximately 200 years after 2500 time steps. Figure 16 indicates that these features are generally replicated in MASON Sugarscape. One difference is the significantly higher average or equilibrium population size, and the oscillations are slightly less regular than the classic model. Analysis was unable to determine the reasons for these differences.

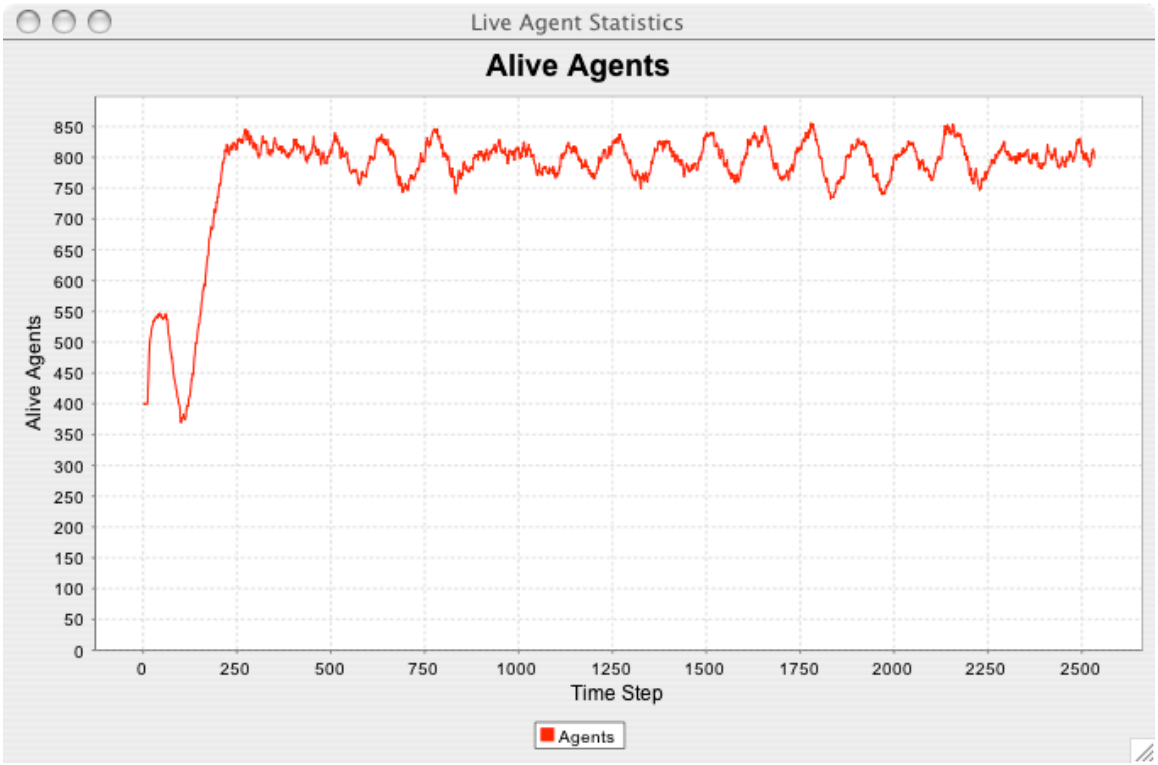


Figure 16. Alive agent population over time from Figure III-3

Figure III-4

This figure is another variation on Figure III-1. Fertility start and stop times are restored to their default values, but the amount of sugar required to reproduce is not in the range of 10 to 40. The major outcome are large, short period oscillations. The amplitudes begin at over 1000 and eventually shrink to approximately 800 after 2500 time steps; oscillation frequency is approximately 125 years.

One potential reason for differences – such as the oscillation periods not appearing to diminish – is that the sugar fertility requirements are implemented as a point value in MASON Sugarscape and probably as a uniform distribution in the classic model. For the run shown in Figure 17, a value of 20 was used.

Figure 17. Significant agent population oscillations from Figure III-4

Animation III-6

This outcome demonstrates random variability in cultural dynamics via **K** and convergence to either 'group' via a majority of tags having values of one or zero. Convergence to either color, longer convergence times for greater numbers of tags, and deep excursions into cultural groups that are ultimately not the convergence group are outcomes of Animation III-6 and Figure III-8. Both of these are achieved in the Figure 19 and Figure 18 below and in multiple simulation runs using MASON Sugarscape. The five panels in show time steps 0, 60, 108, 360, and 575. These correlate with the line graph in Figure 19 and one can see that, even with approximately 80% of the landscape covered by one group at approximately time step 130, eventually the other group dominates. Figure 20 depicts the other possible type of outcome where each sugar mountain has converged to a different color, with the random walk peaking as high as 90% blue before converging.

