Chapter 4
3D Multi-Agent Simulations in the breve Simulation Environment

Jon Klein and Lee Spector

4.1 Overview

Artificial life is often described as the study of life "as it could be," meaning that it involves the exploration of life-like interactions of agents with novel behaviors in novel environments. Although these behaviors and environments can be explored using mathematical models or even real-world experiments, one of the most commonly employed techniques is multi-agent simulation. Using multi-agent simulation software, users can define the rules of an environment and the behaviors of individual agents in order to examine how they behave collectively, their so-called "emergent behaviors."

One of the more challenging and time-consuming tasks in exploring multi-agent systems is developing the software infrastructure for modeling and simulation. Even models of conceptually simple interactions between agents often require a great deal of software infrastructure to manage and coordinate agent behaviors.

Many artificial life and multi-agent system models rely on realistic three-dimensional (3D) spaces, especially those that aim to produce results relevant in explaining real-world behaviors (such as simulations that give insights into biology) or those that may eventually be implemented in the real world (such as simulations of traffic or sensor networks). Moreover, some simulations, such as those dealing with robotics, require realistic simulation of physical interactions between agents and their environments. These applications present a particular challenge in simulation development because they require sophisticated spatial and physical simulation algorithms in order to produce accurate simulation results.

Breve is a free, open-source software package that aims to simplify the creation of 3D simulations of multi-agent systems and artificial life.[1] Breve

[1] Breve was developed and is maintained by Jon Klein. The breve-based research described in this chapter is the product of both authors.

frees simulation authors to focus on the definition of agent behaviors and on descriptions of the simulated environment, whereas the breve engine manages simulation data and algorithms. Agents' behaviors can be written in Python, or using a simple scripting language called "steve." Breve includes support for physical simulation and collision detection for the simulation of realistic creatures or robotics and an OpenGL display engine for the visualization of simulated worlds. Breve runs on a number of platforms and is distributed as a pre-built package for Mac OS X, Linux, and Windows from the breve website [7].

Breve is analogous in many ways to 3D game engines that allow for the rapid development of games while abstracting away many of the details of their implementation. Breve aims to do the same for multi-agent simulations. The similarity is more than conceptual – breve actually preforms many of the same types of computation that 3D game engines do, such as 3D rendering, management of agent interaction, and collision detection. Unlike game engines, however, breve is designed for simulation applications, especially those relating to artificial life and artificial intelligence. To this end, breve includes a rich library of features suited to simulation applications.

4.1.1 Agent-Based Modeling Paradigms

Aside from breve, there are several other multi-agent simulation software packages that can be used to study artificial life. These environments vary greatly in their representations of space and time, and these representations strongly influence the types of simulations to which an agent-based modeling system is best suited.

Some simulation environments have no explicit modeling of space. In these environments, abstract spatial structures (such as networks) can be achieved via connections between individual agents. Another popular representation of space in agent-based modeling is a discrete two-dimensional (2D) or 3D grid. The grid introduces a realistic spatial structure and allows agents to interact based on spatial proximity, but it somewhat limits the granularity of these spatial interactions. Breve uses a fully continuous representation of 3D space, which allows it to be used for realistic 3D simulation of phenomena such as swarms, robots, and vehicles.

Certain phenomena, such as chemical concentrations or temperature (in addition to many other kinds of environmental states), are difficult to accurately and efficiently model using individual agents and benefit greatly from the use of a discrete 2D or 3D grid. To facilitate the simulation of these phenomena, breve supports discrete 2D and 3D grids, which can be used independently or in concert with continuous spaces. This allows agents to move through continuous 3D spaces while interacting with discrete environmental states.

With respect to representation of time, the differences between the various simulation environments can be more subtle. Some systems use large, discrete time-steps to represent a period of time passing. Others, like breve, integrate arbitrarily small time-steps to simulate fully "continuous" time (although at the lowest levels, the representation of time is still discrete). In these environments, it is understood that the time-step is generally as small as possible, given the practical computation constraints of arbitrary small integration steps.

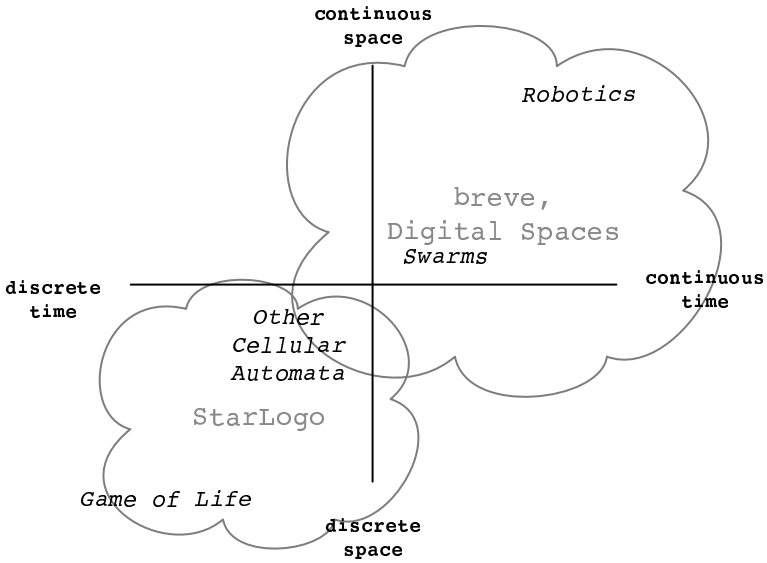

Fig. 4.1 Continuum of representations of space and time in simulation environments.

Fig. 4.1 shows a general representation of the "simulation space" defined by the spectra of discrete and continuous representations of time and space, along with the locations of common simulation paradigms and the general locations of some common agent-based modeling toolkits.

This diagram represents only a broad illustration of the distinction between discrete and continuous time and space in simulation. In reality, the distinction is often less clear, as any sufficiently extensible agent-based modeling environment is capable of modeling any representation of time or space, although doing so may require additional development effort on the part of the simulation author. In particular, some common agent-based modeling systems do not have single built-in representations of time and space. These systems may provide basic multi-agent simulation functionality such as management of agents and their interactions, while leaving the modeling of time

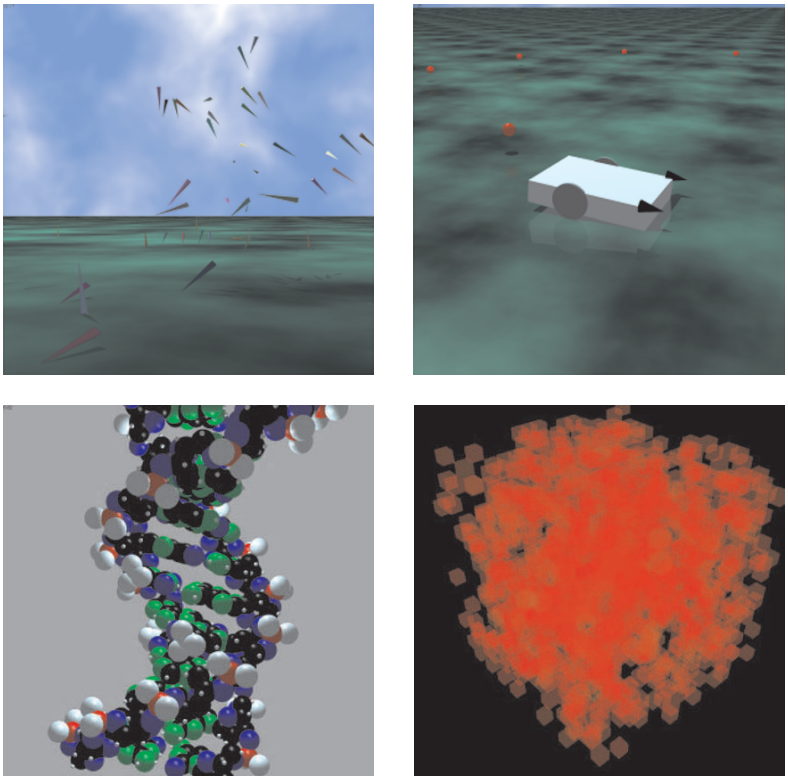

Fig. 4.2 Sample simulations in breve. Clockwise, from top left: flocking agents; a Braitenberg vehicle; a 3D game-of-life cellular automata; a DNA molecule visualized from a PDB file.

and space as a detail for the simulation author to implement according to their needs.

