Artificial Life Simulations: Discovering and Developing Agent-Based Models

Matthias Scheutz
Human-Robot Interaction Laboratory
Cognitive Science Program and School of Informatics
Indiana University
Bloomington, IN 47406, USA
mscheutz@indiana.edu

Introduction

Agent-based simulations have become increasingly important over the recent past in a wide variety of fields, ranging from the simulation of complex physical systems, to the modeling of different kinds of biological and social systems, to various applications in game theory and artificial intelligence. They have been used, for example, to model *bacterial chemotaxis signaling pathways* (e.g., Le Novre & Shimizu, 2001; Andrews & Bray, 2004), *population ecology* (e.g., Railsback et al., 2002; Anderson, 2002; Grimm, 1999), *social, economic, and political systems* (e.g., Conte, 2002; Schermerhorn & Scheutz, 2003), *software engineering* (e.g., Gao, Madey, & Freeh, 2005), *neural networks* (e.g., Schoenharl & Madey, 2004), *business and commerce* (e.g., Bonabeau, 2002) and many other areas. In particular, in artificial life (Alife) research, simulation environments are a critical tool for advancing knowledge and understanding of the mechanisms and principles that govern the emergence or evolution of life or like-like processes.

While artificial life and agent-based simulation environments have found most applications in research, their utility is not limited to the research domain. Rather, simulation environments can also be a useful instructional tool for students helping them to understand the mechanisms and principles at work in complex systems. For example, by being able to watch the dynamics of a system unfold graphically on the computer screen, students will be able to better understand the analytical relationships expressed in dynamics systems equations that govern the temporal evolution of the system. And by being able to modify parameters or even rules that define the behavior of simulated entities, students will be able to explore the space of possible designs and principles in an active manner.

We believe that the computational capacity of modern-day computers provides a unique resource for students and teachers alike that not only facilitates the process of learning relationships in existing systems, but also allows students as part of their discovery process to define new models via interactions with the simulation environment (e.g., Russ, 1997).

In this chapter, we present our SimWorld agent-based artificial life simulation environment (e.g., Scheutz, Schermerhorn, Connaughton, & Dingler, 2006) and demonstrate how it can be used as a tool to allow students to discover and develop models of real-world systems. Specifically, we report the results from a student project where the goal was to develop an agent-based model of the

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

phonotactic behavior of female frogs in a swamp when they are listening to the calls of male frogs and move towards them. We start with a brief high-level overview of the SimWorld environment and its various components, including a description of the programming paradigms that it affords and the standard interfaces it provides for including external open-source software components (e.g., external physics engines such as ODE (www.ode.org) or graphical visualization tools such as OGRE (www.ogre3d.org). We then introduce the particular example of modeling "female choice" in treefrogs and show how SimWorld was used to discover and develop a model. In particular, we show the steps involved in defining and running a simulation model, pointing to the support SimWorld provides for each of these steps. This also includes how simulation output can be analyzed and visualized with open-source tools outside of SimWorld such as R (www.r-project.org) or Scilab (www.scilab.org). Finally, we provide a summary of other agent-based modeling toolkits that are freely available, indicating the degree to which they are suitable for instructional purposes and comparing them to SimWorld.

The SimWorld environment

SimWorld is an agent-based artificial life simulation environment built on top of the SimAgent agent toolkit (Sloman, 1999), a general-purpose agent toolkit based on the Poplog programming environment with a built-in OPS5-style rule interpreter poprulebase . SimWorld can run in one of two main modes: (1) the "GUI mode", which provides a 2D interactive graphical user interface that displays the simulated entities in their simulated environment and allows users to inspect and manipulate simulated entities; and (2) the "batch mode", a non-interactive mode that can be used to run larger-scale experiments (e.g., simulations that attempt to evolve a particular trait of an agent).

SimWorld provides support for various different kinds of entities, such as simple reactive and more complex deliberative agents. These agents can operate in 2D or 3D environments, ranging from simple, grid-based environments (as used for many agent-based simulations where the exact spatial metric does not matter) to complex spatial environments, where distances and spatial extension are accurately modeled. New environments with different topological and metric properties can easily be defined and new agent models can be added by simply "extending" any of the base agents, which is a straightforward process within the object-oriented design of SimWorld (demonstrated in the next section).

Different from many other simulation environments, SimWorld provides extensive support for periodic or event-driven data collection. It is possible to record variables or parameters in the whole simulation environment (including SimWorld system parameters) via a simple specification language (see Step 2 in the next Section).

SimWorld has been used successfully in several diverse research projects over the last six years, ranging from the study of the utility of affect for agent control (e.g., Scheutz & Schermerhorn, 2002, to various evolutionary

investigations (e.g., Scheutz & Schermerhorn, 2005), to the study of conflict resolution strategies (e.g., Scheutz & Schermerhorn, 2004), and others (e.g., biologically plausible models of frog behaviors, Scheutz, Madey, & Boyd, 2005, foraging in hives with limited resources, Schermerhorn & Scheutz, 2005, or UAV swarms, Scheutz, Schermerhorn, & Bauer, 2005). It has also been used as an instructional tool in various courses at the University of Notre Dame, for example, to study multi-agent systems (in CSE 471/571 Artificial Intelligence) and in undergraduate design and research projects (e.g., to study the biologically inspired multi-agent foraging tasks, in CSE 499R Undergraduate Research).

Interfacing external components

SimWorld is an open environment that allows for the integration of external software components. Support for such "external plug-ins" (e.g., to link in external simulation and visualization components such as external physics and graphics engines) is provided via an open "plug-in architecture". We briefly mention two engines that have been connected to SimWorld : the Open Dynamics Engine (ODE), a physics engine used to provide efficient collision and friction detection as well as realistic rigid-body motion, and the Object-Oriented Graphics Rendering Engine (OGRE), a three-dimensional graphics rendering package. The integration of external engines is achieved via segments of shared memory (together with inter-process synchronization mechanisms) that allow SimWorld and external engines to share basic agent features. In the case of a physics engine, for example, the information passed to the engine at each cycle are force vectors representing the forces generated by the agent's controller (via its body); the physics engine then simply returns the updated position and orientation of the agent based on the newly generated force and all other forces that apply. For a graphics engine, location and orientation of each agent (plus additional information about the appearance of the agent, etc.) are shared (see Figure 2 for a comparison of SimWorld and OGRE visualization). Both physics and graphics engines can be included at the same time, and moreover, designers can selectively choose which agent to update and render via the external engines (thus allowing for non-physical information gathering agents that might not need to be displayed).

Parallelization of SimWorld

Automatic parallelization of agent-based simulations is one of the main features that distinguishes SimWorld from all other available simulation environments. Is is based on a novel algorithm that distributes simulations over a set of available hosts (Scheutz & Schermerhorn, 2006). In order to parallelize SimWorld simulations, the JAVA-based SWAGES experimentation environment is required, which manages groups of host computers that can be used for parallelization. SWAGES monitors all hosts and automatically distributes a given simulation in the best possible way based on host availability.

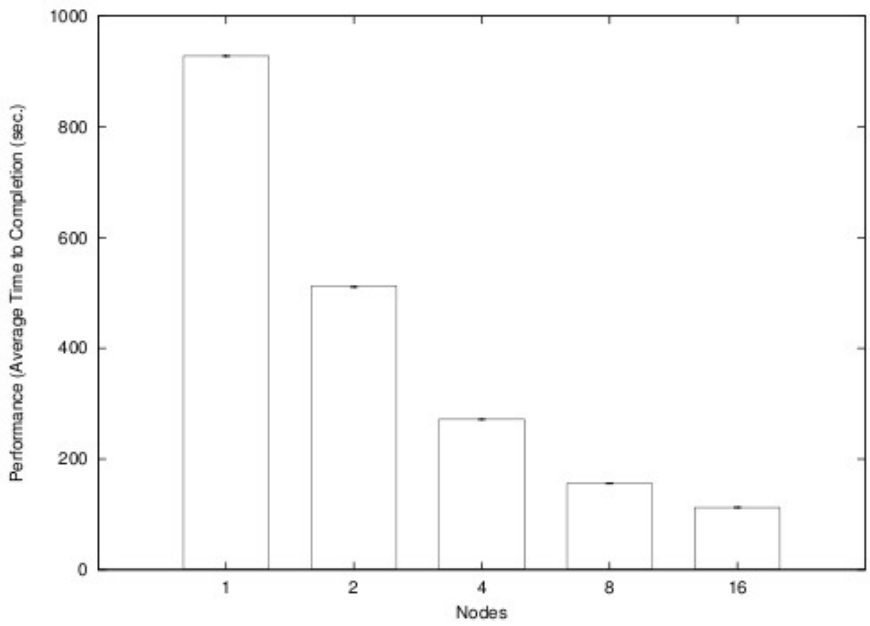


Figure 1: Average time to completion for 1, 2, 4, 8, and 16 nodes for a swarm task , where agents must find and gather at the nearest checkpoint as quickly as possible. (Small) error bars denote confidence intervals for a = 0. 05.

In SWAGES 's parallelization scheme, simulation entities are divided into groups to be simulated on separate hosts. Each host executes a SimWorld instance in which all static and lowcost entities (i.e., those whose updates are less expensive than the overhead of parallelizing them) are duplicated, and more complex entities are divided among the different hosts.

The parallelization algorithm can either run in lock-step mode (i.e., updating all parallel simulations one cycle at a time), or in asynchronous mode where individual simulations update independently for as many cycles as possible until information from other simulations is needed. The asynchronous algorithm utilizes spatial information available about the "sphere of influence" of entities in spatial agent-based models such as SWARMS, ANTS, and many others, where entities can affect their environment only within a given range.