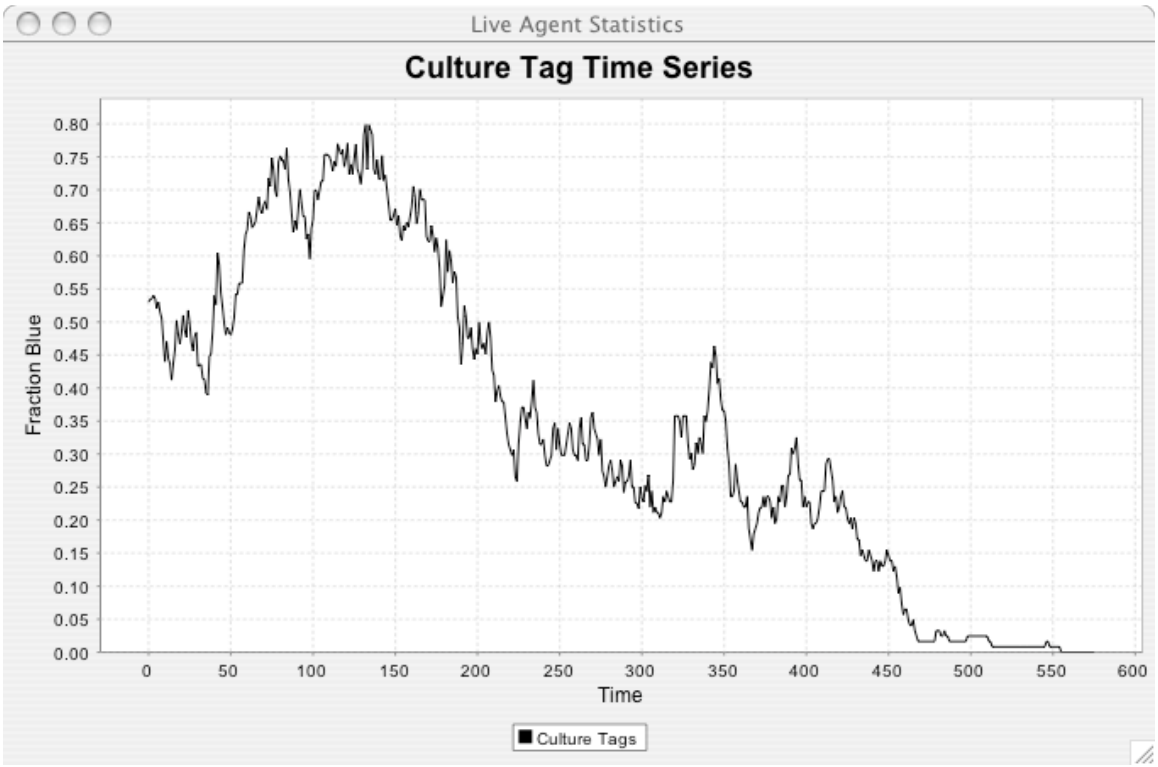


Figure 18. Convergence of culture under rule K to all red

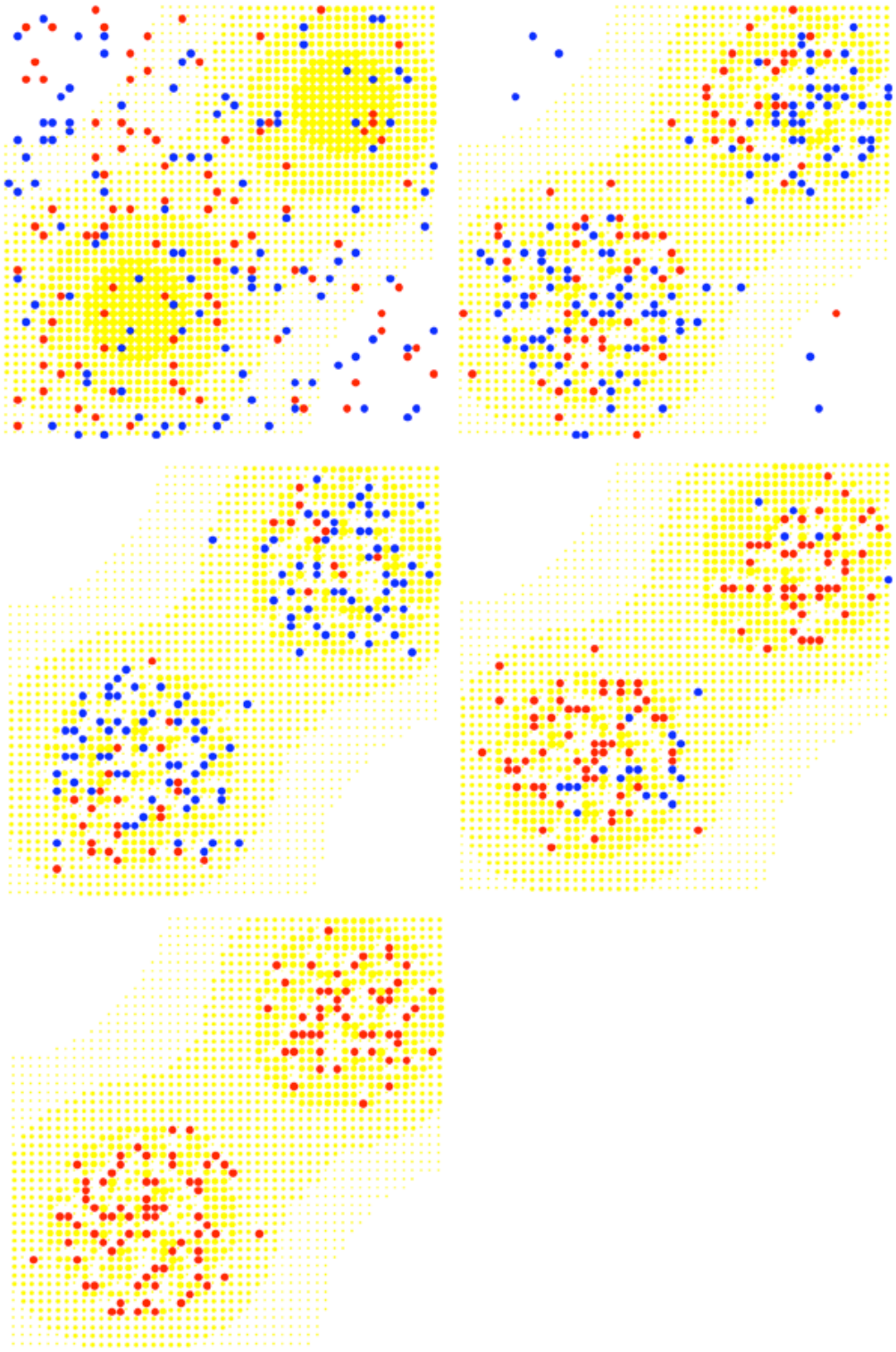


Figure 19. Convergence of culture under K

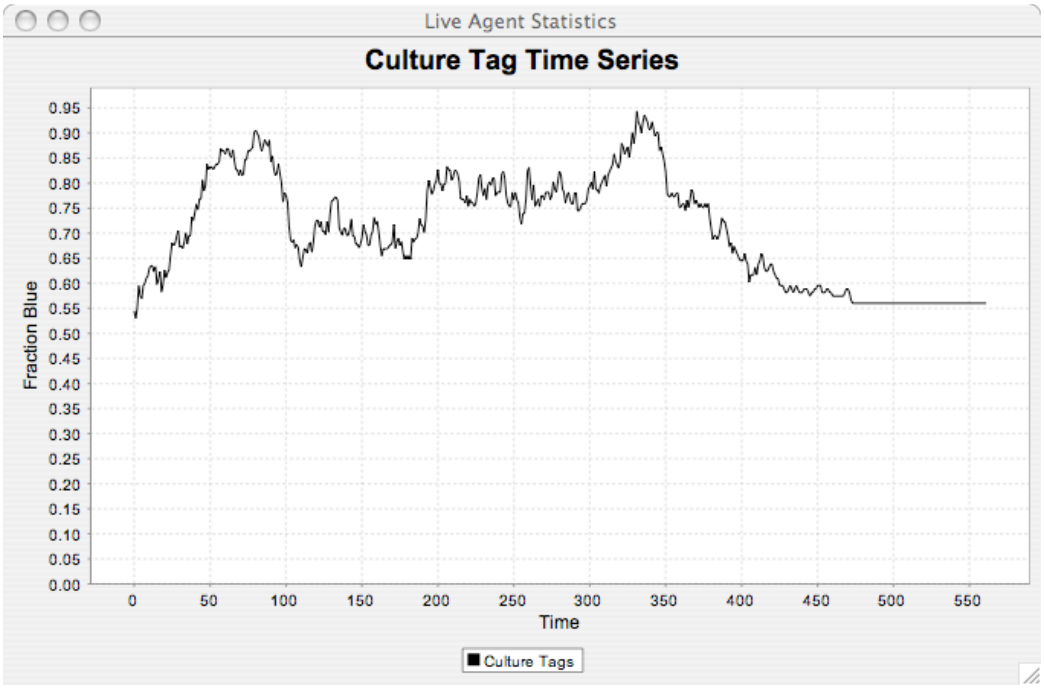


Figure 20. Culture tag time series from Animation III-6

Animation IV-1

Trade is one of the most complex rules in the model. A necessary element of the Trade rule is having two resources on the landscape, and agents calculating potential welfare functions that take into account differential metabolic rates and current holdings. The outcome of Animation IV-1 depicts a small population of agents harvesting both sugar and spice. Some agents with high vision are able to move from sugar to spice mountains, and this is achieved in MASON Sugarscape and depicted in Figure 21. One agent is marked with an 'M'. This agent has vision 9 and is able to move back and forth between several peaks to keep its sugar and spice wealth balanced as necessary. Panels shown in Figure 21 are time steps 0, 1, 8, 32, and 65.

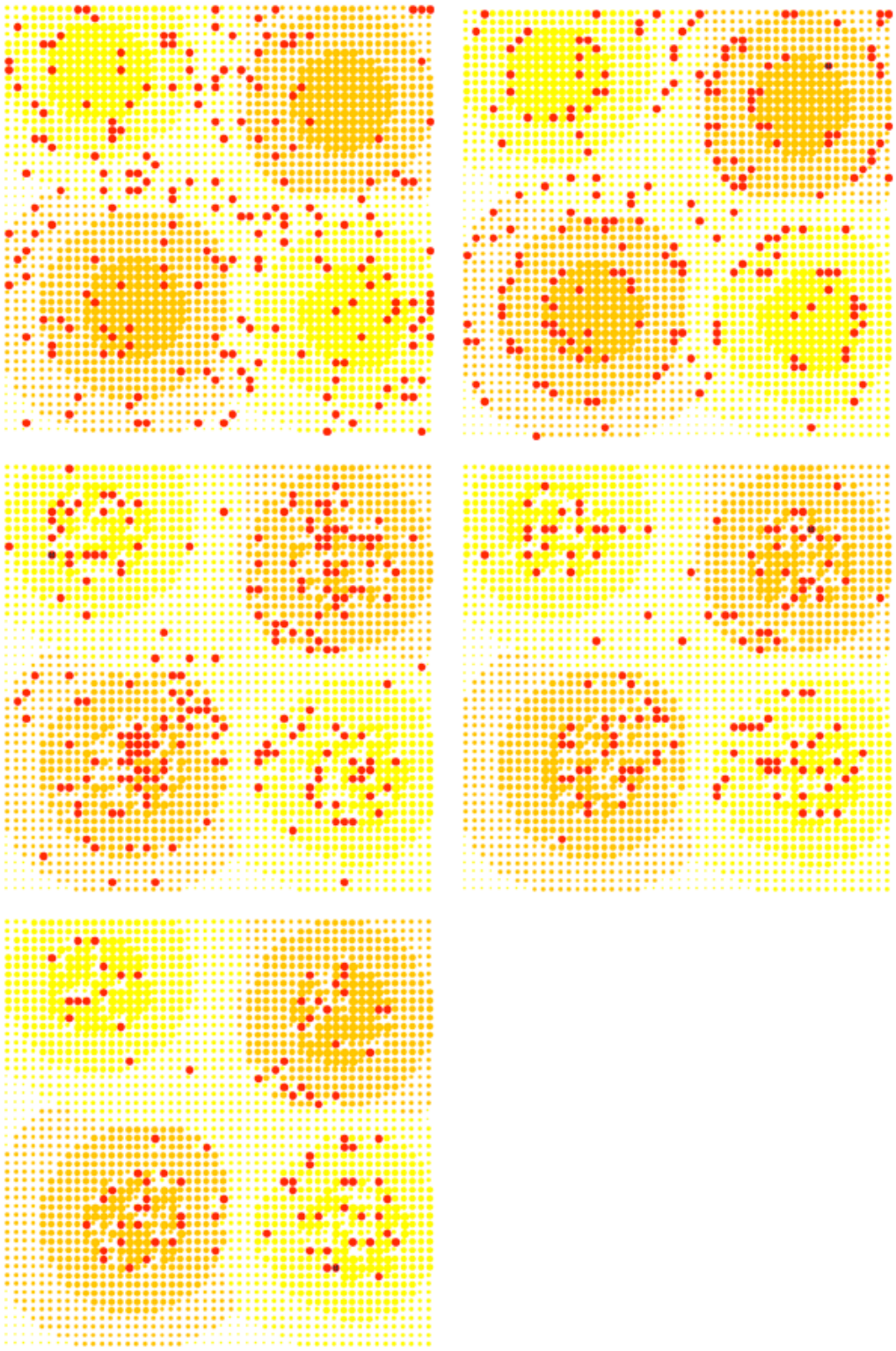


Figure 21. Pollution generation and pollution diffusion effects from Animation IV-1

Figure IV-4

When **Trade** is turned on, average prices and trade volume per time step fluctuate. A major outcome of Figure IV-4 (actually depicted in Figure IV-6) is that there is enough trade to significantly increase carrying capacity. MASON Sugarscape is able to partially replicate this outcome in that **T** does increase carrying capacity, but only by a small amount. Additionally, in either the trade or no trade conditions, carrying capacity is less than the classic model. Since carrying capacity is less in the no trade condition, this is clear evidence that the **M** and 'multi-commodity' code (`WelfareEstimation`) have different or flawed implementations. With a maximum vision of 6, only 20 more agents survive after 500 time steps than without trade in this implementation – in the classic model, it is about 40 more. The total carrying capacities with and without trades in the classic model are also significantly higher. In Figure 22 below, trade volume per time step appears correlated with population levels, so it is not surprising that with a lower population level, fewer trades occur in this implementation than as depicted in Figure IV-4.

