Agent-based Modeling

Adam Getchell
Department of Physics
University of California, Davis
acgetchell@ucdavis.edu

June 9, 2008

Abstract

The theory and practice of agent-based modeling is reviewed, and agent-based modeling toolkits are evaluated and discussed. A tractable selection of toolkits, RepastPy, Repast Simphony, and breve are then employed to develop and visualize a series of increasingly sophisticated agent-based models, starting with a simple network-interaction diagram and proceeding onto the Boids 3D flock simulation, a 3D collision and gravity system, the chaotic Gray Scott diffusion reaction, a sophisticated agent behaviors game of Capture the Flag, finally culminating with an complex Boids evolutionary swarm simulation. To accomplish the latter, genetic programming techniques are briefly reviewed. Finally, an overview is presented with future directions.

Introduction

The use of computer systems to solve problems of interest in physics, biology, chemistry, economics, and social sciences has been well-established for decades. The great advances in computing power, software development, computer graphics, communications networks, and a host of other technologies have elevated the domain of applicability to problem solving from simple arithmetic calculations to advanced numerical methods and the creation of large simulations solving myriad systems of complex equations in real-time.

A recent development has been the introduction of the Agent-Based Modeling paradigm, which has proven useful for a wide range of problems in physics and chaotic attractors [1], biology [2], economics [3], social science [4], and geospatial simulations [5]. Commensurate with this new paradigm has been the development of a range of toolkits to apply Agent-Based modeling to a particular range of problems.

This purpose of this paper is to provide a general introduction to Agent-Based modeling and demonstrate its use in the modeling of a chaotic system. Organization is as follows: Section I begins with a general discussion of Agent-Based Modeling (hereafter ABM), including definitions and characteristics of systems amenable to this approach. Section II will provide an up-to-date review of the various Swarm modeling toolkits currently available at this time of writing. Section III will show the use of a particular toolkit, Repast, using both the simplified RepastPy interface and the more complex RepastS integrated development environment (IDE), and cover the development of a simple swarm model. Section IV will cover the simulation of the chaotic system _______, and Section V will wrap-up with conclusions and further references.

Section I - Overview

Following the general overview by Castle and Crooks [6] and elsewhere [7], a dynamic model is defined as a simplified representation of reality that evolves over time. The unstated art behind this method is the intuition/experience in creating a simulation that is sufficiently complex enough to show interesting dynamics of the system, while being simple enough to have a tractable software representation on a computing system.

In particular, "dynamics" is defined in as the study of change and evolving systems [8], while "tractable" is an artifact of the particular tool used. As we shall see, ABM makes particular classes of problems very easy to solve, that would otherwise be very difficult to attack using more traditional dynamical methods.

For the purposes of this paper, an Agent, as part of an ABM system, is defined with the following characteristics:

- Activity: Each agent independently acts according the rules of the simulation and their own pre-programmed behaviors. These rules and behaviors can take one or more of the following features:
    - Goal-direction: The agent acts in such a way as to achieve a particular goal, which can be either a relative or extremal value. For example, an agent may be designed to maximize accumulation of a particular resource.
    - Reactivity/Perceptivity: The agent senses its surroundings, or is supplied with a map such that it is aware of its environment. For example, an agent could be aware of resource node locations.
    - Bounded Rationality: Generally, goal-direction in agents operates on the rational-choice principle, which generally implies unlimited access to information and computational resources. However, experimental evidence suggests that non-optimal decisions are often closer to reality. Therefore, in order to provide greater predictive power, the agents can be constrained in terms of information resources or analytical ability. For example, an agent might be able to sense only those resource nodes within a finite range, or possess a map of resources that does not take into account the actions of other agents.
    - Interactivity: Continuing on the principle of bounded rationality, agents may interact or exchange information with other agents. These interactions may have particular effects on the agent, including its destruction or change in goal-seeking behavior.
    - Mobility: Interactivity with the environment and other agents is vastly improved if the agent can roam the model space independently.
    - Adaptation: Alteration of an agent's current state based upon interactions with the environment or other agents provide a useful form of learning or memory. This adaptation can be provided for at the level of the individual agent, or groups of nearby agents, all the way up to the population level of the entire set of agents .