Fig. 4.2 shows a sample of the different simulation paradigms that can be represented in breve, including continuous non-physical simulations, continuous physical simulation, and discrete 3D cellular automata.

4.1.2 Comparison to Other Agent-Based Modeling Systems

There are several other agent-based simulation modeling environments that are useful for exploring artificial life and multi-agent systems. Some of the more notable environments are discussed here briefly.

As alluded to earlier while describing representations of space and time in simulation environments, choosing between simulation packages for a particular simulation application is, by and large, not a matter of a "laundry list" comparison of simulation features among the packages. All of the environments described here are powerful and extensible environments with support for authoring custom simulations. The important differences between the environments is generally found in the types of simulations they are best equipped to model.

- StarLogo and StarLogo TNG – StarLogo [17] is the oldest of the agent-based simulation packages described here. StarLogo allows users to model multi-agent systems through simultaneous manipulation of multiple Logo "turtles" in a discrete 2D world, using an integrated programming language derived from Logo. In addition to its multi-agent simulation applications, StarLogo (like the original Logo) is used as an educational tool to teach students and novice programmers how to program and how to construct multi-agent models based on simple agent behaviors.

  StarLogo was one of the original inspirations for breve. Although breve does not place as strong an emphasis on education applications as StarLogo, the inspiration to use an integrated, interpreted language grew directly out of StarLogo's implementation.

  StarLogo TNG ("The Next Generation") described in Chapter 6 is a new version of StarLogo currently in development that includes support for continuous 3D space and for a visual programming language.
- Repast – Repast [16] described in Chapter 2 is a Java-based agent simulation toolkit that is geared toward social simulation and that also allows for visual construction of simulations. Although Repast does include support for simulation and visualization of agents in 3D spaces, it does not provide support for physical simulation.
- MASON – MASON [12] is a Java-based simulation toolkit that supports 2D/3D simulation as well as network topologies. Like Repast, MASON supports simulation and visualization of agents in 3D spaces but does not include physical simulation functionality.
- Swarm – Swarm [13] is an Objective-C-based simulation library originally developed at the Santa Fe Institute. Compared to the other systems described here, Swarm provides support for multi-agent modeling at a relatively low level. Swarm provides a library of tools for managing agent behaviors and relationships, with less of an emphasis on features such as spatial representation and visualization. Swarm does offer a "Space" library with support for discrete 2D spaces, and sample code on the Swarm website details how Swarm can be extended to model discrete 3D spaces.
- Digital Spaces – One of the newest additions to this list is the Digital Spaces simulator, which is also the most similar to breve in terms of simulation "niche." Digital Spaces includes support for 3D physics in simulations and for importing complex 3D environments. Although Digital Spaces has not been used heavily for artificial life research, it does offer

an impressive feature set that makes it an excellent candidate for such applications.
- Framsticks – Framsticks [9] described in Chapter 5 was not originally developed as a general-purpose agent-based modeling toolkit, but it does deserve a special mention in comparison to breve because it models artificial life agents with articulated bodies in 3D physical worlds. Framsticks has traditionally taken a more specialized simulation approach with a focus on evolved virtual creatures, although versions 2 and 3 provide more general agent-based modeling capabilities in the software.

4.2 Motivations

The fundamental goal of breve as a simulation environment is to abstract away simulation implementation details as much as possible and to allow users to instead focus on constructing agent behaviors and the environment.

4.2.1 A Personal Motivation

Although there was a considerable amount of academic inspiration behind the development of breve, an equally important inspiration (for Jon Klein, breve's primary author) came on a more personal level from a college friend with a great interest and insight in artificial life and artificial intelligence, but (to put it gently) no particular talent in computer programming.

This friend would observe a simulation or other programming project, typically the product of several weeks of work, and cheerfully make an astute suggestion for a "simple" improvement that would invariably require several more weeks of work. This particular interaction was repeated so many times that, in frustration, one would eventually snap "do it yourself!" to which the friend *cheerfully* retorted "I can't!"

And so a small – but genuine – motivation for the development of breve was to silence this friend.

4.2.2 Design Principles and Goals

The guiding motivation in the implementation of breve is that experimentation with artificial life and multi-agent systems should be simple and should not be limited by programming ability. To this end, a number of design principles behind breve are described here. Given the rapid pace of evolution in the fields of multi-agent simulation and artificial life (and computing in gen-

eral), these principles represent moving targets, so even after several years and many revisions of the software, these concepts still guide the continued development of breve.

- *breve should be approachable by people of all programming abilities.* Practically speaking, this means breve should use a simple language and simple interfaces to artificial life simulation features such as evolvable code, genetic algorithms, and physical simulation.
- *breve should be an integrated development environment, not a software library or application programming interface (API).* Although some powerful agent-based simulation toolkits, such as Swarm, are implemented as a software library, it can represent a considerable barrier for new users, especially those without a great deal of prior programming experience.

  The decision to implement an integrated environment instead of an external library does come with downsides. Integrating breve into other software projects, for example, is a more complicated prospect than integrating an environment that is distributed as a software library.
- *breve should allow users to immediately see the connection between code and simulation behavior.* Breve is intended not only to enable the development of of multi-agent systems but also to encourage the exploration of these systems through an integrated visualization engine and by streamlining the write/run/revise cycle of simulation development. This principle (and especially its manifestation in StarLogo) strongly influenced the decision to make an integrated scripting language a main component of the simulation environment.

Ironically, the decision to implement a custom language to make breve more accessible to new users may have had the opposite of its intended effect. Although the steve language is arguably simpler and more approachable than many other languages used for simulation software (such as Java, C, C++), the main breve developer may have overestimated the enthusiasm that potential users would feel for learning a new, proprietary language. With programming languages, familiarity may trump simplicity.

Because of this, and keeping in line with breve's development goals, breve now supports the Python language and is in the process of transitioning toward Python as the preferred simulation language.

As for the college friend who so greatly influenced the development of breve, he still does not use breve to implement his simulation ideas. Whether this is a shortcoming of breve or a character flaw is, after all these years, not entirely clear.

4.3 Writing Simulations in breve

This section provides a conceptual overview of how simulations are constructed in breve. Although breve simulations can be written in either a proprietary language called steve or in the popular Python language, this section describes general concepts which apply to writing simulations regardless of which language frontend is used. The details of the individual scripting languages supported by breve are described later in Sections 4.3.5 and 4.3.6.

4.3.1 Object Orientation and the Built-in breve Classes

The breve environment makes extensive use of object-oriented concepts both as a programming model and as a more general representation of the simulated world. Every object in a simulation corresponds directly to a programming language object in the steve or Python frontend language.