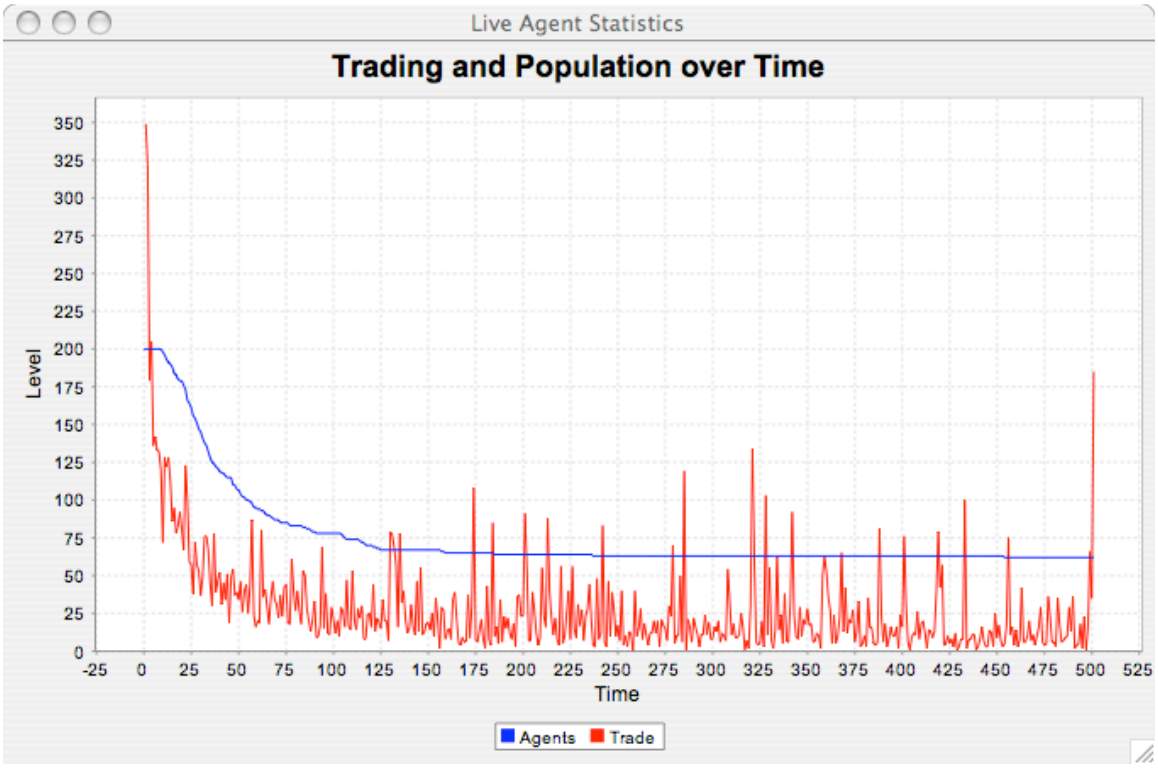


Figure 22. Trading and population over time

Results Summary

A majority of rules described in *GAS* were implemented using MASON. In general, with some notable exceptions, MASON Sugarscape simulation outcomes replicated those in *GAS*. For some of the population size based outcomes, replication results were significantly different. New infrastructure was developed to manage simulation runs and parameter sweeps and for diagnosing simulation behavior.

4. Discussion

This section assesses MASON's maturity, overall pattern of replication outcomes, forwards some implications for the methodology of agent-based modeling in social science, and outlines potential future research directions.

MASON Maturity

A primary conclusion is that MASON is mature enough to implement the Sugarscape model and potentially other classic agent-based models. It provides scheduling facilities, spatial data structures, utility classes, visualization classes, and enough Javadoc and examples documentation to enable MASON novices to make reasonable progress. Its designers have stated that MASON is designed to be used by a "experienced Java developer" as I consider myself to be. I also had direct access to both the primary MASON designer/developer, which offset unfamiliarity with MASON capabilities and lack of optimum usage knowledge. Reasonable Java competence cannot be developed in a year, and competence becomes especially important when making code changes for performance or when minor code changes have dramatic effects. The latter occurs often in agent-based models where emergence is often not expected or in exploratory research. The role of software development an in model design and implementation is a theme which will be addressed in the remaining sections as well.

Replication Results

The overall pattern of replication 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.

As an example of another effort that attempted to replicate elements of Sugarscape, Figure 23 below shows two panels from a NetLogo implementation of **G**, **M** with a starting block configuration like Animation II-6 (Densmore, 2005). Figure 23 shows that a wave is emerging at time step 2, but at time step 5 it has dissipated and for the rest of the simulation most agents remain close to the mound rather than migrating in waves to the other mound. My replication of this outcome is nearly identical unless adding the *GAS* unspecified harvest grow back constraint described in sections 3.2.1 and 3.5.

The two most complex rules – **Trade** and **Sex** – depend on having sufficient population size to have significant effect, and outcomes which depend on these two rules proved to be the most difficult to replicate. Debugging and testing of these rules led to **Movement** and `WelfareEstimation` code sections being rewritten and scrutinized many times. It is likely that an assumption in *GAS* was misinterpreted or an

implementation error occurred because numerous errors during the development process were self-induced. Table 4 below summarizes implementation and simulation errors that I encountered throughout the research: Frequencies of each type were not recorded.