The time-savings of the algorithm for one example SWARM simulation are shown in Figure 1 (see Scheutz & Schermerhorn, 2006 for details). Briefly, the agents in this simulation must gather at the nearest "checkpoint" in the environment as quickly as possible, with each agent executing a basic reactive architecture that causes the agent to move directly to the nearest detected checkpoint, and wander randomly when none is detected. These results demonstrate the performance of the asynchronous parallelization algorithm for cases where good splits of agents can be computed (e.g., because different groups

of swarms are located far apart, so that they cannot influence each other). The results reported are averages over 20 simulation runs of 100 cycles each. Each of the 20 initial conditions was simulated using 1, 2, 4, 8, and 16 nodes in a dedicated Linux cluster of dual 2.4GHz Xeons with 1GB RAM. The times reported include all overhead of starting and finishing SWAGES, as well as distributing the simulations when more than one node is used (for details of the algorithm and experiments, see Scheutz & Schermerhorn, 2006).

Figure 2: Simple reactive ant-like agents on their hunt for food in OGRE and in the simple 2D SimWorld GUI (brown circles with black dots) superimposed on the upper right.

SWAGES works in homogeneous fixed clusters (e.g., Beowulf clusters) and heterogeneous ad-hoc clusters (e.g., individual workstations that can only be used if nobody is logged in) alike. Moreover, it requires no setup procedures on the host participating in simulation experiments (other than the standardly installed secure shell tools for secure, remote login and file transfer) and will run on all operating systems that support the JAVA virtual machine and the Poplog environment (for SimWorld).

Using SimWorld to discover and develop agent-based models

We will now demonstrate in some detail how SimWorld can be and has been used by students to explore the space of possible designs, which in turn leads to the

discovery and development of agent-based models that illustrates principles found in natural systems. The particular example presented here is from a student project jointly supervised by the author and one of his colleagues in biology. The goal of the project was for the student to study an agent-based model of female choice in treefrogs. We begin with a brief description of the biological background and then go through the steps of the model development, including running larger-scale simulations and analyzing the resultant data.

Biological background: female choice in treefrogs

It is often assumed in studies of mate choice that females "choose" a single mate from a group of males based on some criteria. Female treefrogs, for example, show phonotaxis toward calls of males with higher pulse numbers (e.g., Schwartz, Buchanan, & Gerhardt, 2001; Schwartz, Huth, & Hutchin, 2004). Females are thus assumed to make an active choice (Gibson & Langen, 1996), show a directional bias (more pulses are better, Ryan & Keddyhector, 1992), and differentiate between individual males up to a maximum of 5 (Gerhardt, 1991; Greenfield & Rand, 2000). While there are several proposed rules for female sampling and decision making (Jennions & Petrie, 1997; Valone, Nordell, Giraldeau, & Templeton, 1996), the most prevalent theories for frogs suggest that females choose either the "best" of the closest n = 1, . . . , 5 (best-of-n theory, Janetos, 1980) or choose the first male they encounter whose quality is above a minimum threshold for acceptance (min-threshold theory, Jennions & Petrie, 1997). While these two strategies make different predictions in some cases, there is independent evidence for each of them. The goal of the project was to test them based on real-world data collected from frogs in the swamp on two consecutive nights. Specifically, given the position of males in the swamp and their respective pulse numbers , the question was which of the two strategies would lead to an average faster mating time (i.e., the time that it takes females to find and mate with a male partner). The hypothesis was that the best-of-n theory should lead to shorter average paths from initial female positions to males, and thus lead to overall shorter average mating times.

Step 1: Developing an agent model

In a first step, we need to define two new types of agents: male and female frogs. Agents in SimWorld can be defined in any of the available programming languages in Poplog (i.e., Pop11, Prolog, ML, Scheme, C-Lisp) or via condition-action rules in poprulebase . Additional support for external function calls to code written in other programming languages is provided via a C-function call interface as well as JAVA socket serialization mechanisms that can serialize and de-serialize Pop11 and JAVA objects, to the extent that they are similar in nature. Here, we show how to extend the built-in agent model in the Pop11 programming language. This is most easily accomplished by deriving a new class "treefrog" from the base class for simulated entities "thing" and adding those properties of frogs that are shared by male and female frogs (Figure 3). Then, two new classes for male and

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

female treefrogs can be derived from the frog class that contain only those properties that are specific to male and female treefrogs.

```
;;; define the base treefrog class
define :class treefrog; is thing;
    slot size == 4;             ;;; size of the frog in cm
    slot mating_range == 4;     ;;; range within which frogs can mate
    slot sound_range == 2000;   ;;; range within which frogs can hear
    slot speed == 0;            ;;; speed of movement

    slot sim_x == false;        ;;; x location in swamp
    slot sim_y == false;        ;;; y location in swamp
    slot heading = random(360); ;;; direction in which frog faces
    slot bodycolor == 'green';  ;;; color of frog in GUI
enddefine;

;;; defines a male treefrog
define :class male_treefrog; is treefrog;
    slot callpower = 0.00012447722025400753;  ;;; 102 db at 25 cm
    slot pulsepercall = gaussian(10,MALESTDDEV); ;;; gaussian
    slot territory_range = 70;  ;;; extension of mate site
enddefine;

;;; defines a female treefrog
define :class female_treefrog; is treefrog;
    slot call_speed == 1.86;    ;;; the speed if moving towards male
    slot matingmalesound == 0;  ;;; call quality of mating partner
    slot finalx == false;       ;;; the x position at mating
    slot finaly == false;       ;;; the y position at mating
enddefine;
```

Figure 3: The object-oriented structure of agents in SimWorld: the “treefrog” class that contains the properties shared by both male and female frogs, derived from the base class “things” for all simulation agents in SimWorld; and derived classes for male and female frogs.

In a next step, we need to define the behavior of these two new agents. Female frogs sample the environment for calls from male frogs, evaluate calls according to a particular strategy, and then decide in which direction to move. For male frogs, we restrict ourselves to a subset of male behaviors, where male frogs remain stationary throughout their calling period and simply call all the time with their given initial call parameters (i.e., number of pulses per call and the given call power). In comparison, male frogs in the real world change their calls based on perceptions of other calls, move to different calling sites, stop calling altogether and become so-called “satellites” or engage in aggressive encounters with other male frogs (see Figure 4 for a screenshot from a more complex research simulation, Scheutz & Boyd, in preparation).

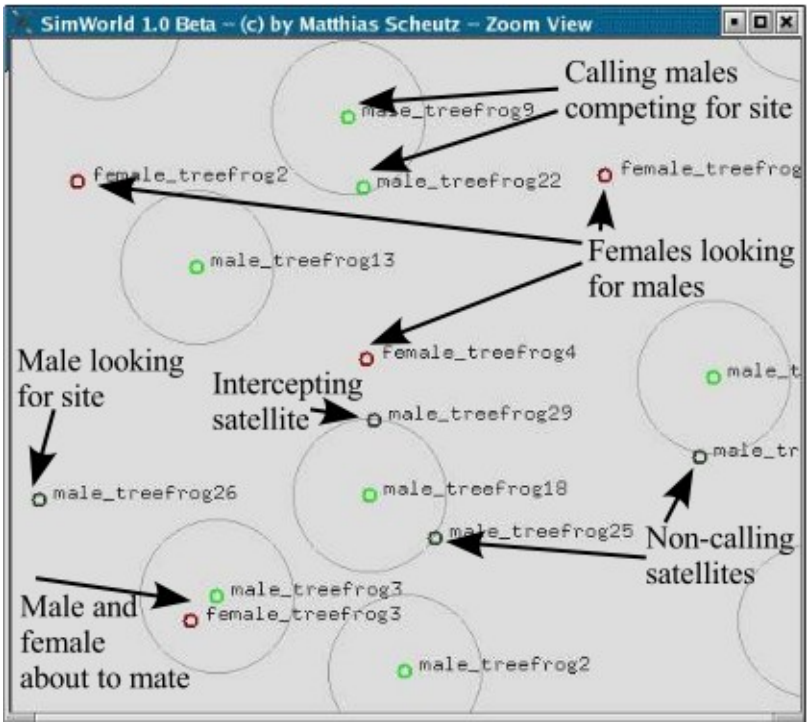


Figure 4: Screenshot from a more detailed simulation of male behavior in treefrogs.

Hence, there are functions that need to be implemented for both kinds of agent: (1) how to determine the female perceptions at each time, and (2) how to update bodily states based on those perceptions (including the execution of actions). In SimWorld , there are function templates that can be filled in:

```
define :method sim_run_agent(female:female_treefrog,entities);
define :method sim_run_agent(male:male_treefrog,entities);
```

Here, “entities” is a list of entities that the female/male could potentially perceive. Thus, implementations of the “sim run agent” method will typically check whether an entity is within some sensory range (e.g., the “sound range” given in the frog class), and if so, compute what exactly is perceived of that entity (e.g., the call rate of a calling male). In the case of the females, the perceptions are simply the male call characteristics, in the case of males there are no perceptions. Moreover, another function template needs to be filled in that determines how agents react to their perceptions (e.g., by changing bodily parameters and/or initiating motion):

```
define :method update_body(male:male_treefrog);
define :method update_body(female:female_treefrog);
```

For males, only the call action needs to be performed, but no internal states will change, while females update their position by moving in the direction of the male the have chosen based on their evaluation strategy (i.e., best-of-n or min-threshold) from all the males whose calls they could perceive. Specifically, females using the best-of-n strategy compare the calls of the closest *n* males they can perceive, select the best caller, and move it his direction, while females using the *minthreshold* strategy move towards the closest male they can perceive, whose call rate is above the minimum threshold (both n and threshold parameters are set at the beginning of a simulation run).

Finally, an initialization template needs to be filled in which determines how the simulation starts out:

```
define :method initialize(agent:treefrog,parents);
define :method initialize(obj:nest_site,parents);
```

This typically determines the initial position of an agent in the environment. In the case of male frogs, they might be placed in particular locations (e.g., as determined in field studies), while the females might be placed in random locations at the border of the swarm.