- Autonomy: Each agent is free for activity as defined above, with the ability to make independent decisions.

- Heterogeneity: Although each agent may begin as a member of a limited set of common templates, develops individuality through autonomous activity in the sense describe previously.

Agents need a stage for their behaviors, and this is taken as the definition of the Environment. Although the environment may itself change dynamically according to the actions of the agents, these changes occur passively, rather than in the active fashion of agent time evolution. That is, the state of the environment evolves dynamically, but only in response to the actions of the agents, rather than as a result of particular goal-seeking or adaptive behavior.

As an example, given an environment populated with resource nodes and a population of agents each looking to maximize intake of resources, the agents alter (consume) the distribution of resources, while the environment passively adjusts resource distribution based upon agent action. Granted, more complex environments are possible which dynamically alter their own resource distribution, but this alteration is typically given by a simple rule, rather than the result of goal-seeking behavior.

(Note that more complex environments can be modeled in turn by the addition of a new group of agents to act as the “resources”).

In sum, agents are active, while the environment is passive.

As a final note, Agent Based Modeling (with particular distinctions) can also be found under the terms Agent-Based Computational Modeling [9], Agent-Based Social Simulation [4], Multi-Agent systems [10] [11], Distributed Artificial Intelligence [12] [13], and Swarm Intelligence [14] [15][16]. However, Agent Based Modeling and Swarm Intelligence appear to be the more contemporary of the terms used in the literature.

Section II – Agent Based Modeling tools

There are good summaries in the literature which compare and contrast the various ABM tools [17] [18]. However, they are somewhat dated in terms of the versions of the tools that were reviewed. For example, de Smith et. al., though updated in 2008, referred to versions of Repast from 2006.

The rapid pace of software, therefore, warranted a fresh look at the current state of the art.

Given the previous reviews of ABM frameworks and libraries, this paper considers the following software:

- Swarm [19] [20] – Latest stable release is Swarm 2.2 released in February 2005.

- MASON [21] – Latest stable release is MASON Version 12, released in the July 2007 timeframe.

- NetLogo [22] – Latest stable release is NetLogo version 4.0.2, released December 5th, 2007.

- RePast [23] – Latest stable release is Repast Simphony 1.0 released December 3rd, 2007.

- MetaABM [24] – Latest stable release is
- Breve [25] – Latest stable release is 2.7.2 released February 19th, 2008.

With the exception of NetLogo, all of these frameworks are both open source (hence, free for use and source code inspection) and based upon an object-oriented language (usually Java). This is due in large part because object oriented frameworks and programming is a natural fit for ABM.

Swarm was originally written in Objective-C, and then ported to Java. Written as a library and framework of simulation tools, rather than as a finished application per se, Swarm is one of the oldest agent-based modeling toolkits, and there are hundreds of example applications and demos, and several of the newer toolkits are based upon it. However, given the age of the last stable release, and the existence of newer toolkits with more friendly environments (especially Repast), Swarm is not considered further in this paper. Nonetheless, the documentation and research papers on Swarm established many of the foundational concepts and ideas in ABM, and reading over these materials serves as an excellent introduction to the large and growing field of agent-based modeling.

MASON, or Multi-Agent Simulator of Neighborhoods/Networks is a multiagent simulation library in Java, designed to serve as the base class structure for custom Java simulations. It also includes a model library and suite of 2D and 3D visualization tools, and is developed with an emphasis on speed and portability. Although well-regarded, at the time of this writing, the Windows batch files for starting and running MASON 12 did not work on Windows Vista 64-bit, so MASON is not considered further in this paper.