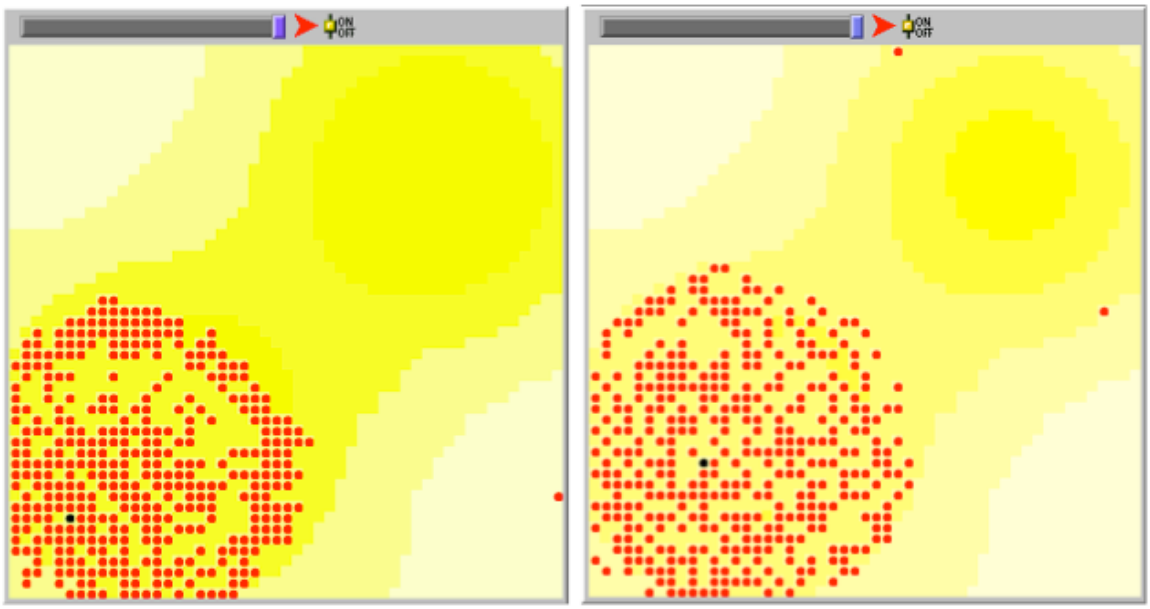


Figure 23. NetLogo Sugarscape screenshots at time steps 2 and 5 showing emergence and breakup of wave and halt of migration (Densmore, 2005)

Table 4. Classification of errors encountered

Error or Problem Type	Description
Timing	Not using the schedule in such a way that direct errors occurred (exceptions) or incorrect simulation outcomes occurred
Not fully randomizing a key behavior aspect	For behavior aspects that matter, such as which site to choose if equidistant sites have equal resources, non-random selection can generate spurious behavior
Incorrect simulation parameter	Forgetting to change or misspecifying a simulation parameter
Unknown simulation parameter	For certain outcomes in *GAS*, initial parameters are not directly specified
Incomplete initial class or method verification	Assuming that preliminary, face validity tests confirm correct implementation of a new class or method
Incomplete verification after class or method redesign	Assuming that preliminary, face validity tests confirm correct implementation a redesigned class or method

Implications and Future Research

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 eliminate 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 developing agent-based models.

Standardized notation for OO approaches such as UML diagrams can help specify aspects of model design and behavior. These diagrams should help researchers understand their creations more clearly and well as serve as means to more effectively communicate with colleagues. UML diagrams are not a substitute, however, for deep domain knowledge, software engineering experience, nor the significant amount of creativity required in generating new models and replicating existing ones.

Clearly more experience software developers will be more likely to produce correct code more quickly and develop more evolvable code. The ideal agent based modeler, assuming a single-person approach, will have significant development experience and deep domain knowledge regarding the phenomena of interest. It is probably not reasonable to expect significant numbers of researchers with these qualifications.

A significant aspect of this research concerned the visual presentations and clear communication style of *GAS*. With the authors stated focus on using the most simple rules possible, careful use of technical terms, and vivid graphics and diagrams, it was easy to form an impression that developing or replicating Sugarscape is a straightforward task. No examples were found via internet searches of fully successful replications of chapter 3 and 4 outcomes.

Although some exploration and excursions are described, the authors did not spend much time, although understandable, describing dead ends, and techniques they used to search the rules design space. Searching the space during exploratory modeling is . In the case of *GAS*, the authors' command of social sciences literature clearly influenced model evolution. Design is a creative process, and 'searching' high dimensional spaces will remain an art, though perhaps moving from different regions of these spaces can be made less laborious, quicker, and less prone to error by carefully derived techniques and tools. Clancy et al. (1997) have created a set of techniques for revising qualitative simulation models of systems with discrete structural components that might be adapted. The cognitive science literature on diagnosis of complex systems may be another source of techniques or guidance.

Another barrier to successful implementation was the use of the animations, both statically presented in *GAS* and dynamically during MASON simulation runs, to understand local and global behavior. The power of the human visual system is that it can quickly identify patterns and detect some kinds of changes over time. The visual system is biased in certain ways, and these biases can be reinforced by the way data is presented. In this case, both the static animation panels and MASON visualizations portray the end of each time step, but the relationship between the last time step and the completed one as visualized is not direct. For example, one cannot ascertain the potential trading partners of a particular agent by looking at the neighbors displayed for the most recently completed time step. Any of those neighbors might have executed after the target agent, or there might have been agents who were neighbors, but moved away before the target agent executed. While this is not difficult to understand, the influence of the current state as graphically depicted does lead to incorrect inferences at times.

Before publishing *GAS*, Axtell and Epstein published a short paper entitled "Agent-Based Modeling: Understanding Our Creations," the theme of which is summarized at its beginning –

> 'if we cannot understand these artificial complex systems any better than we understand the real ones then we haven't made progress.' (Axtell and Epstein, 1994, p. 28)

In this paper, they offer a four-level model linking performance measures and analysis requirements. This model is not explicitly discussed in *GAS*, though it appears to be the core of their verification and validation methodology. In this paper, Axtell and Epstein propose two specific techniques for understand agent-based model outcomes: logging of all agent and entity internal states at every time step to create a large database, and developing special agents that are actually data gathers and anomalous state reporters. A preliminary attempt was made to implement the first technique via the new logging infrastructure, and manual 'marking' and diagnostic mode output of one to three

agents during simulation execution was a crude attempt at the latter. Although not implemented, the ability for a user to visually 'lasso' a set of objects by drawing an irregular boundary, and have the system monitor and present their states, would be useful as well.