Step 2: Running the model in GUI mode.

Once all the classes and their behaviors are defined, the simulation can be run in SimWorld in GUI mode. This mode allows for the interactive modification of the model (e.g., debugging) but also for the exploration of the model's properties. For example, it is possible to interact with the simulation via the command line (which prints the current cycle number) and query the status of agents:

```
34 ? see female_treefrog5 heading
```

The above, for example, will print the heading of the agent with the name "female treefrog5".

Alternatively, right-clicking on the agent will also reveal an agent's internal state. If desired, the heading can be easily changed (e.g., to 25 degrees):

```
34 ? set female_treefrog5 heading 25
```

Similarly, it is possible to define new agent types, add and remove agents from a running simulation, and track and record the values of their slots (e.g., their positions throughout a run). This greatly facilitates the development and debugging of agent models.

To run a simulation, three parameter lists have to be specified (see Figure 5): (1) a list for setting up the simulation, (2) a list for defining entities, and (3) a list for scheduling events.

```
[
  [ /*** set up simulation environment ***/
    [initfile 'frog2.p']
    [compilehere 'define checkend(objects,cycle);
                    female_list==nil and cycle > 1;
                  enddefine;']
    [quitif checkend]
    [world WIDTH 1500 HEIGHT 1500]
    [world UNLIMITED]
    [compilehere '19.5 -> USEMINQUALITY;']
  ]
  [ /*** set up entities ***/
    [male_treefrog [startup [[[sim_x -696] [sim_y -70]
                              [pulsepercall 17.6]]]]]
    [male_treefrog [startup [[[sim_x -525] [sim_y 670]
                              [pulsepercall 18.6]]]]]
    [male_treefrog [startup [[[sim_x -538] [sim_y 360]
                              [pulsepercall 20.7]]]]]
    [male_treefrog [startup [[[sim_x -582] [sim_y 210]
                              [pulsepercall 15.1]]]]]
    [male_treefrog [startup [[[sim_x -540] [sim_y -75]
                              [pulsepercall 20.1]]]]]
    [male_treefrog [startup [[[sim_x -312] [sim_y 712]
                              [pulsepercall 21.6]]]]]
    [male_treefrog [startup [[[sim_x -273] [sim_y 660]
                              [pulsepercall 17.9]]]]]
    [male_treefrog [startup [[[sim_x -115] [sim_y -164]
                              [pulsepercall 19.3]]]]]
    [male_treefrog [startup [[[sim_x 102] [sim_y -140]
                              [pulsepercall 20.3]]]]]
    [male_treefrog [startup [[[sim_x 150] [sim_y -702]
                              [pulsepercall 22.7]]]]]
    [male_treefrog [startup [[[sim_x 300] [sim_y -250]
                              [pulsepercall 25.8]]]]]
    [male_treefrog [startup [[[sim_x 463] [sim_y 395]
                              [pulsepercall 17]]]]]
    [male_treefrog [startup [[[sim_x 593] [sim_y -75]
                              [pulsepercall 17.8]]]]]
    [male_treefrog [startup [[[sim_x 683] [sim_y 517]
                              [pulsepercall 22.9]]]]]
    [male_treefrog [startup [[[sim_x 640] [sim_y 405]
                              [pulsepercall 18.4]]]]]
    [male_treefrog [startup [[[sim_x 657] [sim_y 328]
                              [pulsepercall 23]]]]]
    [female_treefrog [startup 5 []]
    [record [at death [initialx][initialy][finalx][finaly]
                      [matingmalesound]]]]
  ]
  [ /*** schedule events ***/
  ]
]
```

Figure 5: The three-part setup of a simulation.

The first list contains information about which agent definition files to load ("initfile"), whether to compile additional code ("compilehere") that changes default functionality (in this case, the simulation is supposed to end if no females

are left, so the function “checkend” is re-defined and used as argument for the “quitif” keyword that defines termination conditions), and the extension of the world (“UNLIMITED” following the specification of the limited 2D world in terms of the “WIDTH” and “HEIGHT” means that agents can leave the area, but that random initial placements of agents will be confined to it). Also note that global simulation parameters can be set as part of the start-up (e.g., 19.5 is assigned to the global variable “USEMINQUALITY”).

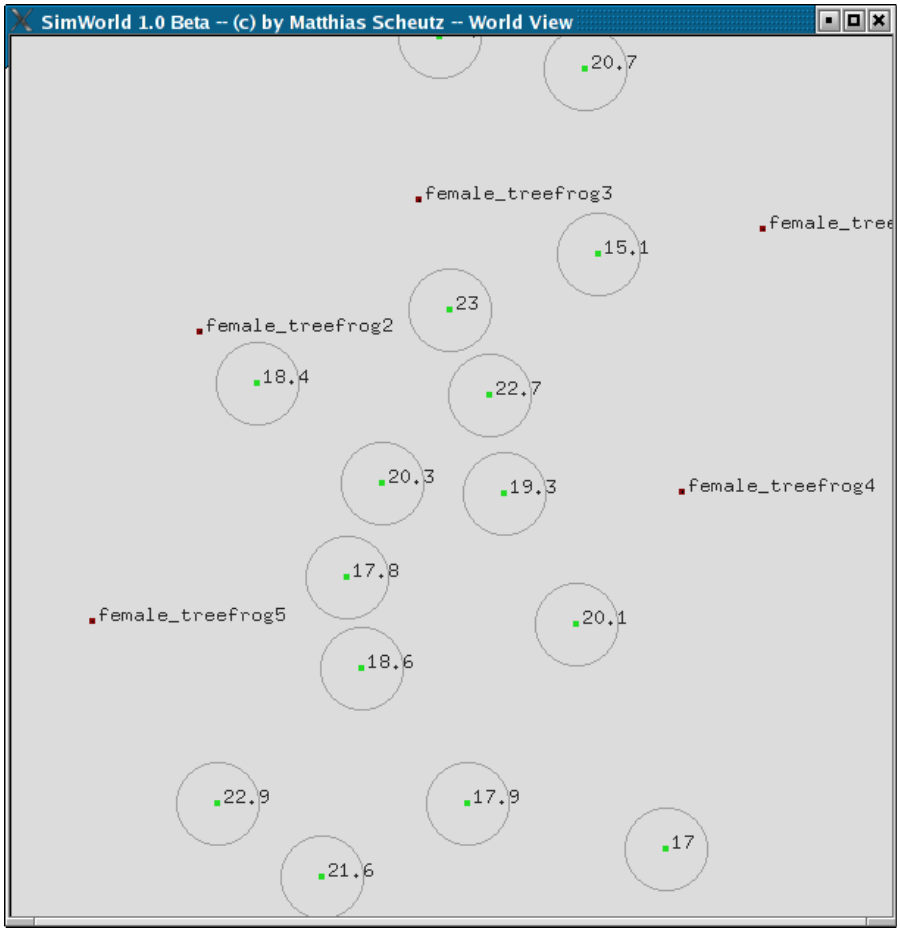


Figure 6: Initial configuration of environment...

The second list will both define all entity types to be used in the simulation as well as how to set them up (via the “startup” keyword, followed by entity parameters such as their position, or the initial call rate for male frogs). Note that female frogs do not have an assigned position, hence SimWorld will generate random positions within the specified 1500x1500 area. Furthermore, the “record”

keyword can be used to specify what information should be recorded for entities at what time (e.g., the "at death" qualifier means that the listed agent variables should be recorded for the agent before it is removed from the simulation–in the current case of the frog simulation, we simply remove both male and female frog when they are in the same location, i.e., when they start "mating", as neither frog will be able to mate with anybody else that night and we are only simulating one night at a time). Since the information of where females mated and what the call rate of their mating partner was is the information we need to be able to compare and evaluate different partner selection strategy based on the length of the female's trajectory to her partner and the call rate of that partner (which is a measure of his "fitness"), we need to record the initial and final positions for females as well as the call rate of their male partners (this is particularly important in "batch mode", where no graphical simulation output is available, but only the various values recorded throughout the simulation as specified via the "record" keyword).

The third list (left empty in Figure 5) is used to schedule events (e.g., adding new entities or changing entity properties at particular times).

Figure 6 shows the initial placement of male and female frogs on the first night (numbers next to males depict their call rate). Figure 7 shows the situation for females using the best-of- 3 strategy at a point when only one female ("female treefrog1") is left (all other females have already mated and were removed from the simulation). As can be seen from the traces of the past trajectories, "female treefrog1" originally moved towards the same male as chosen by "female treefrog4", and when that male stopped calling (due to his mating with "female treefrog4"), "female treefrog1" reevaluated her choice and picked the best of the now closest three males. Unfortunately, another female ("female treefrog2") managed to mate with that male before her, so she was forced to reevaluate her choice again (this time, however, she will be able to mate successfully with the chosen male as there not contenders left).