Furthermore, all programming in breve is preformed by extending the built-in classes – there is no code or data in the steve language that is not associated with an object. (Although the Python language frontend does allow code outside of classes, this code is not able to interact directly with the breve engine.) The breve distribution includes a standard library of classes that interface with the internal features of the breve engine or that themselves implement useful simulation features. These classes are then subclassed to implement custom behaviors.

The special class "Object" represents the root object class in breve, and all other objects are subclasses, either directly or indirectly, of the Object class. Practically speaking, very few classes inherit directly from Object. Instead, breve defines two Object subclasses that serve as a logical distinction of all objects in breve: the Real subclass, which includes all objects which have a physical presence in the simulated world, and Abstract, which includes those that do not. Abstract objects are most often used to encapsulate data and computation, and they more closely resemble objects in traditional programming languages than do Real objects.

4.3.2 The Controller Object

The "controller" object in breve is a special object that is created by the engine when the simulation begins. The controller object is a custom subclass of the class "Control" that the user implements to set up and manage the simulation. Because the controller object is the only object created when a simulation begins, it is responsible for creating all other objects from its *init* method.

The controller object is accessible to every object in the simulation and, because there are no global variables in the steve language, the controller plays an important role in holding global simulation state data and in facilitating communication between the agents.

4.3.3 The breve Simulation Loop

Agents in a breve simulation are simulated following a pattern called the *breve simulation loop*. The basic behavior of each agent is summarized in the following pseudocode:

```
init agent [run user defined ``init'' method]
while ( simulation running )
    foreach agent:
        iterate agent [run user defined ``iterate'' method]:
            examine internal/external states (using sensors)
            perform computation
            change behavior (using actuators)
```

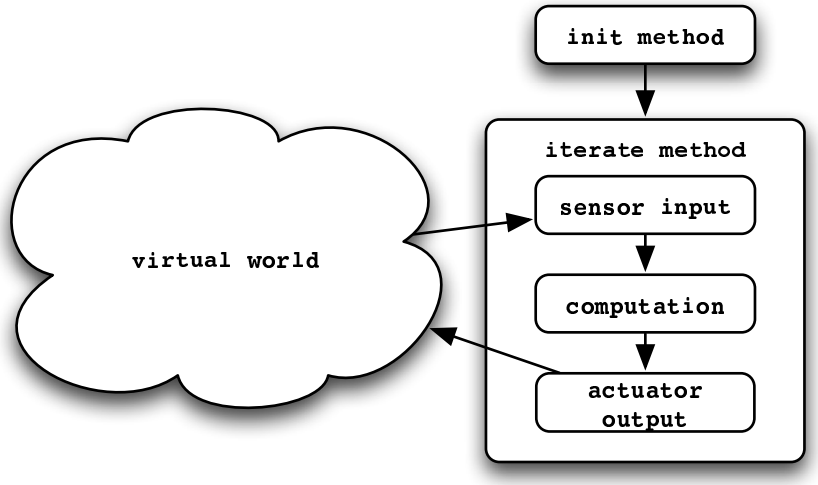

Fig. 4.3 The breve simulation loop.

An abstract diagram of this process is illustrated in Fig. 4.3, and two examples of how this process is implemented with actual simulations are shown in Fig. 4.4 and Fig. 4.5.

Fig. 4.4 shows the simulation loop for Creatures, a demo breve simulation of physically simulated creatures evolving via a genetic algorithm. In spite

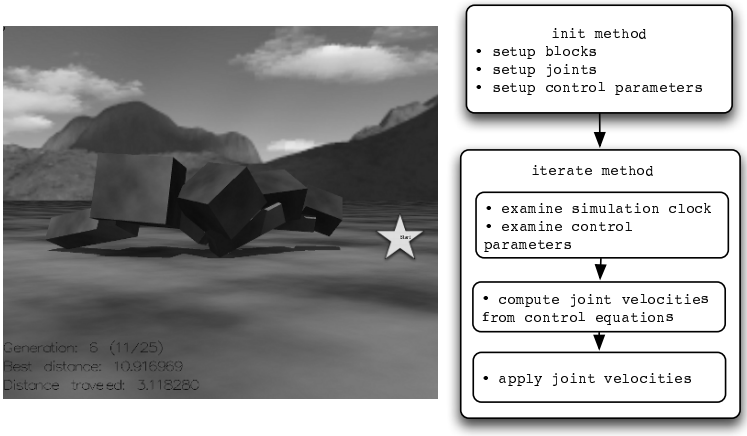

Fig. 4.4 The breve simulation loop for an agent in the "Creatures" simulation.

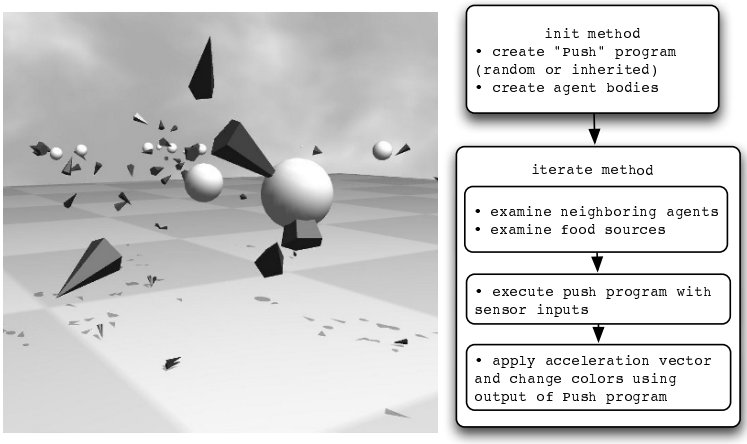

Fig. 4.5 The breve simulation loop for each agent in the "SwarmEvolve 2.0" simulation.

of the complexity of the simulation behind the scenes, the simulated agent's behavior is rather straightforward: The agent takes simple inputs from its internal state and from its genome and produces outputs in the form of forces applied to joints.

Fig. 4.5 shows a breve demo, SwarmEvolve-2.0, with more complex interactions between agents and their environment. In this simulation, agents are initialized by creating evolvable code in the Push programming language (described in detail in Sect. 4.4.4) that will control their behavior. Then at each time-step, the agents sense the world to provide input to their Push

programs, execute the Push programs, and then use the results to change various behaviors, such as heading and color. This figure shows only a conceptual overview of the SwarmEvolve-2.0 agent behaviors and many of the details are omitted. The SwarmEvolve agents, for example, have access to many other types of input sensors and output actuators than those shown. A more detailed description of the simulation is described in Sect. 4.6.1.

4.3.4 Defining Callbacks and Agent Behaviors

Callback functions are used in breve to allow the simulation engine or front-end application to trigger events in simulation code. When possible, callback behaviors are automatically defined through the presence of methods with special names. In some cases, though, callbacks are defined explicitly via requests from simulation code. Some of the main callback types are described here.

- In response to collisions. Breve handles the collisions between objects at two different levels. First, if physical simulation is enabled for an object, the collision is resolved in the physics engine. Then the collision triggers a callback to one or more agents at the user simulation level. This callback can be handled by providing the breve engine with a callback method to be executed upon collisions with a certain object type.
- In response to object "announcements". Object announcements in breve provide a simple way for agents to communicate general events or state changes to all interested parties, without keeping track of message recipients or making method calls directly. With object announcements, any agent can register itself as a listener for another agent and can schedule a method call to be executed when a specific announcement is made. Agents may have any number of different announcement types identified by unique strings. Methods in the Object class are provided for both making announcements and for registering as listeners to announcements in other objects.
- In response to keyboard and mouse input. A number of special callbacks are defined that are executed automatically in the simulation controller object in response to keyboard and mouse events. Support for keyboard or mouse interaction can be added to a simulation simply by defining callback methods with the the proper names.
- In response to network events. Breve includes rudimentary network transfer support, and, as with other events, network transfers trigger method calls in the simulation's controller object. This allows users to react and respond to simulation objects sent over the network from other simulations.