Progress has been made in the last 20-30 years to thoroughly understand computational behavior and artifacts in 1-D and 2-D cellular automata. Huberman and Hogg (1988) promote the study of 'ecology of computation' and have identified many different behavioral regimes for populations involving local behavior and global effects. Is it possible to exploit effectively exploit this knowledge and knowledge of asynchronous computational effects to better understand why simulation outcomes occur given model design and parameters? Can we use ecology of computation results for difference and differential equations approaches to better predict, understand, and verify agent-based models and simulations?

Edmonds and Hales (2003) recently addressed the significance and recommended approaches to replicating agent-based model simulations. They attempted to replicate Riolo's model of cooperation evolution via tags involving 100 agents over 30000 generations. In addition to strong claims about invalidating some of the author's results, they provide some heuristics and guidelines regarding replication, including that there should be a publication norm regarding results – "namely that the description of the simulation should be sufficient for others to be able to replicate (i.e. re-implement) the simulation." Edmonds and Hales also recommending replicating a simulation model using two different languages and two different implementers, though it is unclear why different languages should be a primary technique in replication. They suggest using 'unseen' or new parameter settings and running simulations for each implementation using these new parameter sets to provide an additional verification. Other heuristics involve the use of statistical tests such as the Kolmogorov-Smirnov test to verify, or at least attempt to disconfirm, a hypothesis regarding replication of outcomes that come from the same distribution (Edmonds and Hales, 2003).

The list below provides a summary of lessons learned during development and replication of MASON Sugarscape:

1. Simple models do not necessarily result in simple code.
2. It is often difficult to predict what computational behavior and artifacts will emerge in simulations and/or to determine cause and effect.
3. Very small changes to core rules can have dramatic, ripple effects across simulation outcomes, particularly where other rules interact with the changed rule.
4. It can be difficult diagnose one's code and/or infer correct behavior from pseudocode, screenshots, and a few aggregate statistics.
5. How rules and models were evolved and generated may not be described.
6. The architecture used for rules, action, and sequence may evolve significantly.

7. Any one of: lack of domain knowledge, lack of software language experience, or lack of knowledge regarding the simulation toolkit in use can dramatically affect both development efficiency and 'accuracy.'
8. It is easy to assume that even relatively simple code is correct simply by viewing graphical behavior output.
9. Discrete time step based visualizations of agent state and space (model state) can be deceiving. Vivid graphical portrayal of models can lead to a self-deceptive belief that implementing and/or replication will be easy.
10. Distributing simulations with different parameter sets across computing resources can be cumbersome.

Potential heuristics for addressing these issues are as follows:

1. Source lines of code counts and other software complexity metrics for original work should be published so that correlations can be made between software metrics and rules to better understand their scope and complexity.
2. Literature and results from 1D and 2D cellular automata and ecology of computation fields should be mined and distilled to better differentiate results from artifacts and to help understand models. Methods from qualitative simulation should be explored and integrated to help improve the search of model space.
3. Emphasis on the correctness of the core rules is paramount.
4. Researchers should consider archiving and publishing detailed simulation output – object states, a variety of aggregate measures, and perhaps fractal dimension measures for every significant outcome. This would help with both code verification diagnosis and model replication.
5. Researchers should attempt to document how the model design space was searched.
6. #3 notwithstanding, implementation of complex rules – rules interacting with other rules should not be deferred for too long due design impacts. A balance is necessary.
7. Researchers will have to spend significant time in multiple disciplines to become effective.
8. Unit testing methods ought to be used as frequently as possible. Unit tests should not simply be conducted for single classes and methods, but for combinations of classes/rules.
9. Sequence diagrams, space-time diagrams, and other debiasing and cognitive aids should be used.

10. One approach to grid computing, distributing parameter sets across nodes rather than distributing elements of a single simulation, ought to be considered for inclusion in future agent-based modeling and simulation toolkits.

5. Summary

In support of a broader objective of advancing and examining the practice of agent-based modeling for social science, this research had two primary goals: to demonstrate the suitability of MASON to replicate a classic agent-based model – Sugarscape; and to demonstrate that general replication of Sugarscape is possible. Sugarscape consists of a set of very simple rules – some rules are not simple to implement – yet the resulting dynamics in simulation produce complex, emergent phenomena. General replication was achieved, although significant discrepancies remain unresolved, particularly for population-size based outcomes. MASON was mature enough and provided core capabilities to replicate Sugarscape; it was extended in a few ways to enhance the process of simulation testing, such as automated parameter sweeping infrastructure and automated logging. During the process of replication, obstacles emerged such as numerous researcher induced errors and the difficulty of understanding the relationship between source code and emergent behavior. Reducing these barriers for social scientists through better tools and techniques is a significant challenge.

REFERENCES

REFERENCES

Ajzen, I. (1988). Attitudes, personality, and behavior. Milton-Keynes, England: Open University Press.

Archer, M. (1990). Human Agency and Social Structure: A Critique of Giddens. In J. Clark, C. Modgil, & S. Modgil (Eds.), Anthony Giddens: Consensus and Controversy (pp. 73-84). London: The Falmer Press.

Brajnik, G. and Lines, M. (1998, January). "Qualitative modeling and simulation of socio- economic phenomena." Journal of Artificial Societies and Social Simulation, 1. Retrieved August 20, 2004, from http://www.soc.surrey.ac.uk/JASSS/1/1/2.html.

Clancy, D. , Brajnik, G., and Kay, H. 1997. Model Revision: Techniques and tools for analyzing simulation results and revising qualitative models." In 11th International Workshop on Qualitative Reasoning, Cortona, Siena Italy; June 1997, pp. 53-66. Retrieved August 20, 2004, from http://www.dimi.uniud.it/~giorgio/papers/qr97-mr.ps.gz

Cioffi-Revilla, C. (2002). Invariance and Universality in Social Agent-based Simulations. *Proceedings of the National Academy of Science*, 99, 7314-7316.

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

Doran, J. (2000a) Trajectories to Complexity in Artificial Societies. In T. Kohler & G. Gumerman (Eds.), Dynamics in Human and Primate Societies: Agent-based Modeling of Social and Spatial Processes. New York: Oxford University Press.