In this particular case, the ability to compare the outcomes of simulations visually is very helpful for gaining a quick understanding of the trade-offs between different values for *n* in *best-of-n* : the smaller *n* , the lesser the competition among females for males with high call rates, which in turn leads to shorter overall trajectories, but also lower overall "fitness" of the mating males (as determined in terms of their average call rate). Conversely, the higher *n*, the greater the competition among females, which in turn leads to longer female trajectories (as some females will moves towards males in vain as in the above case), but also higher average fitness of the mating males. These kinds of insights can be quickly obtained by watching the simulation unfold on the screen. At that point, particular hypotheses can be formulated (e.g., whether *best-of-n* or *min-threshold* are better) and tested in larger-scale simulations run in batch mode.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

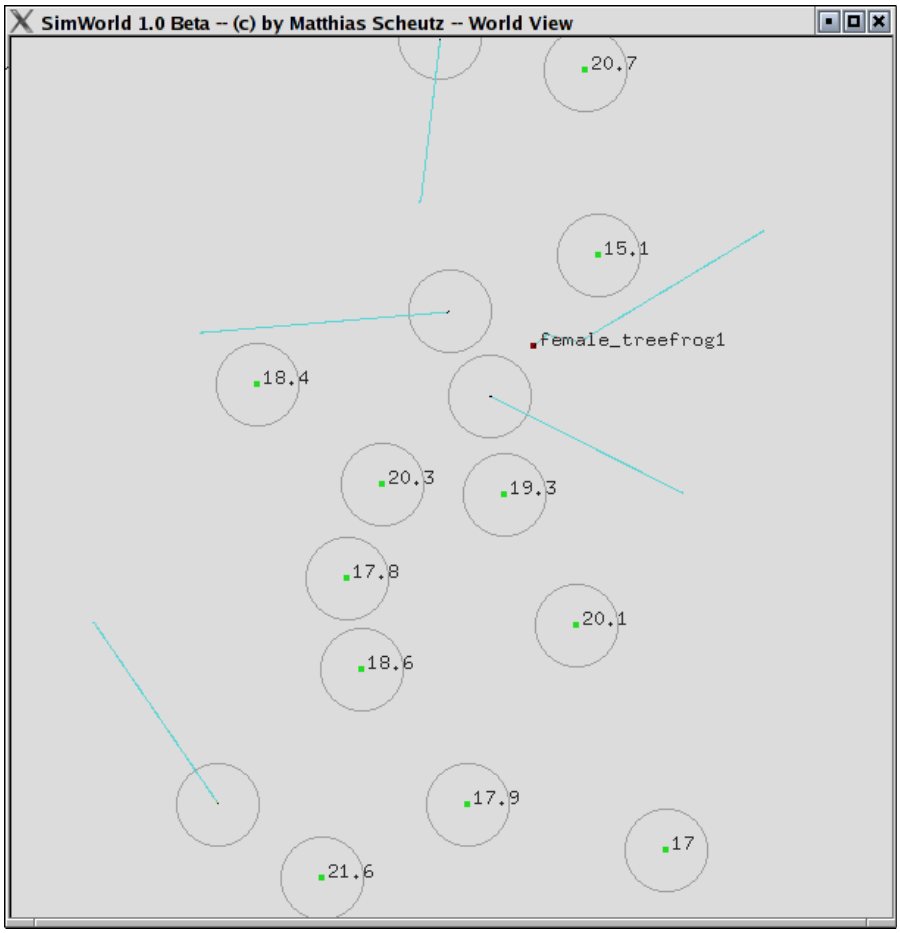


Figure 7: All females except for one have mated, female trajectories are shown in light-blue ...

Step 3: Defining and running batch experiments

Running simulations in batch mode is a straightforward process, once all agents have been defined and tested (in GUI mode). An additional list of simulation parameters, this time not for SimWorld, but for the SWAGES environment that schedules and supervises multiple simulations in batch mode, has to be defined (see Figure 8).

The main difference between simulations in GUI mode and batch mode is typically that the latter consist of a whole set of simulation runs (e.g., varying initial conditions of agents such as their placement in the environment or internal states to investigate the extent to which simulation outcomes depend on or are sensitive to changes in initial states). There are several ways to specify sets of

simulations. For random variations of initial conditions only (for all those internal states of agents, including their position in the environment, that are generated at random) SWAGES provides the keyword “replicates” which specifies the number of simulations that need to be repeated with different initial conditions (i.e., a different “random seed” for the random number generator). It is also possible to specify an initial seed for a whole experiment set (that way all simulations part of that set will receive a unique random seed and thus be replicable). In the case of the treefrog experiments, we run the simulation for 50 different initial conditions, keeping the position and call rate of all males the same, while altering the initial position and heading of all females (moreover, we specify an overall “ranseed” to be able to reproduce the simulations later, e.g., to be able to watch some of them in GUI mode).

```
[ /*** swages setup ***/
  [name 'minqual1' 'resultsdir']/* where to store */
  [user 'airolab'] /* run as this user */
  [priority 5] /* run at medium priority */
  [ranseed 29187] /* same initial conditions */
  [replicates 50] /* across all replicates */
  [watch 120 30] /* supervise execution */
  [email 'mscheutz@cse.nd.edu'] /* notify user when done */
  [savestate 'NO'] /* don't save simulation state */
  [copyimages '/tmp/'] /* temp space for sim state */
  [copystats 'statsdir'] /* where to store stats */
]
```

Figure 8: The additional setup of an experiment set in batch mode.

SWAGES also allows for the definition of “variates” (i.e., variables that are systematically changed) as part of the SimWorld setup. Each variate gets assigned the range and granularity of variation and based on that information SWAGES will produce individual simulations for each combination of values for all variates (e.g., if one female were supposed to be placed systematically in all positions from (0,0) to (100,100) at a granularity of 10, then SWAGES would produce 110 simulations that only differ with respect to the initial female position– this significantly reduces the number of experiments that need to be specified by users).

To allow for easy interaction, SWAGES provides a web-based interface that can be used with any standard web browser. Through this interface, it is possible to schedule and reschedule experiments, watch their progress, and inspect the results. Different users can use the system in parallel, and the SWAGES administrator can assign detailed privileges to each user (e.g., whether they can schedule experiments and if so, how many, with what priority, etc.). This is particularly useful for instructional settings where the instructor administers a set of host computers that will be shared by students for model simulations. Figure 9 shows the interface depicting scheduled and running experiments.

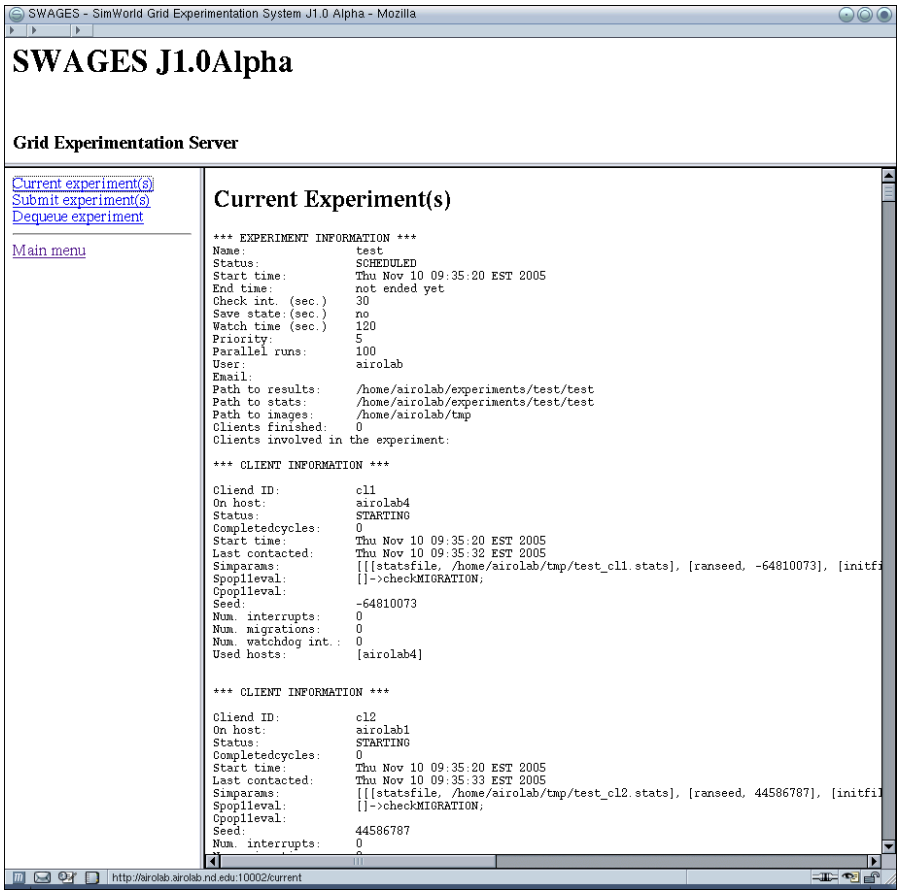


Figure 9: A scheduled experiment set and simulations running concurrently on several hosts.

Step 4: Analyzing the resultant data

Once the batch simulation experiments have finished, SWAGES will have produced a set of statistics files that contain the information specified via the "record" keywords (and some additional general information about the experiments). SimWorld comes with a set of tools that allow for mining and converting this data into various external format, e.g., export filters and scripts to interface common open-source statistics, visualization, and scientific computing software. Figure 10 shows the output of an automatically generated results file containing the relevant experiment statistical data formatted for import into R, and a (manually produced) batch file for R to perform basic ANOVA analyses. Moreover, Figure 11 shows a comparison space of the performance of min-threshold strategies based on male/female sex ratio and different minimum thresholds compared to the best-of-n strategy for a fixed n (the space was produce

in Scilab based on experimental output from SimWorld formatted for import in Scilab).

"Type"	"Night"	"Run"	"Fem"	"Cyc"	"InitX"	"InitY"	"FinX"	"FinY"	"PPC"
1	1	1	1	276	750	-491	300	-250	21.6
1	1	1	2	180	750	348	463	395	20.1
1	1	1	3	41	750	552	683	517	20.7
1	1	1	4	171	-750	-384	-696	-70	23.0
1	1	1	5	506	-750	-630	-115	-164	20.3
1	1	2	1	292	91	-750	300	-250	21.6
1	1	2	2	239	-750	646	-312	712	25.8
1	1	2	3	31	-750	-54	-696	-70	23.0

```
...
read.table("frog.dat") -> all; # load data from "frog.dat''
all$Type = factor(all$Type); # set factor strategy type
all$Night = factor(all$Night); # set factor for night
a = lm(all$Cycle ~ all$Night * all$Type); # create the model
summary(a); # print summary
anova(a); # run the ANOVA
b = lm(all$PPC ~ all$Night * all$Type); # create the model
summary(b); # print summary
anova(b); # run the ANOVA
```

Figure 10: R data file produced by SWAGES output filter and script in R to analyze the data.