NetLogo, though not open source in the strictest sense, is freeware, and designed for educational use, being based upon a simple Logo-type language. Originally developed in 1999 by Uri Wilensky, NetLogo has been under continuous development since then, and has a large and extensive user community with lists of community models. NetLogo is not considered in this paper primarily due to its use of a non-standard programming language (Logo), which makes it more difficult to integrate with other toolsets and programs using object oriented programming languages more typically found in research (e.g. Java or Python).

Repast is based upon Swarm, but implemented in Java from the ground up. Repast has several versions available; the current standard Repast Java version is version 3.

A simplified version of Repast, RepastPy was produced that introduced a friendly GUI and the use of a subset of Python as a scripting language. RepastPy is quicker and easier to use than Repast, and is generally recommended by the Repast community as being a good version for prototyping models (leveraging the Python language characteristics of simplicity and productivity). Behind the scenes, the Python scripts generate Java objects.

A version of Repast based on the .NET runtime, Repast.NET was made. The .NET runtime is flexible and powerful, and has a large number of useful libraries for doing almost anything, as well as an elegant successor language to Java, C#, and the ability to run any language that can be targeted on the .NET platform (including Visual Basic, Python, Ruby, and F# to name a few). Unfortunately, Repast.NET was written against the .NET 1.1 runtime, and appears not to have been updated since. The software source code contains project files which are not compatible with later versions of Visual Studio (Visual Studio 2008 was tried), and so extensive troubleshooting to correct this issue was skipped in favor of using other versions of Repast.

Repast Simphony is the latest version of Repast, combining the powerful Eclipse integrated development environment with automated connectors to additional tools such as R, VisAD, Weka, MATLAB, and iReport. This paper will showcase both RepastPy and Repast Simphony.

MetaABM is a powerful agent-based modeling system written by Miles Parker and including a neutral, standards-based representational format using the Eclipse Modeling Framework. MetaABM also features a powerful GUI environment based on Eclipse which can export models to other environments, including Repast Simphony and Ascape (another Swarm successor).

Section III – Results – Repast, RepastPy, Repast Simphony, and Breve

All toolkits and software listed above, with the exception of Swarm (due to the existence of newer systems) and NetLogo (due to its Logo foundations) were installed and run using two machines: a Dell D830 Laptop with 2.4 GHz Core 2 Duo with 4GB of RAM, and a MacBook running MacOSX 10.5.3.

Repast was initially designed to meet the needs for ABM in social sciences , including the use of geographic information systems. Repast Simphony offers a more robust toolset for 2 and 3D lattice simulations in the physical sciences.

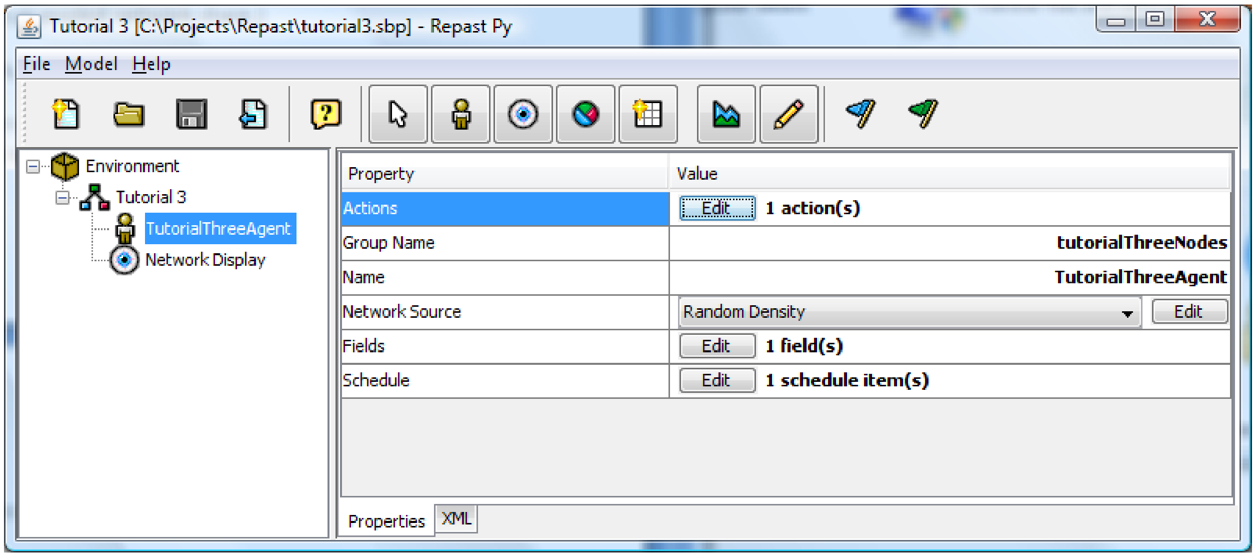
Repast was chosen for several reasons:

- Ability to use several languages (Python, Java, .NET)
- Quality of documentation
- Quality of Repast Simphony IDE
    - Currently uses Groovy, a dynamic language for the Java Virtual Machine that compiles to Java byte-codes, but offers features like Domain-Specific Language support, annotations, generics, and meta-programming. [26]
    - Ability to “freeze-dry” simulations. Models can be run, serialized to a flat file structure (similar to Python’s pickle), and then re-hydrated to pick up where the simulation left off on a different system.
    - Built-in support for the export of data to movies, VisAd (interactive visualization toolkit) [27], Weka (machine learning toolkit) [28], or MatLab.
- Currency of Implementation – In particular, Simphony installs on current versions of operating systems such as Windows Vista. (Although RepastPy must run as Administrator, which presents issues of file storage in that the user must be able to access the same directory used by the Administrator to save the model, and the default Documents directory is the Administrators’

Document directory, hence inaccessible to the typical user. Likewise, networked user directories are unavailable to the Administrator without an explicit network share.)

Given the heavy Python use in current class work, the first model chosen was done using RepastPy.

However, RepastPy quickly showed limitations in the original design slant towards social science modeling. For example, only 2D grids could be generated. Nevertheless, a nicely illustrative network simulation model showing agent based interactions was created and run in a very short time.



Property	Value
Actions	Edit 1 action(s)
Group Name	tutorialThreeNodes
Name	TutorialThreeAgent
Network Source	Random Density Edit
Fields	Edit 1 field(s)
Schedule	Edit 1 schedule item(s)

Figure 1: RepastPy network model setup screen

```
if (self.getNumOutEdges()):

        otherAgent = (TutorialThreeAgent)self.getRandomNodeOut()

        otherWealth = otherAgent.getWealth()

        if (otherWealth > self.wealth and otherWealth > 2):

                self.wealth = self.wealth + 2

                otherAgent.setWealth(otherWealth - 2)

        else:

                self.removeEdgesTo(otherAgent)

                otherAgent.removeEdgesFrom(self)

                self.makeRandomOutEdge(self.model.getAgentList(),
        DefaultDrawableEdge(), false)

else:

        self.makeRandomOutEdge(self.model.getAgentList(), DefaultDrawableEdge(), false)

self.setNodeLabel(String.valueOf(self.wealth))
```