4.3.5 The steve Programming Language

Breve simulations can be written in a simple object-oriented language called "steve." The steve language resembles and borrows features from languages like C, Objective-C, and SmallTalk and includes support for 3D vectors and matrices as native types.

Table 4.1 shows a simple demo simulation written in the steve language. In this simple simulation, agents are given an initial upward velocity and a downward acceleration to simulate the behavior of a fountain of particles.

```
@include "Mobile.tz"
@include "Control.tz"

Controller Fountain.

Control : Fountain {
    + to init:
        250 new Particles.

        self point-camera at (0, 9, 0) from (40.0, 2.1, 0.0).
}

Mobile : Particle (aka Particles) {
    + to iterate:
        if (self get-location)::y < -6.0:
            self reset.

    + to init:
         self set-acceleration to (0, -9.8, 0.0).
        self reset.

    + to reset:
        self set-color to random[(0, 1, 1)].
        self move to (0, 0, 0).
        self set-velocity to random[(10, 20, 10)] + (-5, 4, -5).
        self set-rotational-velocity to random[(.6, .6, .6)].
}
```

Table 4.1 A sample simulation in steve.

Method calls in steve resemble SmallTalk and Objective-C in that they use *keywords* to designate method arguments and the method arguments may be placed in any order. This feature enhances the readability of method calls for readers not familiar with the breve APIs.

Although steve is object-oriented, simple data-types such as int, float, matrix, vector, list, and hash are not first class objects in steve. Steve's type system treats these simple datatypes differently than true objects that are added to the breve engine and follow the simulation-loop pattern described above. Similar to typing in Objective-C, the simple data-types are statically typed, while the true objects follow a dynamic-typing scheme in which method calls are resolved at runtime and do not depend on the object type, so long as the object responds to the given method.

Steve also includes garbage collection via a reference counting scheme. This garbage collection is used automatically for the simple datatypes, but it must be manually enabled for other objects on a per-object basis. This is because objects in a breve simulation are referenced internally by the breve engine and may continue to play a role in a simulation (through their iteration methods) even when they are not referenced explicitly by any other simulation object. In addition, objects not explicitly referenced by others (in the classical programming sense) may reacquire references through simulation events (such as announcements or collisions) or through the steve language's "all" command, which provides a list of all objects of a requested type.

4.3.6 The Python Programming Language

As of breve version 2.6, simulations can be written in the popular Python language. Because of Python's powerful reflection and introspection features, most of the benefits of steve's tight integration with breve can be emulated with Python, so that, by and large, the user experience and APIs are identical for simulations written in both steve and Python. Table 4.2 shows the same sample simulation listed in Table 4.1 but this time implemented in Python.

The Python and steve languages can communicate seamlessly through bridge objects, allowing code from both languages to be mixed together in a single simulation. This allows even existing simulations written in steve to take advantage of the rich library of Python modules, one of Python's greatest strengths. Python objects can be instantiated and manipulated directly from steve, and vice versa.

4.4 Breve Features and Technical Details

This section describes some of the basic simulation features that breve provides and details on how they are implemented.

4.4.1 3D Spatial Simulation

The foundation of the simulation functionality provided by breve, and the one upon which the others are built, is a subsystem that manages objects and their interactions in 3D worlds. The spatial simulation system handles placement of objects, manages collisions, and integrates accelerations and velocities to compute object positions. This system preforms all of the *non-*

```
import breve

class Fountain( breve.Control ):
    def __init__( self ):
        breve.Control.__init__( self )
        Fountain.init( self )

    def init( self ):
        breve.createInstances( breve.Particle, 250 )
        self.pointCamera(
            breve.vector( 0, 9, 0 ),
            breve.vector( 40.0, 2.1, 0.0 ) )

class Particle( breve.Mobile ):
    def __init__( self ):
        breve.Mobile.__init__( self )
        self.setAcceleration( breve.vector( 0, -9.8, 0.0 ) )
        self.reset()

    def iterate( self ):
        if ( self.getLocation().y < -6.0 ):
            self.reset()

    def reset( self ):
        self.setColor( breve.randomExpression( breve.vector( 0, 1, 1 ) ) )
        self.move( breve.vector( 0, 0, 0 ) )
        self.setVelocity( (
            breve.randomExpression( breve.vector( 10, 20, 10 ) ) +
            breve.vector( -5, 4, -5 ) ) )

        self.setRotationalVelocity(
            breve.randomExpression( breve.vector( 0.6, 0.6, 0.6 ) ) )

# Create an instance of our controller object to initialize the simulation
Fountain()
```

Table 4.2 A sample simulation in Python.

physical simulation tasks, although it is tightly integrated with the physical simulation engine, which is described below.

4.4.2 Physical Simulations

Breve includes support for realistic rigid body dynamics for simulation of virtual creatures and robotics. While the core functionality and interface to the breve physics engine has stayed the same over the years, the under-the-hood implementation has evolved, piece by piece, from a home-grown implementation to its current state, which largely uses the Open Dynamics Engine (ODE) library, a physical simulation engine based on a generalized coordinate method which models articulated bodies as a series of differential equations.

The original physics engine implementation was based heavily on Brian Mirtech's work on impulse-based rigid body simulation [15]. The original release of breve included its own physics engine based on the Featherstone

dynamics algorithm, a reduced coordinated algorithm that calculates the motion of articulated bodies by recursively computing force and mass properties for subtrees of jointed bodies. Physical contact in the engine was modeled using a micro-collision model in which resting contact is simulated through the use of a large number of low-velocity collisions and impulses. Collision detection was implemented as a two-pass process: The first stage, a prune-and-sweep sweep, found potential collisions by maintaining lists of sorted bounding box minima and maxima in all three dimensions; the second stage, which determined which candidate pairs were intersecting, used Mirtech's V-Clip algorithm [14], based on the Lin-Canny closest feature collision detection algorithm, with extensions to properly handle the interiors of polyhedra.

The prune-and-sweep portion of the original collision detection model is still used for collision detection and for the breve's *neighbor detection*. For this task, the prune-and-sweep algorithm efficiently discovers object pairs overlapping by a user-defined threshold, without running a second-stage collision check. This allows objects to quickly and efficiently discover nearby objects without preforming an $O(n^2)$ search of the entire simulation space – because of spatial coherence between simulation steps, this method of discovering neighbors is very close to $O(n)$.

Although the physical simulation in breve is a useful tool for developing simulations involving artificial life, evolution of creature morphologies, and robotics, it should be stressed that neither the original breve physics engine nor the ODE engine are intended to be truly predictive of real-world physical behaviors. Although the physics engine used in breve does place an emphasis on stability, it is still important for simulation developers to include sanity checks in physical simulations to avoid situations that can lead to instabilities or inaccuracies. In particular, excessively large velocity and mass values can cause simulations to "blow up" and should be avoided.

When used in conjunction with evolutionary computation, this aspect of physical simulation represents a particular challenge due to the uncanny ability of the evolutionary process to exploit any available means of improving performance, including exploiting bugs or inaccuracies in physical simulation. Karl Sims described this problem in his work on evolving virtual creatures [20]. For this reason, work with physical simulation often requires additional care and attention to detail.