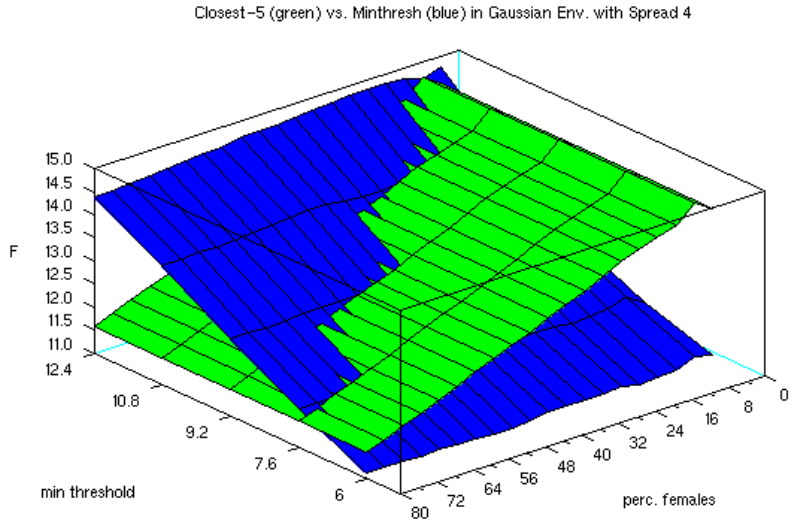
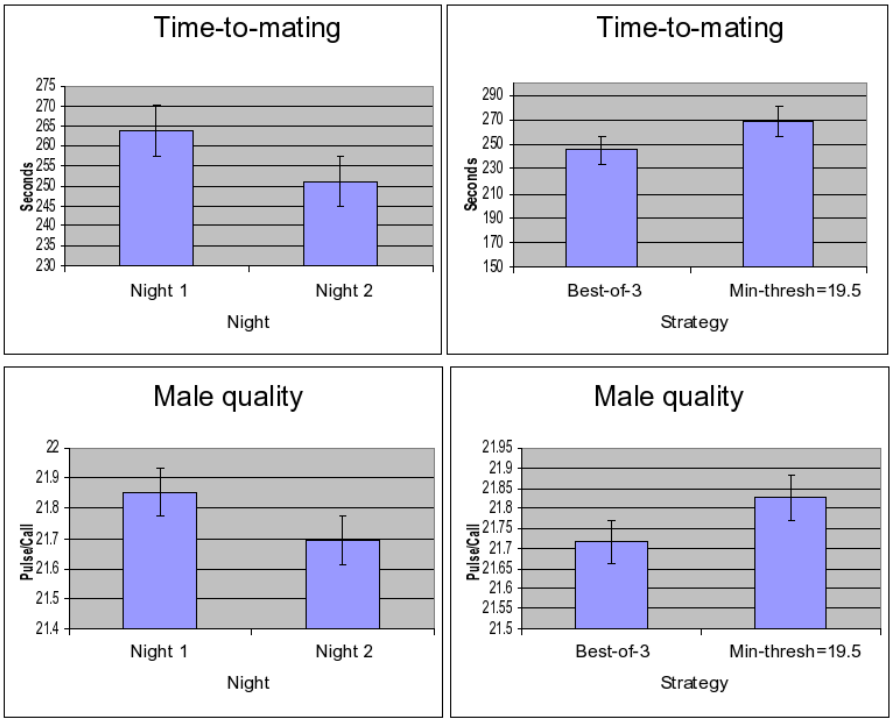


Figure 11: A performance space produced in Scilab (based on data generated by SWAGES) comparing two agent kinds (green and blue) along several dimensions.

In addition to tools for interfacing external statistics and visualization software, SWAGES also provides basic built-in statistical analysis tools (e.g., for performing simple significance tests), and libraries for data extraction and combination (in various formats including HTML, TeX, and plain text) for typical statistical operations that can be performed within the SWAGES web-interface.

The specific results for the frog experiments are shown in Figure 12 (generated from data imported into openoffice.org and from running the R script shown in Figure10 on the experiment output data).



	Time-to-mating			Male quality		
	Df	F	Pr(>F)	Df	F	Pr(>F)
Nights	1	1.50	0.22	1	1.13	0.29
Strategy	1	5.19	0.02	1	0.55	0.46
Night:Strategy	1	0.13	0.72	1	0.02	0.90

Figure 12: Results shown in graphs produced in openoffice.org based on the output of the simulation and based on ANOVAs performed in R. The bold-face number is statistically significant for $\alpha = .05$.

The top figures show "time-to-mating" broken down by nights and by strategy in terms of the overall time it takes a female (on average) to find a mating partner.

While there was no statistically significant difference between the two nights, the *best-of-3* strategy lead to a significantly shorter average time-to-mating (and thus a shorter average trajectory) compared to the *min-threshold*=19.5 strategy. This supports the initial hypothesis that *best-of-n* (for some *n*) can lead to shorter trajectories (compared to the population average minimum threshold) by rejecting the null hypothesis that there is no difference in trajectory length.

The lower figures compare the average male quality of the mating partners broken down by night and strategy in terms of the male call rate. Here we find no statistically significant difference, neither between the strategies nor between nights. Hence, the data does not support the hypothesis that *best-of-3* leads to a better average male fitness of the mating partner compared to the *min-threshold* strategy for the population average (by not allowing us to reject the null hypothesis that there is no difference in quality between the two strategies). Finally, there is no statistically significant interaction between any of the factors (see the ANOVAs in the table).

In sum, the simulation results confirmed one part of the initial hypothesis, but failed to provide evidence for the other. Beyond that, we can now also use the results of the simulations to make specific predictions, for example, that a slightly higher minimum threshold (e.g., of 20.5) will likely lead to significantly higher average mating partner fitness (as compared to *best-of-3*). This prediction can now be tested experimentally in additional simulation experiments.

Discussion

The above four-step procedures demonstrates one way of how artificial life simulations can be used to investigate interesting real-world phenomena in agent-based environments. In an instructional setting, instructors will typically introduce a particular problem in the domain to be studied (like the female mate choice in treefrogs) and then ask students to think about rules that would govern agent behavior in the given context (e.g., the decision-making process of frogs based on the two strategies). In a next step, students will be asked to implement the rules and experiment with them. This will typically require some familiarity with the SimWorld environment, which we have addressed in the past by assigning the tutorial included with SimWorld beforehand. The tutorial allows students to become familiar with the poplog environment (e.g., the key commands, how to load and start simulation, how to call up help files, etc.). In particular, it introduces them via simple examples to the general operating principles of the SimWorld environment, demonstrating simple agents and agent behaviors. As part of the tutorial, students learn how to use the graphical user interface and how to manipulate agents. At the end, they will have written simple rules modifying agent behavior and thus be ready for the actual project.

Typically, steps 1 and 2 will take some time until students have mastered the implementation (e.g., of rules for agents) and developed a model they are satisfied with (in our CSE 471/571 course, we assigned as a group project the development of rules for finding food that would allow agents to survive as long as

possible; the typical time frame for completing the setup was two weeks for groups of three students).

For some instructional situations, steps 1 and 2 will be sufficient, while for others, it might be useful to let students further investigate the model using steps 3 and 4. It is also possible for instructors to use steps 3 and 4 to evaluate the quality of the models produced by students. We have in the past, for example, used batch simulations to let agents produced by different student teams compete against each other in the same simulated world. The performance of these agents in direct competition was then used as a measure of success and to assign grades to student projects. Alternatively, student models could be compared to a standard model produced by the instructor in very much the same way. Depending on the instructional setting, therefore, SimWorld can be used by instructors to demonstrate the behavior of complex systems (e.g., by showing the simulation in class), or it can be used by students for assignments of varying complexity (e.g., using only step 2, i.e., to run simulations, or developing a model adding step 1, or investigating overall relationships among agent behaviors adding steps 3 and 4–for advanced projects several iterations through the four-step process might be required). In each case, it is up to the instructor to decide how much of the model infrastructure to provide (e.g., whether students get a whole agent model, or only parts of it, or have to implement agents from scratch). Hence, SimWorld can be a useful tool for students in different fields alike (e.g., biology students might largely adapt pre-defined models according to different biological theories, while computer science students might develop models of a particular given theory from scratch). Either way students will be able to develop an understanding of the dynamics of complex systems and might be able to discover principles and emergent behaviors that they would not have been able to see otherwise.

Related Work

Many simulation environments for the study of agent-based or Alife models have been proposed over the last decade. We distinguish between general agent toolkits and specific modeling toolkits, i.e., agent-based modeling toolkits and Alife toolkits , and will describe several examples of all three in the following.

General agent toolkits

General agent toolkits are intended as general purpose frameworks that provide components for creating programs using “agents”. Typically, the notion of “agent” is defined very broadly, referring to a computational structure that allows users to create distributed systems in which their agents are merely representations of resources in a program.

Mozart: The Mozart (Van Roy & Haridi, 1999) programming system implements the programming language Oz, which allows users to specify computation, concurrency, and distribution in a straightforward manner. Agents are programmed in Oz, taking advantage of its lightweight thread system and transparent data flow

mechanism. Distribution is also transparent, allowing agents access to local and remote resources with equal ease. Mozart uses tickets as global resource identifiers. Resources (i.e., agents) make their services available by sharing tickets with other agents. Agents in Mozart are allowed to dynamically add tickets as they are needed, providing flexibility to the application writer.

Mozart's transparent access to remote resources make it ideal for software projects that wish to provide a unified abstraction of distributed devices. There are, however, no abstractions provided for artificial life agents. As a general-purpose toolkit, it has the flexibility to support such agents, but implementation would have to begin largely from scratch.