Figure 2: RepastPy agent actions behavior using Python scripting

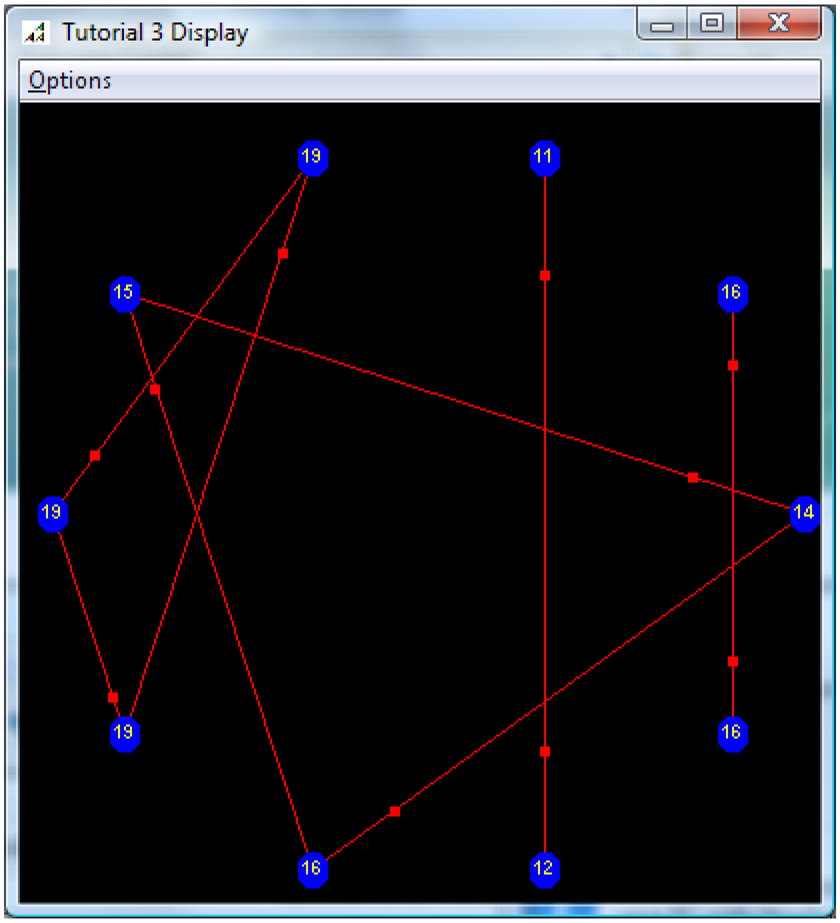


Figure 3: A RepastPy network interaction simulation

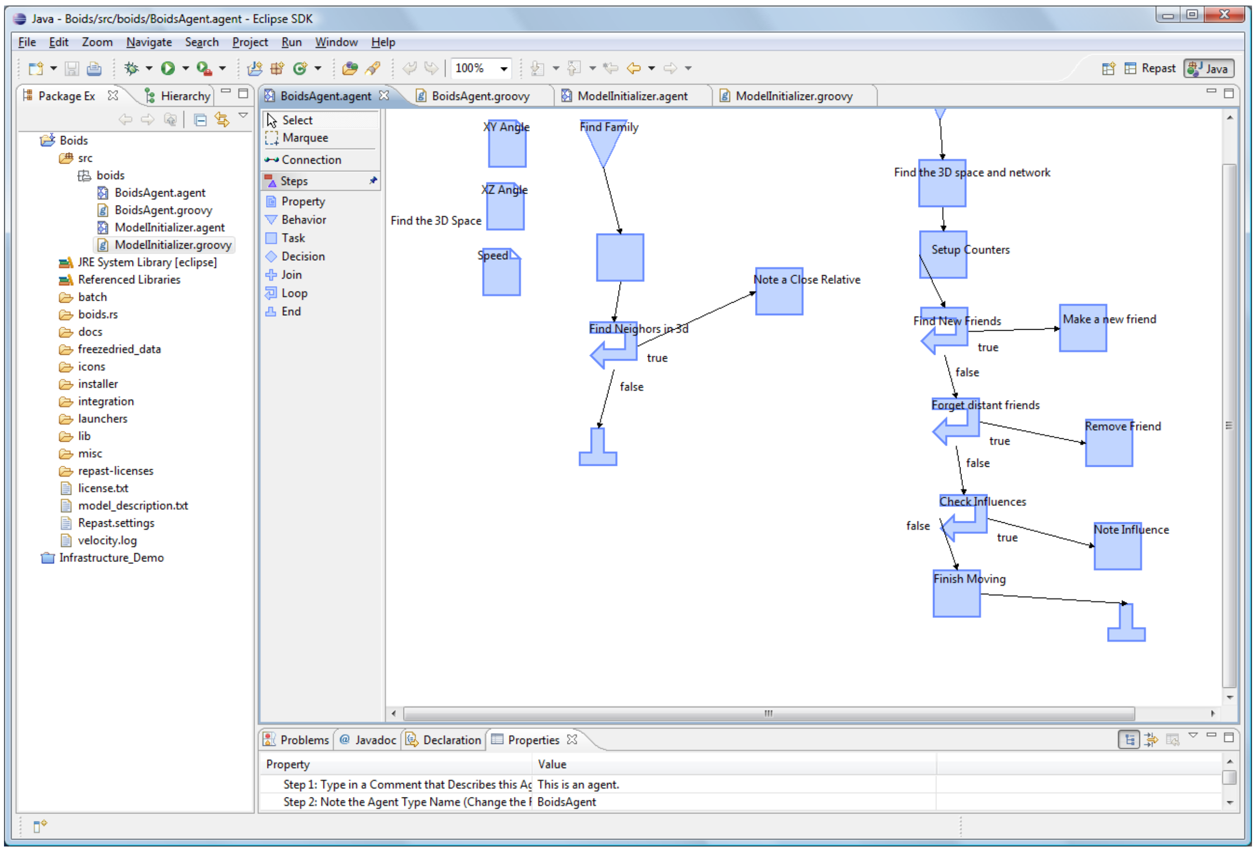
Repast Simphony was employed for the next model.