4.4.3 Visualization

Breve includes an OpenGL-based visualization engine that renders 3D simulations in realtime. On several occasions (described later in more detail), breve's visualization engine has allowed for the discovery or understanding of phenomena that may have otherwise gone unnoticed.

Beyond simply rendering agents and their environment, the visualization engine can render lines, drawings, and a variety of "special effects" that may be used to enhance understanding of the simulation state. Shadows and reflections, for example, provide additional visual cues of agent location and orientation.

In addition to conveying "literal" information about agents in a simulation (size, location, orientation), the visualization engine can be used to convey arbitrary information about an agent's state by changing display qualities using a number of visualization features. High-dimensional agent states can be represented using visual qualities such as color, size, and transparency. In this way, even simulations modeling abstract agents or non-spatial simulation can benefit from 3D visualization.

The performance impact of high-quality visualization in breve is minimal. Most modern computers include powerful graphics cards that support hardware-accelerated 3D rendering, so in most cases visualization can be achieved with virtually no computational costs and thus no with impact on simulation speed.

4.4.4 The Push Programming Language

Push [40] is a stack-based programming language intended primarily for use in evolutionary computation systems. Although the steve and Python languages, in which breve simulations are written, are intended to be simple for human programmers to use, neither is well suited for use in evolutionary computation due to syntactical and semantical language constraints.

Push has an unusually simple syntax that facilitates program evolution. Despite its simple syntax, Push provides more expressive power than most other program representations that are used for program evolution. Push programs can process multiple data types (without the syntax restrictions that usually accompany this capability), and they can express and make use of arbitrary control structures (e.g., recursive subroutines and macros) through the explicit manipulation of their own code (via the "CODE" stack and data type) and through manipulation of their own execution (via the "EXEC" stack).

These features allow Push to support the automatic evolution of modular program architectures in a particularly simple way. Push can also support entirely new evolutionary computation paradigms such as "autoconstructive evolution," in which genetic operators and other components of the evolutionary system themselves evolve (as in the Pushpop [22] system, and in SwarmEvolve2, described in Sect. 4.6.1).

Push offers a powerful way to evolve open-ended agent behaviors in breve. By providing Push programs with direct access to agent methods via callback

instructions into simulation code, agent behaviors can be fully controlled via evolving code.

4.5 Development History and Future Development

Breve was originally developed by Jon Klein as part of master's thesis work at Chalmers University in Göteborg, Sweden. The first version of breve was released in late 2001. The initial versions of the breve integrated development environment were released for Mac OS X only, although the breve engine was available as a command-line application for Linux and Windows platforms. Since 2002, continued development of breve has been largely supported by Hampshire College.[2]

4.5.1 Transition to Open Source

With the release of breve 1.7, in late 2003, breve made the conversion from a closed-source project to a fully open-source project with source code available under the GPL. Since opening the source code to breve, users have contributed many improvements to the code, such as an integrated graphical interface for breve on Linux and Windows platforms as well as initial revisions of code to interface with Python and other frontend languages.

4.5.2 Push3

Beginning with version 2.0, released in late 2004, breve includes built-in support for the Push programming language designed specifically for use with genetic programming. The Push programming language is described in more detail in Section 4.4.4 and its applications in breve are described in Section 4.6.

[2] This material is based upon work supported by the National Science Foundation under Grant No. 0308540 and Grant No. 0749184. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

4.5.3 Python Integration

As of version 2.6, released in late 2007, breve features support for writing simulations and accessing all breve engine functionality through the Python scripting language. Because of its power and its popularity, Python is now the preferred language for new simulation development.

4.5.4 Future Development

Currently in development for the next release of breve is enhanced support for XML import and export of data. Simulation parameters in particular will be separated into standard XML files to be loaded when a simulation begins. This change will greatly simplify the process of exploring the parameter space of a simulation, which currently requires user interface interaction or changes to simulation files themselves.

Also in development for the next release of breve is a new graphical user interface, using the open-source Qt environment, that will unify the simulation experience on Mac OS X, Linux, and Windows. Among other improvements, the new interface will provide graphical user interfaces for managing the simulation parameter sets described above.

A final area of breve development in the longer term is providing greatly enhanced networking support. Breve currently offers support for simple network transfer of individual objects. Future enhancements will expand this functionality to support real-time interactions of agents in different simulations, similar to networking capabilities of modern game engines.

4.6 ALife/AI Research with breve

Here we demonstrate the range of breve's applicability by describing, briefly, some of the AI and ALife research that we have conducted using the software. Other individuals and research groups have been putting it to additional uses in a diverse set of areas including physically simulated evolving creatures [11], evolving ecologies [10], swarm robotics [6, 42], artificial intelligence applied to homeland security applications [43], simulations of sorting behaviors in ants [5], cognitive science research [4], and self-assembly in physical systems [2].

4.6.1 Evolving Swarms

The original distribution of breve included a Swarm demo that was essentially a re-implementation of Reynold's Boids system [18]. SwarmEvolve is an evolutionary extension of the original Swarm demo, in which the flying agents reproduce and evolve in the context of a simple energy dynamics: They collect energy from "feeders" and expend energy when they fly, collide, or crowd one another.

In the simplest, most highly constrained version of SwarmEvolve (1.0), the velocities of agents are computed as weighted combinations of environmental vectors (toward the nearest feeder, toward the closest agent, etc.), following the general scheme used in Reynold's Boids but extended for a modestly enriched environment. The weights used by each agent are derived from its genome, which is a sequence of floating-point numbers. Each agent is a member of one of three pre-defined species, and when an agent dies, it is replaced by a new agent whose genome is inherited, with mutation, from the most fit member of its own species.[3] In SwarmEvolve 2.0 there are no pre-specified species groups, behavior patterns, or reproductive regimes; an agent's genome is a program expressed in a Turing-complete language (Push), and its behavior (including perceptual, motor, and reproductive behavior) is the result of executing that program.

The SwarmEvolve systems have been used to demonstrate the emergence of several types of collective behavior, including multicellular organization, altruistic feeding behaviors, and tag-mediated cooperation [38, 37, 39]. They also produce life-like, visually compelling displays of 3D flocking behavior [30]. Screen shots of SwarmEvolve are shown in Fig. 4.6.

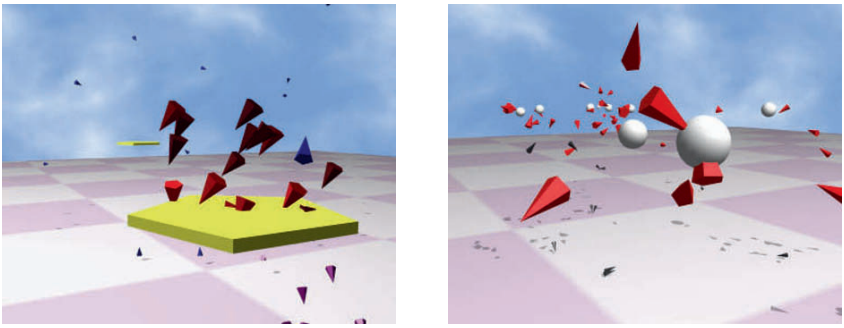

Fig. 4.6 A screenshot of SwarmEvolve 1.0 (left). The large pentagons are feeders from which the agents receive energy. In SwarmEvolve 2.0 (right), the feeders are spheres that shrink when they are eaten and re-grow slowly. The dark cones falling to the ground are corpses of agents that have run out of energy.

[3] Fitness is calculated as the product of age and energy.