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

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

Festinger, L. (1957). *A Theory of Cognitive Dissonance*. Stanford, CA: Stanford University Press.

Giddens, A. (1979). Central Problems in Social Theory. Berkeley, CA: University of California Press.

Gilbert, N. & Doran, J. (Eds.). (1994), *Simulating Societies. The Computer Simulation of Social Phenomena*. London: UCL Press.

Gizzi, M., Lairson, T., & Vail, R. (2003, October). Modifications to a original model created by: Wilensky, U. (1998). Mesa State College, Center for Agent-Based Modeling. Retrieved August 10, 2004, from http://www.modelingcomplexity.org.

Gigerenzer, G., Todd, P., & the ABC Research Group. (1999). *Simple Heuristics That Make Us Smart*. New York: Oxford University Press.

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

Holland, J. (1995). Hidden Order: How Adaptation Builds Complexity. Reading, MA: Addison Wesley Publishing Company, Inc.

Holland, J. (1998). *Emergence: From Chaos to Order*. Reading, MA: Perseus Books.

Jager, Wander. (2000). Modelling Consumer Behavior. Unpublished Ph.D dissertation. Retrieved August 1, 2004, from http://docserver.ub.rug.nl/eldoc/dis/ppsw/w.jager/

Huang, C., Sun, C., Hsieh, J., & Lin, H. (2005). Simulating SARS: Small-World Epidemiological Modeling and and Public Health Policy Assessments. *Journal of Artificial Societies and Social Simulation*, 7, Retrieved April 1, 2005, from http://jasss.soc.surrey.ac.uk/7/4/2.html

Huberman, B., and 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*. Hoboken, NJ: John Wiley and Sons, Inc.

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*. Dordrecht: Kluwer.

Lewis, R. (1999). Cognitive Theory: Soar. Draft manuscript. Retrieved September 26, 2004, from http://en.wikipedia.org/wiki/SOAR

Liverani, Mario, and Domenico Parisi. (1998). Assiri virtuali: un laboratorio didattico. *Iter* 1:27–38.

Luke, S., Cioffi-Revilla, C., Panait, L., & Sullivan, K. (2004). MASON: A New Multi-Agent Simulation Toolkit. Retrieved September 9, 2004, from http://cs.gmu.edu/~eclab/projects/mason/publications/SwarmFest04.pdf

Luke, S., Cioffi-Revilla, C., Panait, Liviut, Sullivan, K., and Balan, G. (2005) MASON: A Multi-Agent Simulation Environment. *Simulation*. To be published.

Malkov, Artemy S. (2004). Spatial Modeling of Historical Dynamics. In *Proceedings of the International Conference on Mathematical Modelling of Social and Economical Dynamics, June 23–35, 2004*. Moscow, Russia.

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*. Dordrecht: Kluwer.

Parisi, Domenico. (1998). A cellular automata model of the expansion of the Assyrian empire. In *Cellular Automata: Research Towards Industry*, edited by S. Bandini, R. Serra and F. S. Liverani. London: Springer.

Ritzer, G. (2000). Modern Sociological Theory (5th ed.). Boston: McGraw-Hill.

United Nations Development Programme. (2004). Technical Note 1: Calculating the Human Development Indeces. Retrieved September 18, 2004, from http://hdr.undp.org/docs/statistics/indices/technote_1.pdf

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

Wilensky, U. (1998). NetLogo Wealth Distribution model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Retrieved August 10, 2004, from http://ccl.northwestern.edu/netlogo/models/WealthDistribution.

Wright, Henry T. (2000). Agent-Based Modeling of Small-Scale Societies. In T. Kohler & G. Gumerman (Eds.), *Dynamics in Human and Primate Societies: Agent-based Modeling of Social and Spatial Processes*. New York: Oxford University Press.

APPENDIX 1: Glossary of Terms

Adaptive Toolbox – "a collection of specialized cognitive mechanisms that evolution has built into the human mind for specific domains of inference and reasoning, including fast and frugal heuristics." (Gigerenzer, Todd, and the ABC Research Group, 1999, p. 30)

Agency – A term with many definitions and viewpoints in sociological theory. Describes attributes or phenomena of agents, typically individual human actors (Ritzer, 2000).

Agent-based model – A simulation model having an environment, individuals or agents which act in that environment using environment information – including information about other agents, and take actions in that environment via internal rules. (Gilbert, 2000, p. 9; Epstein and Axtell, 1996, p. 6). Agent-based models may be used to refine theory or explore specific aspects of a real-world social context.

Artificial Society – A context-free computational model involving agents, environments, and rules. Model behavior, dynamics, and patterns are the focus, rather than validation of the model against a particular real-world society. (Epstein and Axtell, 1996; Doran, 2000b) The defining aspect of an artificial society is "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." (Epstein and Axtell, 1996, p. 6)

Cognitive Dissonance – A psychological phenomenon resulting conflicting knowledge elements held by a person that "exerts pressure in the direction of bringing the appropriate elements into correspondence with that reality." (Festinger, 1957, p. 11)

Complex Adaptive System – A system composed of interacting agents that change their rules as experience accumulates; agents adapting to other agents generate complex temporal patterns (emergence) (Holland, 1995, p. 10).

Docking – A synonym for replication in which different computational implementations of a model are 'aligned', replicated, or validated against one another such that each implementation generates the same results.

Emergence – A phenomenon regarding systems that are generated, in which the wholes are greater than the sum of their parts (Holland, 1998, p. 225).

Gini Coefficient – A measure of income inequality defined as "twice the area between the Lorenz curve and the 'equality line'" (Kleiber and Kotz, 2003, p. 30)

Heuristic – A rule, or rule of thumb, used to inferencing behavior, such as searching, stopping, and decision – making. (Gigerenzer, Todd, and the ABC Research Group, 1999, p. 143-144)

Institution – "The most deeply-layered practices constitutive of social systems … are institutions…[they are] standardized modes of behavior which play a basic part in the space-time constitution of social systems." (Giddens 1979, 65)

Lorenz Curve – A graphical representation of income inequality where perfect equality is a 45 degree diagonal line, and increasing inequality is presented as more concave curves below the diagonal. (Wikipedia, 2004)

New Wealth Distribution Model – A modified version of the wealth distribution model in Sugarscape, in which the social context is transformed from "a basic hunter-gatherer society to an agrarian society complete with entrepreneurship, employment, population-growth, and inheritance of wealth." (Gizzi, Lairson, and Vail, 2003)

Order – A MASON construct in the `Schedule` class for specifying an exact order of execution between different groups of `Steppable` objects.