However, working through the basic tutorials generated several errors that contradicted the Repast documentation. Repast Simphony is an aspect of the Eclipse Java IDE, which proved to be a blessing and a curse. For example, Repast Simphony is oriented towards Eclipse Europa (v 3.3), but on MacOSX 10.5 there are bugs and other JRE issues. These can be addressed by running Eclipse Ganymede (v3.4), but then documentation doesn't cover installation, and there are issues loading the Repast libraries. On the Windows side, Repast Simphony + Eclipse Europa in the bundle have subtle bugs in loading of libraries, and loading the full Ecplipse Ganymede Software Architects version causes load and out of memory issues with unrelated UML parsers. The best approach on both platforms seems to be loading a minimal Eclipse 3.4, and then loading the bare minimum libraries needed. Needless to say, this fussing about with various versions was quite time-consuming (several bugs were filed), so other solutions were looked at even as a model was developed.

Repast .NET – This would be excellent, and the author has extensive experience with Visual Studio and .NET. Unfortunately, Repast .NET used the old version 7 solution and project files (Visual Studio 2003); when loaded into Visual Studio 2008 importing the project into a version 9 solution left out the repast.csproj – the very libraries needed to use Repast.NET. Needless to say all the test cases failed.

Finally, a model was developed in Repast Simphony. This simulation is the classic Boids model developed by Craig Reynolds in 1986 [29], which is a kind of 3D Life simulation producing chaotic behavior with the following rules:

1. Boids try to fly towards the center of mass of neighboring Boids (usually, the perceived CoM with respect to that particular Boid)
2. Boids try to keep a small distance away from other objects (including other Boids)
3. Boids try to match velocity with nearby Boids (perceived velocity of neighbors)



Property	Value
Step 1: Type in a Comment that Describes this Ag	This is an agent.
Step 2: Note the Agent Type Name (Change the I	BoidsAgent