Open Agent Architecture: The Open Agent Architecture (OAA) (Martin, Cheyer, & Moran, 1999) system provides mechanisms for distributed integration of heterogeneous applications. These mechanisms include facilitators, application agents, meta-agents, and user interface agents. Facilitators are the primary mechanism for allowing agents to cooperate. They coordinate communication and provide global memory for agents. Complex systems are allowed to utilize more than one facilitator. Application agents are service providers that may be specifically written for OAA or legacy applications encapsulated by wrappers. Meta-agents are domain-specific assistors for the facilitator. A meta-agent may make use of knowledge of the application to provide services difficult or impossible for the facilitator to provide. Finally, user interface agents provide humans access to the system. When an agent connects to the system, it notifies the facilitator and provides a list of the services it is willing to provide. The facilitator can then process requests from other agents and direct them to the new agent when appropriate.

OAA's ability to integrate preexisting heterogeneous applications into the agent framework makes it every flexible, even compared to other general agent toolkits. However, it is exactly this flexibility that makes it a less than ideal platform for instructional purposes – without substantial prior programming and setup of the computational infrastructure (including simulation environments) OAA is not suitable for educational projects.

Agent-based modeling toolkits

Different from general agent toolkits, agent-based modeling toolkits are developed with being suitable for agent-based models in mind. Here, agents typically perform as members of a population, thus providing a more traditional view of agents and allowing users to create simulation environments for examining the behavior of their agents. Typically, users can specify the details of the environment and agents without having to attend to the details of running the simulation. The framework manages updating the environment and agents and keeping track of relevant information over time. This allows experimenters to quickly begin observation and avoids duplication of effort.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Swarm: The Swarm toolkit (Minar, Burkhart, Langton, & Askenazi, 1996) was designed as a platform for studying complex systems, including artificial life simulations and simulated economies. In Swarm, agents are instantiated objects (i.e., code and data) whose member functions define the agents' actions. Groups of agents are collected in swarms, which, in turn, can function as agents. So, for example, Swarm can be used to model hierarchical systems in which an ecology is represented by a swarm of organisms, which in turn are represented as a swarm of organs, which in turn are represented by organelle objects. Users define a scheduler for each swarm that defines what events occur during the function of the swarm. Swarm provides measurement tools to observe agents' activities during a simulation. Swarm is a very powerful, yet flexible system onto which artificial life projects map very well. Resources and costs are determined by the user as part of the swarm definition. The object oriented programming model is easy to understand and extend, and useful measurement tools are provided.

While Swarm's generality is a major advantage for using it as an instructional tool, it requires a fair amount of time to become familiar enough with the environment to implement models, which might limit its use for less advanced students.

Hive: Hive (Minar, Gray, Roup, Krikorian, & Maes, 1999) is a toolkit that allows users to construct applications out of multiple networked agents. Cells, shadows, and agents make up the Hive architecture. Cells function as hosts for shadows and agents on the network. Communication is accomplished via cells, and manage requests for local resources. Local resources are encapsulated by shadows. They provide interfaces to hardware resources that can be invoked by agents. Unlike shadows, agents are mobile–they can move from cell to cell, using shadows local to those cells and moving on when a needed resource is not available locally. Agents communicate with one another to accomplish complex tasks in a network of cells.

Hive's focus is on providing simple connectivity between distributed agents in smart devices. Resources on remote devices are made available via a powerful set of abstractions. However, as a teaching platform for simulating biological agents, Hive is not a good choice given that any agent-based simulation model will basically have to be developed from scratch.

RePast: The RePast toolkit (Collier, 2003) provides an discrete-event simulation framework, borrowing many concepts from the Swarm toolkit. Different from Swarm, RePast has been implemented in several programming languages and environments and provides mechanisms for evolutionary computation. RePast is object-oriented and uses a fully concurrent discrete event scheduler, which supports both sequential and parallel discrete event operations. RePast also provides several mechanisms for recording and displaying simulation data, including various pre-defined 2D agent environments, which users can build on and easily modify (interfaces exist to many different programming languages like

C, C#, Java, Python, and others). Notably, RePast also includes tools for social modeling and has integrated support for geographical information systems (GIS).

While RePast is primarily a research platform, it has been used for educational purposes as well. The support of different programming languages and computing platforms together with the availability of models and its extensive documentation will in many cases make up for the time required to learn the basics of the toolkit.

Netlogo: NetLogo (Tisue &Wilensky, 2004) is an agent-based modeling environment intended for the simulation of biological and social phenomena. Agent behaviors can be programmed in a variant of the Logo programming language, which was originally designed as a graphical programming language for students with little to no programming experience. As such, Netlogo provides an easy-to-use graphical interface that allows user to interact with simulations (e.g., to inspect agents, or determine the behavior of an agent collective as it unfolds over time). A unique feature particularly well-suited for the classroom is the included HubNet tool, which allows students to control agents in a given simulation via hand-held devices.

Netlogo is a very versatile, easy-to-use modeling tool that, different from most other tools, has been targeted at educational settings, including undergraduate and K12 settings (even though it has been been used for modeling research as well). Moreover, due to its implementation in JAVA, it can run on different computational platforms and models can be easily turned into web applets that can then be put on web pages for demonstration purposes.

Artificial life toolkits

Similar to agent-based modeling toolkits, artificial life systems are more specialized than general agent toolkits, also providing frameworks specifically designed to allow experimentation with agent-based models. Different from agent-based modeling toolkits, Alife toolkits often focus on evolutionary, biological processes and thus typically provide additional built-in mechanisms for their support, sometimes at the expense of being applicable to other agent-based modeling domains.

Tierra: Tierra (Ray, 1994) system is an early example of an artificial life environment. In Tierra, the organisms are programs whose sole purpose is to reproduce. The system time-slices between them, and the processor represents the only resource in the system. Tierra introduces mutations during reproduction, allowing the genetic information (i.e., the program) to evolve over time. Some mutations lead to program errors; when a program commits an error, it moves up on the "reaper queue," an ordering of processes scheduled to die. Other mutations, however, allow the program to reproduce more efficiently, either by making their instruction count smaller (thus allowing them to reproduce more often per time slice) or by stealing resources (i.e., processor time) from other processes.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Tierra is an intriguing study of emergent behavior, especially when one realizes that Ray wrote only one simple agent for it that mutated into a large number of viable organisms. Tierra's limited scope, however, limits its applicability as instructional tool. Multiple-resource problems cannot be modeled using Tierra, and the programming language used to create organisms (a simple assembly language designed to be easily mutated) would be difficult to program to use for other architectures.

Avida: Avida (Adami & Brown, 1994) is similar to Tierra. Agents are strings of program instructions, and processor time is the only resource. Unlike Tierra, which places agents in a flat "soup" of memory, Avida uses a grid structure to impose locality effects on agents. Thus, for example, when agents reproduce, the offspring are placed in an adjacent cell (in Tierra, they could be assigned memory from anywhere in the soup). Avida also adds an element of user interaction: agents can be rewarded (with additional processor time) for evolving user-specified tasks, such as addition.

Avida's reward mechanism allows it to evolve complex programs that perform tasks specified by the user; this means that, in principle, Avida could be used to solve pragmatic problems. However, it also inherits many of Tierra's limitations.

Evo: The Evo (Krumpus, 2001) framework is designed specifically to allow researchers to experiment with evolution in artificial agents. Based on the Swarm toolkit (described above), Evo adds mechanisms for genetic inheritance, agent traits, and agent behavior. Agent traits are the unit of inheritance, and are represented as floating point values. Users are allowed to specify whether a trait is observable. Thus, a trait such as size could be observable by other agents and could effect their behaviors. Agent behaviors are composed of instructions, senses, and observations. Instructions are simple acts, such as "eat." Senses allow agents to determine characteristics of their surroundings, such as temperature. Observations are traits of agents, as described above. The user specifies which instructions, senses, and observations can be parts of behaviors. After that, the Evo system randomly creates behaviors for the initial agents, and these are passed on during reproduction. During reproduction, the traits of each parent are combined using a crossover function that creates a new list of traits and a new list of behaviors for the offspring. Mutation is also provided, and can operate on either traits or behaviors. The rate of mutation is user-specifiable.

Evo is a sophisticated platform and a useful tool for exploring evolutionary processes.

Xraptor: XRaptor (M¨ossinger, Polani, Spalt, & Uthmann, 1997) is a simulation environment developed to allow experimentation with agent control systems. The system provides an environment in which agents (flies and bats) forage for resources (fruit and flies, respectively). The body mechanisms for both agent types

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

are provided, along with an interface to the control system. Users create control systems, which are placed into bodies by the simulator during experimental runs. Xraptor provides users with the flexibility of conducting experiments in either two-dimensional or three-dimensional space.

XRaptor is a useful tool for students as it has been designed with focus on education, providing a platform upon which students' control systems can compete. Moreover, new body types can be introduced for more varied experiments.

Echo: Simulation of ecological systems is the primary focus of Echo (Forrest & Jones, 1994). An Echo run consists of a world subdivided into a fixed number of sites. Each site may contain several agents, as well as several resources. Agents have a genome that determines which resources they can gather, as well as their observable characteristics and internal traits. Agents may interact by trading resources, mating, or fighting, depending on the two genotypes involved. Sexual reproduction is supported, and uses a two-point crossover mechanism. Mutation can alter the genotype at any locus, including toggling resource genes and altering the strength of genes controlling external characteristics and internal traits.