4.6.2 Evolution of Cooperation

Using breve, we investigated the evolution of tag-mediated cooperation – altruism toward other agents that share a similar "tag" marker [19] – in multiple simulation paradigms in order to understand the phenomenon from different perspectives and in different contexts. Some of these studies used breve's 2D or 3D spatial simulation facilities to explore the interaction between neighborhood structures and the emergence of cooperation [37, 32, 33].

Breve's visualization engine played an important role in understanding the underlying mechanism of how colonies of cooperators were able to flourish in the spatial simulations. Without writing additional simulation code for visualization, we were able to step through simulations generation by generation, to observe altruistic donations as they took place, and to determine the conditions under which such cooperative activities would spread through the population.

4.6.3 Division Blocks

The Division Blocks project [34] combines several of breve's facilities, including complex physical simulation, high-quality graphical output, and support for efficient neural network processing, to explore the open-ended evolution of development, form, and behavior. Division Blocks are simulated rectangular blocks that can grow and shrink, divide and form joints, exert forces on joints, and exchange resources. They are controlled by recurrent neural networks that evolve, along with the blocks, by natural selection. Energy is approximately conserved, and all energy derives ultimately from a simulated sun via photosynthesis. A screen shot of the Division Blocks system is shown in Fig. 4.7.

4.6.4 Genetic Programming Research

Breve has proven to be useful as a vehicle for a wide range of evolutionary computation research – even research that does not use the system's core facilities for three-dimensional simulation. Breve's incorporation of the Push programming language [21, 41, 40, 36], in particular, has supported several research projects that extend or apply genetic programming techniques. Examples of such projects include:

- Automatic Quantum Computer Programming. In this project we used breve's support for PushGP in combination with its support for the QGAME quantum computer emulator to evolve quantum algorithms that

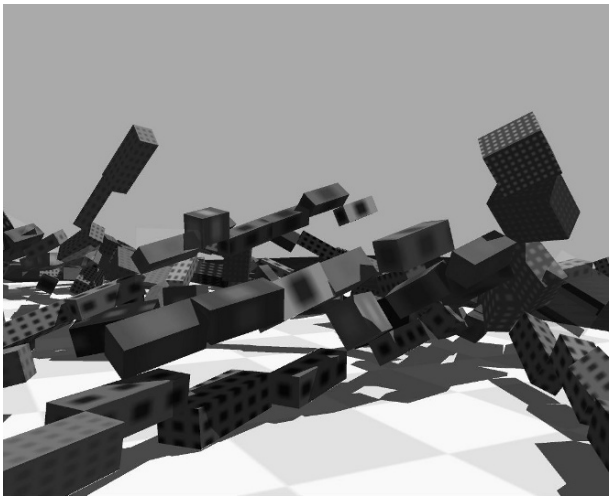

Fig. 4.7 A screen shot of the Division Blocks system.

compute functions of interest. This project has produced several human-competitive results [24, 26, 25, 1, 27, 23].

- Trivial Geography. In this project we used breve's PushGP implementation to develop and test a simple population-structuring technique that produced surprisingly dramatic improvements in problem-solving performance on a suite of test problems (ten symbolic regression problems and a quantum computing problem) [31].
- Unwitting Distributed Genetic Programming. In this project we developed a web-based system for distributed computation of genetic programming via asynchronous javascript and XML (AJAX), requiring no explicit user interaction and no installation of client-side software. Clients automatically and unknowingly participated in a distributed genetic programming run simply by visiting a webpage, thereby allowing for the solution of genetic programming problems without running a single local fitness evaluation [8]. Breve was used to handle server-side tasks such as population management and reproduction.
- Genetic Programming for Finite Algebras. In this project we applied genetic programming to problems in pure mathematics, in the study of finite algebras. We documented the production of human-competitive results in the discovery of particular algebraic terms, using both breve's implementation of PushGP and the ECJ genetic programming system [44]. We showed that GP can exceed the performance of every prior method of finding the relevant terms in either time or size by several orders of magnitude [28].

4.7 Educational Applications of breve

One of the more rewarding and unexpected uses of breve has been its application in educational settings. Because breve offers a relatively low barrier of entry to experimenting with multi-agent systems, it has been a useful tool in many educational contexts, even in those that do not deal explicitly with artificial life or simulation.

4.7.1 Artificial Life and Braitenberg Vehicles

The Braitenberg vehicles simulation included with breve has served as a useful introduction to many topics related to agent-based simulation, artificial life, artificial intelligence, and computer programming in general. It has been used for this purpose in several artificial life courses at Hampshire College.

Braitenberg vehicles, named after Valentino Braitenberg, are simple robotic designs that are capable of displaying surprisingly life-like behaviors from very simple circuitries of sensors and wheels [3]. One of the simplest creatures, dubbed "aggressor," demonstrates aggressive light-chasing behavior with a simple criss-crossed connection of light sensors and motors: Light sensed with the right sensor causes activation of the left wheel and vice versa, such that the vehicle turns toward and speeds in the direction of any lights in the environment.

Even by the standards of the rest of the breve demos, which are designed to be easy to use and to modify, the Braitenberg Vehicles simulation is exceptionally simple. A set of Braitenberg classes included with breve allows for the creation of Braitenberg vehicles and for placement of wheels and sensors in only a few lines of code. Students with no programming experience at all are quickly able to learn to design vehicles and place them in environments with lights (which the vehicles sense) and other obstacles. The students are thus able to quickly design and run simple robotics experiments with realistic physical simulation. A breve Braitenberg vehicle is shown in Fig 4.2.

4.7.2 Reactive Bouncy Balls for Kids

The "bouncy" breve simulation was developed for use in an activity with a fifth grade class at the Smith College Campus School. Like the Braitenberg vehicle simulation, the bouncy simulation allows users to create agents simply, but here the quest for simplicity has been taken to an extreme: The user can create an agent and specify its behavior with a single line of code such as

```
new bouncy with size 3.0 color (0.7, 0, 0.4) preferences (0, -1, 0.1).
```

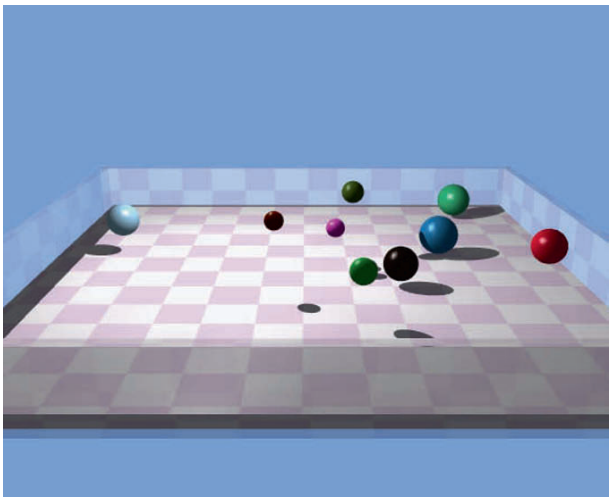

Fig. 4.8 The breve "bouncy" simulation.

This creates a "bouncy" ball with size 3, color reddish-purple (the three numbers in parentheses specify amounts of red, green, and blue), and a preference to move quickly *away* from green and slowly *toward* blue. Users can create arbitrary numbers of such balls, with different sizes, colors, and preferences, and observe the often complex dynamics that emerge as the balls bounce around in a physically simulated arena. A slightly more complex syntax allows for the specification of balls that change their colors and preferences when they collide with other balls, producing extremely complex patterns of activity. A screen shot of the bouncy simulation is shown in Fig. 4.8.