Sugarscape – An agent-based model developed by J. Epstein and R. Axtell to "begin the development of a more unified social science, one that embeds evolutionary processes in a computational environment that simulates demographics, the transmission of culture, conflict, economics, disease, the emergence of groups, and agent coadaptation with an environment, all from the bottom up." (Epstein and Axtell, 1996, p. 19)

APPENDIX 2: Pseudocode for Rules

G

```
for each site in scape_grid
       if  (regen_time >= seasonal_rate and current_level <   max_capacity)
      current_level = current_level + 1
      regen_time = 1
   else regen_time = regen_time + 1
```

M

```
distance = 1
while distance <= vision
   scan randomly in 4 directions for unoccupied sites at distance
   add unoccupied sites to unoccupied_sites_list
add current location as last site on unoccupied_sites_list
best_level = 0
best_dist = 9999
for each site on the unoccupied_sites_list
   level = sugar at site
   level = level / site_pollution
   if resource at that site is >= best_level
      if level > best_level
         best_site = current_site
         best_distance = current_distance
      else if distance to site < best_distance
         best_site = current_site
         best_distance = current_distance
      else if distance = best_distance
         add current site to list of best_sites
if unoccupied_sites_list size > 1
    randomly select best_site from unoccupied_sites_list
if best_site !=null
    move to best site
    wealth = wealth + site_resource_level
    site_resource_level = 0
```

B

wealth_sugar = wealth_sugar – metabolic_rate_sugar
age = age + 1
*total_pollution = metabolic rate * pollution_metabolic_rate*
*total_pollution = total_poluation * pollution_harvest_rate*
if (wealth_sugar < 1) or (my_age > my_max_age)
    alive = false
    remove agent from agent_grid
    remove agent from schedule

$R_{[a,b]}$

if (dead and replacement == true)
    create new agent
    add new agent to schedule to begin execution at t+1
    find first random unoccupied site
    add agent to agents_grid at unoccupied site

$S_{\alpha\beta\gamma}$

if (north_season == SUMMER)
    north_season = WINTER
    south_season = SUMMER
else
    north_season = SUMMER
    south_season = WINTER

$P_{\pi,x}$

get scape at current location
add generated pollution to scape current pollution

D_{α}

for each site
        new_pollution_temp[site] = average of current pollution levels at site's 4 way
        neighbors
for each site
        current pollution[site] = new_pollution_temp[site]

S

```
if (age > fertility_end) or (age < fertility_start) or (wealth  < reproduction_minimum)
  end
else
   identify all neighbors
   for each neighbor
            if (neighbor is not same sex as self) and            (neighbor_wealth >=
            initial_endowment)
               if there is an open site adjacent to self or partner
                 create a new child with cross genetic attributes
                    add the new child to the open site
                    add the child to the schedule to execute at t+1
                       subtract one half initial_endowment from wealth of   each parent
```

K

```
Randomly scan for 4-way neighbors
for each found neighbor
   pick a random position in the tag set
      if the neighbor's tag is not the same as self tag in that  position
       flip the bit
```

Multicommodity M

```
distance = 1
while distance <= vision
   scan randomly in 4 directions for unoccupied sites at distance
   add unoccupied sites to unoccupied_sites_list
add current location as last site on unoccupied_sites_list
calculate current welfare_sugar
calculate current welfare_spice
calculate projected welfare for sugar for finds of 1,2,3,4
calculate projected welfare for spice for finds of 1,2,3,4
best_level = 0
best_dist = 9999
for each site on the unoccupied_sites_list
   calculate level at current site using precalculated projected welfares
   level = level / site_pollution
   if level at that site >= best_level
      if level > best_level
         best_site = current_site
         best_distance = current_distance
```

```
      else if distance to site < best_distance
         best_site = current_site
         best_distance = current_distance
      else if distance = best_distance
         add current site to list of best_sites
if unoccupied_sites_list size > 1
    randomly select best_site from unoccupied_sites_list
if best_site !=null
    move to best site
    wealth_sugar = wealth_sugar + sugar_site_resource_lev
         wealth_spice = wealth_spice + spice_site_resource_lev
    site_resource_levels = 0
```

T

```
Randomly scan for 4-way adjacent neighbors
for each neighbor
   calculate my_MRS
   calculate neighbor_MRS
   while my_MRS != neighbor_mrs
     calculate price
          set direction of sugar trade
     set direction of spice trade
     calculate my_welfare
     calculate neighbor_welfare
     calculate my_new_welfare
     calculate neighbor_new_welfare
           if (my_new_welfare > my_welfare) and (neighbor_new_welfare >
           neighbor_welfare)
              calculate my_new_MRS
              calculate neighbor_new_MRS
                   if the sign of my_new_MRS-neighbor_new_MRS = sign of my_MRS-
                   neighbor_MRS
                      add/subtract sugar to self
                      add/subtract spice to self
                      add/subtract sugar to neighbor
                      add/subtract spice to neighbor
                   else done tradine with this neighbor
      else done trading with this neighbor
```

APPENDIX 3: Source Lines of Code per Class

Table 5. SLOC per Class

SLOC	Class
927	Sugarscape.java
508	agent.java
358	ResultsGrapher.java
249	WelfareEstimation.java
247	Statistics.java
225	ParamSweeper.java
185	Charts.java
165	SugarscapeWithUI.java
159	SugarscapeWithUIHigh.java
136	TradeTester.java
134	Scape.java
126	Histogram.java
78	Culture_K.java
56	DiffuserPollution.java
50	AgentPortrayal2D.java
45	ScapePortrayal2D.java
42	Growback.java
36	Movement.java
35	Culture.java
33	DuplicatesHashtable.java
28	RulesSequence.java
19	Biological.java
17	Trade.java
17	Replacement.java
15	Reproduction.java
15	Death.java
14	Pollution.java
14	Environment.java
7	ResourceDistPair.java
6	EnvironmentRule.java
5	Rule.java