Echo provides an environment in which to explore the potential for complex patterns of resource exchange to emerge. Its evolutionary mechanisms allow it to start with simple agents and observe as they evolve into cooperative (or combative) agents. However, the genotype is limited, making it difficult to create more sophisticated control systems. An interesting feature of Echo is the ability to associate a "tax rate" with a site, essentially making some places more expensive to occupy than others. A mechanism like this would be useful for exploring the effects of movement in more difficult terrain. As with several other above described modeling systems, Echo will have limited use in education beyond ecological models.

Gecko: The Gecko (Booth, 1997) simulator is loosely based on Echo, described above. It simulates a world composed of a number of sites, which contain agents and resources. Unlike Echo, however, which treats agents essentially as points, Gecko models agents with extent, making space an important component in agent interactions. Agents are represented as spheres projected onto sites, and many of their functions are influenced by their size. The sphere represents the area of influence of the agent, rather than the physical size. In Gecko, interactions are between agents whose spheres intersect. The size of the sphere is related to the biomass of the agent. Thus, resource uptake is regulated by the size of the agent, and the tax assessed is a function of the rate associated with the site multiplied by the size of the agent.

Gecko is designed to allow experimenters to test hypotheses about the interactions of individuals within an ecosystem. It has been used to simulate ecosystems with insects and spiders (Booth, 1997) and colonies of bacteria (Kreft,

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Booth, & Wimpenny, 1998). Its treatment of spatial extension is an improvement on Echo for observing interactions. Gecko, however, eliminates the evolutionary mechanisms provided with Echo; it is intended as a simulator of existing static populations, rather than evolving populations. Otherwise, it has the same limitations as Echo apply for instruction.

Geb: Control system evolution is the target of the Geb (Channon & Damper, 1998a, 1998b) environment. Geb is a two-dimensional world in which agents may reproduce, fight or move (including changing orientation to the left or right). Reproduction is accomplished using a one-off crossover mechanism (i.e., the crossover point in the second parent is always one off from the crossover point in the first, leading to variation in genotype size) and random mutation. The environment does not provide (nor require) resources. Fighting is the only cost implemented in Geb; losing a fight is fatal.

Geb provides a nice platform for experimenting with open-ended evolution of neural network control systems. The environment does not impose any artificial constraints on fitness, allowing natural selection to take its own course. However, the lack of any resource or cost structure beyond the fight mechanism makes the results of a Geb simulation difficult to relate to the real world, and thus limits its utility for instruction.

Sims' 3D Agents: Sims (1994) created a contest environment in which artificial three-dimensional agents compete for control of a shared resource. The environment is an arena, and the resource is a block. Agents fight for control of the block, where 'control' is defined in terms of proximity and contact. Thus, the agent closest to and in greatest contact with the block wins (roughly speaking). The agents are three dimensional compositions of blocks, attached by various joint types (e.g., twist, universal, rigid). Movement is accomplished by actuating joints. Agents are selected for reproduction based on their performance in contests; agents that perform well will have more offspring in the next generation. Random mutation of the genotype ensures an evolving population.

One key feature of Sims' environment is that it is physics-based, hence it is useful for any model where physical forces and realism are critical, but it is inappropriate for any other kind of non-physical model.

Tileworld: One problem facing agents that plan is the fact that the world can change in mid-plan. Tileworld (Pollack & Ringuette, 1990) provides an environment that models the dynamic nature of realistic environments to explore the performance of various reasoning strategies. In Tileworld, agents are given points for filling in holes with tiles that can be pushed around. Agents must navigate an environment populated by obstacles that prevent tile movement and other agents that are also trying to fill in holes. Furthermore, the environment is unpredictable: new obstacles, tiles, and even holes can be generated at random intervals.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Tileworld's unpredictability makes the environment appropriate for examining how different planning mechanisms perform. Agents must decide between making fast decisions that lead to suboptimal solutions and prolonged plans that may not map onto the world by the time they are implemented. However, Tileworld is of limited use outside the planning domain.

Fishes: The artificial fishes developed by Terzopoulos, Tu, and Grzeszczuk (1994) model the behavior of several species of fishes. Their design is based on ethological studies, making them fairly accurate emulations of their targets. The simulation is three-dimensional, and the physics of swimming are modeled as part of the simulation. Fishes are modeled down to the level of muscles and fins, and they are able to learn how to swim based on trial and error via feedback from the environment. They also learn higher-level behaviors, such as predation, schooling, and collision avoidance.

The physical simulation implemented for the fishes makes this project very interesting. Each fish must learn how to use the tools it is given (i.e., muscles and fins) to move around in the environment, subject to realistic biomechanic and hydrodynamic principles. The range of application is thus restricted to models of fish and their behaviors.

Not Fishes: Zaera, Cliff, and Bruten (1996) created a simulation in which they attempted to evolve schooling behavior in artificial fish. Their simulation does not include the detailed reproduction of reality achieved by Terzopoulos et al., but instead focuses on mechanisms for evolving the neural net that control the agents' behaviors. Reproduction occurs using both crossover and mutation to produce new genotypes. Groups of clones are simulated in a three-dimensional environment and monitored for emergent behaviors. The group succeeded in producing dispersal and aggregation behaviors, but failed to evolve schooling behavior.

This simulation provides a mechanism for evolving control structures that was able to produce simple group behaviors. As with Fishes, the range of application in instructional settings is very limited.

PolyWorld: PolyWorld (Yaeger, 1994) is an ecological simulator that implements agents using biologically inspired parameters (e.g., genetics, metabolism, physiology). It is intended as a platform for studying biological problems such as evolution and ethology. The evolutionary system includes a detailed genetic description of the agents, with mutation and crossover applied at reproduction. Physiology and metabolism are modeled in great detail, with different characteristics and activities constraining resource capacity and use. Agents control their actions using a neural network "brain" that has learning capability.

The mechanisms of physiology and metabolism implemented in PolyWorld are a great step in the direction of a more realistic simulation of agents and make Polyworld an interesting tool for modeling complex systems, yet also restrict it to biological models.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Gaia: The Gaia project (Gracias, Pereira, Lima, & Rosa, 1997) is a two-dimensional simulation that applies four principles to creating a realistic environment: physical laws (e.g., for energy use), biological laws (e.g., for agent interactions), a nervous system (i.e., a neural network control system), and genetic inheritance and evolution. Multiple agent types compete in an environment containing a single resource, food (in the form of plants and dead agents). Gaia uses genetic mutation and crossover to produce variation in the population. Gaia's metabolic mechanism is of interest: costs are assessed based on movement and thought. The physical laws determine the cost of movement, and the cost of thinking is based on the size of the neural network control system. Other costs are assessed based on agent actions (e.g., fighting and collisions).

Gaia is a good environment for the study of costs involved in ecological models.

Discussion

The above toolkits all have their different individual strengths and weaknesses, which make them to varying degrees suitable as instructional tools. As a rule of thumb, general agent toolkits are the least useful and appropriate for educational settings due to their generality and flexibility, which otherwise is a virtue – the effort involved in setting up simulation environments with some basic agents will likely outweigh the benefits of using them in most educational settings.

Agent-based modeling and Alife toolkits, on the other hand, already provide a basic simulation infrastructure and allow users to focus on configuring it according to their needs (this includes adapting the environment, setting up specific agents, etc.). On the flip-side, pre-programmed environments can also limit users in the way they can design simulations and specify agents, which is often limited to the underlying programming language of a toolkit (e.g., C in Tierra (Ray, 2001), C++ in Avida (Ofria & Wilke, 2004), and Java in Hive (Minar et al., 1999)). Some require agent definitions in an interpreted meta-language (e.g., as in Mozart (Peter Van Roy, 1999)). SimWorld provides many different ways of defining agents (from any of the languages supported by Poplog, to external function calls via libraries or sockets). Moreover, most agent-based modeling and Alife toolkits come with built-in components for the simulation environments (e.g., simulation and graphics engines, statistics packages, etc.), which, different from SimWorld , can typically not be replaced.

While many toolkits provide mechanisms for recording data and filters for exporting the recorded data (e.g., Swarm, Hive, Repast), they do not reach the flexibility of SWAGES where any simulation variable defined in SimWorld (all slots of agents, global SimWorld variables, etc.) and all of the underlying Poplog and OS variables (e.g., memory consumption or system time) can be recorded in multiple ways, possibly performing operations on the data before recording it (e.g., as part of the recording process by virtue of dynamically compiled Pop11 functions specified as part of the experiment start parameters). More importantly, other environments do not provide easy mechanisms for defining, scheduling, and

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

running large-scale experiment sets, with mechanisms for fault-detection and automatic recovery from errors, statistical analysis and data visualization, and export filters for various open source software tools (all of which is supported in SWAGES).

Finally, none of the above discussed toolkits supports the automatic parallelization of agent-based simulations and dynamical distribution of simulations over a set of heterogeneous, dynamically changing hosts like SimWorld , even though this features will likely only be of importance in larger projects where students need to run several simulations in batch mode. Aside from the SimWorld /SWAGES environments, we believe that RePast and NetLogo are the best candidates among the above described toolkits for educational settings. Both toolkits provide a versatile, mature infrastructure that can be quickly used by students and teachers alike without requiring much or any prior experience (Netlogo more so than RePast), and both have already been used extensively in educational settings.

Conclusion

In this chapter, we proposed agent-based artificial life simulations as tools for students to discover and develop agent-based models. We presented a brief overview of our agent-based SimWorld simulation. SimWorld allows users to specify agent models in multiple programming languages (e.g., Pop11, Lisp, C, Java) and provides a standard plug-in interface to link in external physics engines and graphical visualization tools. Together with its SWAGES experimentation environment, SimWorld provides a flexible, integrated platform for systematic, large-scale agent-based Alife simulation experiments and support for automatic parallelization of simulation models.