4.7.3 Biology and SuperDuperWalker

SuperDuperWalker is a software-based framework for experiments on the evolution of locomotion. It simulates the behavior of evolving agents in a 3D physical simulation environment and displays this behavior in real time. A genetic algorithm controls the evolution of the agents. Students manipulate parameters with a graphical user interface and plot outputs using standard utilities. The software supports an inquiry cycle that has been piloted in a course titled "Biocomputational Developmental Ecology" at Hampshire College [35]. A screen shot of the SuperDuperWalker simulation is shown in Fig. 4.9.

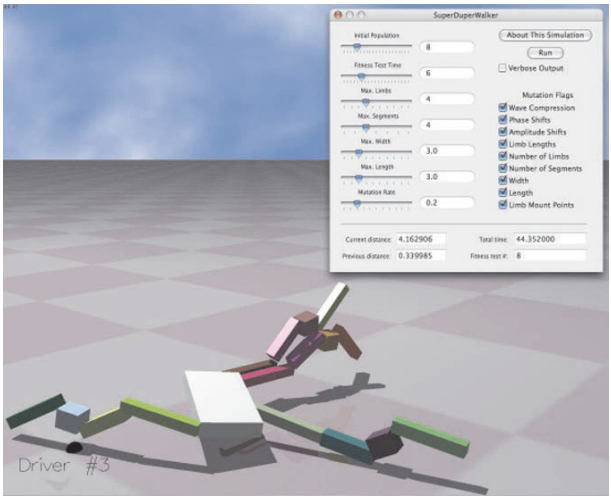

Fig. 4.9 The breve SuperDuperWalker simulation.

4.7.4 Artificial Intelligence in 3D Virtual Worlds

Breve has been used as the foundation for a new approach to the introductory artificial intelligence curriculum, which has been presented as a course twice (as of this writing) at Hampshire College under the title "Artificial Intelligence in 3D Virtual Worlds." This course covers roughly the same material as

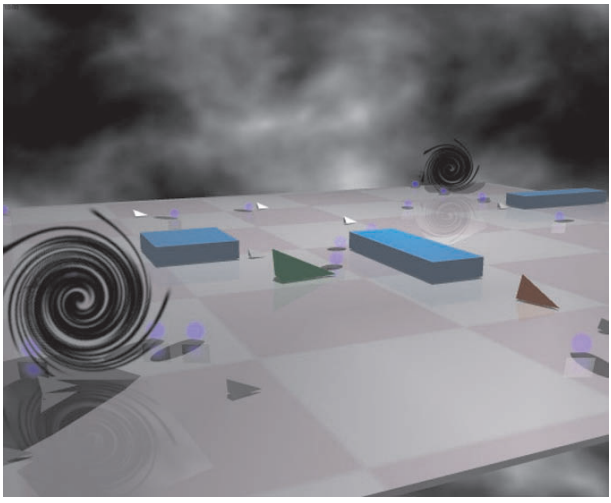

Fig. 4.10 The breve WubWorld simulation.

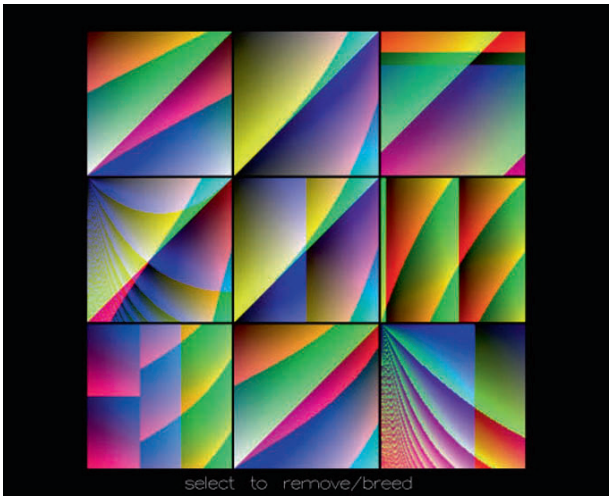

Fig. 4.11 The breve PushImageEvolve simulation.

is covered in other "agent-based" introductory AI courses, with an emphasis on reactive agents, neural networks, and evolutionary computation. Traditional AI topics such as knowledge representation, heuristic search, planning, and logic-based approaches are also covered but in less detail. The primary novelty of the course is that students work within breve, creating programs to control agents in engaging, dynamic, and visually rich virtual worlds. These virtual worlds include "WUB World," an environment inhabited by monsters (Wildly Unpredictable Biots, or WUBs), agents, obstacles, worm-holes, and energy sources, and a world in which teams of agents compete in a "capture the flag" game. The materials for this course, including the breve simulation files, are available online [29]. A screen shot of the WubWorld simulation is shown in Fig. 4.10.

4.7.5 Algorithmic Art

Breve's rich visualization subsystem is useful not only for developing and observing simulations but also for work in the computational arts. An "Algorithmic Arts" course, taught at Hampshire College (twice as of this writing), has used breve to provide students with tools for dynamically manipulating images, lines, shapes, and color in 3D space. The integration of these tools with breve allowed students to develop portfolios of algorithmic artwork that employed turtle graphics, Lindenmayer systems, iterated function systems, reaction/diffusion systems, texture mapping, 3D simulation, and interactive

genetic algorithms. Fig. 4.11 shows a screen shot of the PushImageEvolve simulation for interactive image evolution, which allows the user to evolve image-generating Push programs using a simple point-and-click interface.

4.8 Conclusion

Breve is a powerful, free software package for multi-agent simulation. With features such as realistic physical simulation, real-time 3D visualization, support for evolutionary computation, and integration with powerful scripting languages, breve greatly simplifies the rapid development and exploration of advanced artificial life experiments. We have successfully applied breve to the development of multi-agent systems to study artificial life and a wide variety of other fields.

References

1. Barnum, H., Bernstein, H.J., Spector, L.: Quantum circuits for OR and AND of ORs. Journal of Physics A: Mathematical and General **33**(45), 8047–8057 (2000)
2. Bhalla, N., Bentley, P.J., Jacob, C.: Mapping virtual self-assembly rules to physical system. In: A. Adamatzky, L. Bull, B.D.L. Costello, S. Stepney, C. Teuscher (eds.) Proceedings of the 2007 Conference on Unconventional Computing. Luniver Press, Beckington (2007)
3. Braitenberg, V.: Vehicles: Experiments in Synthetic Psychology. The MIT Press, Cambridge, MA (1986)
4. Cohen, P.R., Morrison, C.T., Cannon, E.: Maps for verbs: The relation between interaction dynamics and verb use. In: L.P. Kaelbling, A. Saffiotti (eds.) Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), pp. 1022–1027. Professional Book Center, Edinburgh (2005)
5. Don, O., Amos, M.: An ant-based algorithm for annular sorting. ArXiv Nonlinear Sciences e-prints (2005)
6. Hamann, H., Wörn, H.: An analytical and spatial model of foraging in a swarm of robots. In: E. Sahin, W.M. Spears, A.F.T. Winfield (eds.) Swarm Robotics, Second International Workshop, SAB 2006, Rome, Italy, September 30-October 1, 2006, pp. 43–55. Springer, Berlin (2006)
7. Klein, J.: breve website. URL `http://www.spiderland.org/`
8. Klein, J., Spector, L.: Unwitting distributed genetic programming via asynchronous javascript and XML. In: H. Lipson (ed.) GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1628–1635. ACM Press, New York (2007)
9. Komosinski, M., Ulatowski, S.: Framsticks – artificial life. In: C. Nédellec, C. Rouveirol (eds.) ECML 98 Demonstration and Poster Papers, pp. 7–9. Chemnitzer Informatik-Berichte, Chemnitz (1998)
10. Kriplean, T.L.: Evolving an ecology of two-tiered organizations. In: F. Rothlauf (ed.) GECCO '05: Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation, pp. 402–406. ACM Press, New York (2005)
11. Lassabe, N., Luga, H., Duthen, Y.: A New Step for Evolving Creatures. In: IEEE-ALife'07, Honolulu, Hawaii, 01/04/2007-05/04/2007, pp. 243–251. IEEE Press (2007)

12. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation **81**(7), 517–527 (2005)
13. Minar, N., Burkhart, R., Langton, C., Askenazi, M.: The swarm simulation system, a toolkit for building multi-agent simulations. Tech. Rep. 96-06-042, Sante Fe Institute (1996)
14. Mirtich, B.: V-clip: Fast and robust polyhedral collision detection. ACM Transactions on Graphics **17**(3), 177–208 (1998)
15. Mirtich, B., Canny, J.F.: Impulse-based simulation of rigid bodies. In: SI3D '95: Proceedings of the 1995 Symposium on Interactive 3D Graphics, pp. 181–188, 217. ACM Press, New York (1995)
16. North, M., Macal, C.: Escaping the accidents of history: An overview of artificial life modeling with repast. In: A. Adamatzky, M. Komosinski (eds.) Artificial Life Models in Software, pp. 115–142. Springer-Verlag New York, Secaucus (2006)
17. Resnick, M.: Starlogo: an environment for decentralized modeling and decentralized thinking. In: CHI '96: Conference Companion on Human Factors in Computing Systems, pp. 11–12. ACM Press, New York (1996)
18. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics **24**(4), 25–34 (1987)
19. Riolo, R.L., Cohen, M.D., Axelrod, R.: Evolution of cooperation without reciprocity. Nature **414**, 441–443 (2001)
20. Sims, K.: Evolving virtual creatures. In: Proceedings of SIGGRAPH '94 (Orlando, Florida, July 1994), pp. 15–22 (1994)
21. Spector, L.: Autoconstructive evolution: Push, pushGP, and pushpop. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pp. 137–146. Morgan Kaufmann, San Francisco (2001)
22. Spector, L.: Adaptive populations of endogenously diversifying pushpop organisms are reliably diverse. In: R. Standish, M.A. Bedau, H.A. Abbass (eds.) Proceedings of the Eighth International Conference on Artificial Life, pp. 142–145. MIT Press, Cambridge, MA (2002)
23. Spector, L.: Automatic Quantum Computer Programming: A Genetic Programming Approach. Kluwer Academic Publishers, Boston (2004)
24. Spector, L., Barnum, H., Bernstein, H.J.: Genetic programming for quantum computers. In: Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 365–373. Morgan Kaufmann, Madison, WI (1998)
25. Spector, L., Barnum, H., Bernstein, H.J., Swamy, N.: Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In: P.J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala (eds.) Proceedings of the Congress on Evolutionary Computation, vol. 3, pp. 2239–2246. IEEE Press, Washington (1999)
26. Spector, L., Barnum, H., Bernstein, H.J., Swamy, N.: Quantum computing applications of genetic programming. In: L. Spector, W.B. Langdon, U.M. O'Reilly, P.J. Angeline (eds.) Advances in Genetic Programming 3, chap. 7, pp. 135–160. MIT Press, Cambridge, MA (1999)
27. Spector, L., Bernstein, H.J.: Communication capacities of some quantum gates, discovered in part through genetic programming. In: J.H. Shapiro, O. Hirota (eds.) Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing (QCMC), pp. 500–503. Rinton Press, Princeton (2003)
28. Spector, L., Clark, D.M., Lindsay, I., Barr, B., Klein, J.: Genetic programming for finite algebras. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation. ACM Press, London (2008)
29. Spector, L., Klein, J.: CS 263: Artificial Intelligence in 3D Virtual Worlds. URL `http://hampshire.edu/lspector/cs263/cs263s04.html`
30. Spector, L., Klein, J.: Evolutionary dynamics discovered via visualization in the breve simulation environment. In: Workshop Proceedings of the 8th International Conference on the Simulation and Synthesis of Living Systems. University of New South Wales (2002)

31. Spector, L., Klein, J.: Trivial geography in genetic programming. In: T. Yu, R.L. Riolo, B. Worzel (eds.) Genetic Programming Theory and Practice III, vol. 9, chap. 8, pp. 109–123. Springer, New York (2005)
32. Spector, L., Klein, J.: Genetic stability and territorial structure facilitate the evolution of tag-mediated altruism. Artificial Life **12**(4), 1–8 (2006)
33. Spector, L., Klein, J.: Multidimensional tags, cooperative populations, and genetic programming. In: R.L. Riolo, T. Soule, B. Worzel (eds.) Genetic Programming Theory and Practice IV, vol. 5, chap. 15, pp. 97–112. Springer, New York (2006)
34. Spector, L., Klein, J., Feinstein, M.: Division blocks and the open-ended evolution of development, form, and behavior. In: H. Lipson (ed.) Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 316–323. ACM Press, New York (2007)
35. Spector, L., Klein, J., Harrington, K., Coppinger, R.: Teaching the evolution of behavior with superduperwalker. In: C.K. Looi, G.I. McCalla, B. Bredeweg, J. Breuker (eds.) Proceedings of the 12th International Conference on Artificial Intelligence in Education, pp. 923–925. IOS Press, Washington (2005)
36. Spector, L., Klein, J., Keijzer, M.: The push3 execution stack and the evolution of control. In: H.G. Beyer, U.M. O'Reilly (eds.) GECCO 2005: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1689–1696. ACM Press, Washington (2005)
37. Spector, L., Klein, J., Perry, C.: Tags and the evolution of cooperation in complex environments. In: Proceedings of the AAAI 2004 Symposium on Artificial Multiagent Learning. AAAI Press, Melno Park, CA (2004)
38. Spector, L., Klein, J., Perry, C., Feinstein, M.: Emergence of collective behavior in evolving populations of flying agents. In: E. Cantu-Paz, J.A. Foster, K. Deb, L.D. Davis, R. Roy, U.M. O'Reilly, H.G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K.A. Dowsland, N. Jonoska, J. Miller (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2724, pp. 61–73. Springer-Verlag, Berlin (2003)
39. Spector, L., Klein, J., Perry, C., Feinstein, M.: Emergence of collective behavior in evolving populations of flying agents. Genetic Programming and Evolvable Machines **6**(1), 111–125 (2005)
40. Spector, L., Perry, C., Klein, J., Keijzer, M.: Push 3.0 programming language description. Tech. Rep. HC-CSTR-2004-02, School of Cognitive Science, Hampshire College (2004)
41. Spector, L., Robinson, A.: Genetic programming and autoconstructive evolution with the push programming language. Genetic Programming and Evolvable Machines **3**(1), 7–40 (2002)
42. Szymanski, M., Breitling, T., Seyfried, J., Wörn, H.: Distributed shortest-path finding by a micro-robot swarm. In: M. Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Sttzle (eds.) Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings, vol. 4150, pp. 404–411. Springer, Berlin (2006)
43. Veeraswamy, A., Amavasai, B.: Optimal path planning using an improved a * algorithm for homeland security applications. In: AIA'06: Proceedings of the 24th IASTED International Conference on Artificial Intelligence and Applications, pp. 50–55. ACTA Press, Anaheim, CA (2006)
44. Wilson, G.C., McIntyre, A., Heywood, M.I.: Resource review: Three open source systems for evolving programs–lilgp, ECJ and grammatical evolution. Genetic Programming and Evolvable Machines **5**(1), 103–105 (2004)