SimWorld and SWAGES have been successfully employed over the last six years both in research and educational projects. Specifically, they have been used repeatedly in our previous CSE 471/571 “Artificial Intelligence” course and in several independent student research projects. We have described one such project here that investigated two different female mate choice strategies in treefrogs and demonstrated how students can go through a sequence of four steps, starting with model development, model observation and refinement, large-scale experimentation, and data analysis that closely mimics the research cycle in computational modeling. The application of SimWorld in instructional settings, however, is not limited to the four-step cycle, but also encompasses classroom demonstrations and various kinds of exercises based on a subset of the four steps. Hence, we intend this chapter to be only an illustration of one of many possible ways, in which agent-based simulation environments can be employed in instructional settings, and thus invite instructors and students alike to explore the utility of agent-based simulations in their fields.

Finally, we would like to point out that both SimWorld and SWAGES are freely available (for noncommercial purposes) at http://hrilab.cogs.indiana.edu and encourage the reader to try them out.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Acknowledgements

The author would like to thank his colleague Sunny Boyd for introducing him to the fascinating subject of female choice and frog phonotactic behavior, Jayme O'Hara, the high school senior using SimWorld in her student project on female choice, for providing valuable feedback about SimWorld, and Paul Schermerhorn for providing some of the summaries for related modeling toolkits.

References

Adami, C., & Brown, C. T. (1994). Evolutionary learning in the 2D artificial life system avida. In R. Brooks & P. Maes (Eds.), Proc. artificial life IV (p. 377-381). MIT Press.

Anderson, J. J. (2002). An agent-based event driven foraging model. Natural Resource Modeling,15 (1), 55-82.

Andrews, S. S., & Bray, D. (2004). Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. Physical Biology, 1 (137-151).

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Science , 99 , 7280-7287.

Booth, G. (1997). Gecko: A continuous 2-d world for ecological modeling. 3 (3), 147–163.

Channon, A. D., & Damper, R. I. (1998a). Evolving novel behaviors via natural selection. In Proceedings of Artificial Life VI (pp. 384–388).

Channon, A. D., & Damper, R. I. (1998b). Perpetuating evolutionary emergence. In Proceedings of SAB 98.

Collier, N. (2003). RePast: An extensible framework for agent simulation. http://www.econ.iastate.edu/tesfatsi/RepastTutorial.Collier.pdf.

Conte, R. (2002). Agent-based modeling for understanding social intelligence. Proceedings of the National Academy of Science, 99 , 7189-7190.

Forrest, S., & Jones, T. (1994). Modeling complex adaptive systems with echo. In R. J. Stonier & X. H. Yu (Eds.), Complex systems: Mechanisms of adaptation (pp. 3–21). IOS Press.

Gao, Y., Madey, G., & Freeh, V. (2005). Modeling and simulation of the open source software community. In Spring simulation multiconference: Agent-directed simulation. San Diego.

Gerhardt, H. C. (1991). Female mate choice in treefrogs - static and dynamic acoustic criteria. Animal Behaviour , 42 , 615-635.

Gibson, R. M., & Langen, T. A. (1996). How do animals choose their mates? Trends in Ecology and Evolution , 11 (11), 468-470.

Gracias, N., Pereira, H., Lima, J. A., & Rosa, A. (1997). Gaia: An artificial life environment for ecological systems simulation. In Proc. artificial life v.

Greenfield, M. D., & Rand, A. S. (2000). Frogs have rules: Selective attention algorithms regulate chorusing in physalaemus pustulosus (leptodactylidae). Ethology , 106 (4), 331-347.

Grimm, V. (1999). Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future? Ecological Modelling, 115 (2-3), 129-148.
Janetos, A. C. (1980). Strategies of female mate choice - a theoretical-analysis. Behavioral Ecology and Sociobiology, 7 (2), 107-112.
Jennions, M. D., & Petrie, M. (1997). Variation in mate choice and mating preferences: A review of causes and consequences. Biological Reviews of the Cambridge Philosophical Society, 72 (2), 283-327.
Kreft, J.-U., Booth, G., & Wimpenny, J. W. T. (1998). BacSim: A simulator for individual-based modelling of bacterial colony growth. 144 , 3275–3287.
Krumpus, M. A. (2001). Overview of the Evo artificial life framework. http://omicrongroup.org/evo/overview/html/overview.html.
Le Novre, N., & Shimizu, T. S. (2001). Stochsim: modelling of stochastic biomolecular processes. Bioinformatics, 17, 575-576.
Martin, D., Cheyer, A., & Moran, D. (1999). The Open Agent Architecture: a framework for building distributed software systems. Applied Artificial Intelligence, 13 (1/2), 91–128.
Minar, N., Burkhart, R., Langton, C., & Askenazi, M. (1996). The Swarm simulation system, a toolkit for building multi-agent simulations. http://www.santafe.edu/projects/swarm/overview/overview.html.
Minar, N., Gray, M., Roup, O., Krikorian, R., & Maes, P. (1999). Hive: Distributed agents for networking things. In First international symposium on agent systems and applications (ASA'99)/third international symposium on mobile agents (MA'99). Palm Springs, CA, USA.
M¨ossinger, P., Polani, D., Spalt, R., & Uthmann, T. (1997). A virtual testbed for analysis and design of sensorimotoric aspects of agent control. Simulation Practice and Theory , 5 (7–8), 671–687.
Ofria, C., & Wilke, C. (2004). Avida: A software platform for research in computational evolutionary biology. Journal of Artificial Life , 10 (2), 191-229.
Peter Van Roy, S. H. (1999). Mozart: A programming system for agent applications. In International workshop on distributed and Internet programming with logic and constraint languages.
Pollack, M., & Ringuette, M. (1990). Introducing the tileworld: experimentally evaluating agent architectures. In Proceedings of the eighth national conference on artificial intelligence.
Railsback, S. F., Harvey, B. C., Lamberson, R. H., Lee, D. E., Claasen, N. J., & Yoshihara, S. (2002). Population-level analysis and validation of an individual-based cutthroat trout model. Natural Resource Modeling, 15 (1), 83-110.
Ray, T. (2001). Overview of tierra at atr. In Techn. inf. 15, technologies for software evolutionary systems.
Ray, T. S. (1994). An evolutionary approach to synthetic biology: Zen and the art of creating life. In Artificial life (Vol. 1, pp. 195–226). MIT Press.
Russ, S. (1997). Empirical modelling: the computer as a modelling medium. The Computer Bulletin, 39 (2), 20-22.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Ryan, M. J., & Keddyhector, A. (1992). Directional patterns of female mate choice and the role of sensory biases. American Naturalist, 139, S4-S35.
Schermerhorn, P., & Scheutz, M. (2003). Implicit cooperation in conflict resolution for simple agents. In Agent 2003.
Schermerhorn, P., & Scheutz, M. (2005). The effect of environmental structure on the utility of communication in hive-based swarms. In Ieee swarm intelligence symposium 2005.
Scheutz, M., & Boyd, S. (in preparation). Female choice revisited: Agent-based models of phonotaxis in female treefrogs.
Scheutz, M., Madey, G., & Boyd, S. (2005). tMANS–the multi-scale agent-based networked simulation for the study of multi-scale, multi-level biological and social phenomena. In Proceedings of spring simulation multiconference (smc 05), agent-directed simulation symposium.
Scheutz, M., & Schermerhorn, P. (2002). Steps towards a theory of possible trajectories from reactive to deliberative control systems. In R. Standish (Ed.), Proceedings of the 8th conference of artificial life. MIT Press.
Scheutz, M., & Schermerhorn, P. (2004). The more radical, the better: Investigating the utility of aggression in the competition among different agent kinds. In Proceedings of SAB 2004. MIT Press.
Scheutz, M., & Schermerhorn, P. (2005). Predicting population dynamics and evolutionary trajectories based on performance evaluations in alife simulations. In Proceedings of GECCO 2005.
Scheutz, M., & Schermerhorn, P. (2006). Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. Journal of Parallel and Distributed Computing, 66 (8), 1037–1051.
Scheutz, M., Schermerhorn, P., & Bauer, P. (2005). The utility of heterogeneous swarms of simple UAVs with limited sensory capacity in detection and tracking tasks. In IEEE Swarm Intelligence Symposium 2005.
Scheutz, M., Schermerhorn, P., Connaughton, R., & Dingler, A. (2006). Swages - an extendable distributed experimentation system for large-scale agent-based alife simulations. In Proceedings of Artifical Life X.
Schoenharl, T., & Madey, G. (2004). The agent based modeling approach to simulating neural networks (Tech. Rep.). Working Paper, University of Notre Dame.
Schwartz, J. J., Buchanan, B.W., & Gerhardt, H. C. (2001). Female mate choice in the gray treefrog (hyla versicolor) in three experimental environments. Behavioral Ecology and Sociobiology, 49 (6), 443-455.
Schwartz, J. J., Huth, K., & Hutchin, T. (2004). How long do females really listen? assessment time for female mate choice in the grey treefrog, hyla versicolor. Animal Behaviour, 68, 533-540.
Sims, K. (1994). Evolving 3d morphology and behavior by competition. In Proc. artificial life IV.
Sloman, A. (1999). Sim agent help file.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.

Terzopoulos, D., Tu, X., & Grzeszczuk, R. (1994). Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. 1 (4), 327–351.
Tisue, S., & Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In International conference on complex systems. Boston.
Valone, T. J., Nordell, S. E., Giraldeau, L. A., & Templeton, J. J. (1996). The empirical question of thresholds and mechanisms of mate choice. Evolutionary Ecology , 10 (4), 447-455.
Van Roy, P., & Haridi, S. (1999). Mozart: A programming system for agent applications. In International workshop on distributed and Internet programming with logic and constraint languages.
Yaeger, L. (1994). Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or polyworld: Life in a new context. In Proceedings of Artificial Life III.
Zaera, N., Cliff, D., & Bruten, J. (1996). (Not) evolving collective behaviours in synthetic fish. In Proceedings of SAB 1996.

A.N. Other, B.N. Other (eds.), Title of Book, 00–00.