Figure 4: Visual model of Boids agent

```
/**
 *
 * This file was automatically generated by the Repast Simphony Agent Editor.
 * Please see http://repast.sourceforge.net/ for details.
 *
 */

/**
 *
 * Set the package name.
 *
 */
package boids
```

```
/**
 *
 * Import the needed packages.
 *
 */
import java.io.*
import java.math.*
import java.util.*
import javax.measure.unit.*
import org.jscience.mathematics.number.*
import org.jscience.mathematics.vector.*
import org.jscience.physics.amount.*
import repast.simphony.context.*
import repast.simphony.context.space.continuous.*
import repast.simphony.context.space.gis.*
import repast.simphony.context.space.graph.*
import repast.simphony.context.space.grid.*
import repast.simphony.engine.environment.*
import repast.simphony.engine.schedule.*
import repast.simphony.engine.schedule.*
import repast.simphony.engine.watcher.*
import repast.simphony.groovy.math.*
import repast.simphony.integration.*
import repast.simphony.matlab.link.*
import repast.simphony.parameter.*
import repast.simphony.random.*
import repast.simphony.space.continuous.*
import repast.simphony.space.gis.*
import repast.simphony.space.graph.*
import repast.simphony.space.projection.*
import repast.simphony.ui.probe.*
import simphony.util.messages.*
import static java.lang.Math.*
import static repast.simphony.essentials.RepastEssentials.*

/**
 *
 * This is an agent.
 *
 */
public class BoidsAgent  {

    /**
     *
     * This is an agent property.
     * @field xyAngle
     *
     */
    @Parameter (displayName = "XY Angle", usageName = "xyAngle")
    public double getXyAngle() {
        return this.xyAngle
    }
    public void setXyAngle(double finalValue) {
        this.xyAngle = finalValue
    }
    public double xyAngle = 0
```

```
/**
 *
 * This is an agent property.
 * @field xzAngle
 *
 */
@Parameter (displayName = "XZ Angle", usageName = "xzAngle")
public double getXzAngle() {
    return this.xzAngle
}
public void setXzAngle(double finalValue) {
    this.xzAngle = finalValue
}
public double xzAngle = 0

/**
 *
 * This is an agent property.
 * @field speed
 *
 */
@Parameter (displayName = "Speed", usageName = "speed")
public double getSpeed() {
    return this.speed
}
public void setSpeed(double finalValue) {
    this.speed = finalValue
}
public double speed = 0

/**
 *
 * This value is used to automatically generate agent identifiers.
 * @field serialVersionUID
 *
 */
private static final long serialVersionUID = 1L

/**
 *
 * This value is used to automatically generate agent identifiers.
 * @field agentIDCounter
 *
 */
protected static long agentIDCounter = 1

/**
 *
 * This value is the agent's identifier.
 * @field agentID
 *
 */
protected String agentID = "BoidsAgent " + (agentIDCounter++)

/**
 *
```

```
     * This is the family finding behavior.
     * @method findFamily
     *
     */
    @ScheduledMethod(
        start = 0d,
        priority = -1.7976931348623157E308d,
        shuffle = true
    )
    public void findFamily() {

        // Note the simulation time.
        def time = GetTickCountInTimeUnits()

        // Use the Repast Simphony Groovy math tools.
        use (MathOperations.mathCategories()) {

            // Find the 3D space
            Context context = FindContext("Boids")
            ContinuousSpace space = (ContinuousSpace)
context.getProjection("Space")
            setXyAngle RandomDraw(0, 2 * Math.PI)
            setXzAngle RandomDraw(0, 2 * Math.PI)
            setSpeed RandomDraw(-2, 2)

            // Make a decision.
            for (neighbor in (new
repast.simphony.query.space.continuous.ContinuousWithin(space, this,
25).query())) {

                // Link to neighbors
                CreateEdge("Boids/Family", this, neighbor, 1.0)
                CreateEdge("Boids/Friends", this, neighbor, 1.0)

            }


            // Exit this scope.
            return

        }

        // End the method.
        return

    }

    /**
     *
     * Move Behavior
     * @method move
     *
     */
    @ScheduledMethod(
        start = 1d,
        interval = 1d,
        shuffle = true
    )
    public void move() {

        // Note the simulation time.
        def time = GetTickCountInTimeUnits()

        // Use the Repast Simphony Groovy math tools.
        use (MathOperations.mathCategories()) {

            // space and network finding
            ContinuousSpace space =
(ContinuousSpace)FindProjection("Boids/Space")
            Network family = (Network)FindProjection("Boids/Family")
            Network friends = (Network)FindProjection("Boids/Friends")
            Context context = FindContext("Boids")
            // temp counter defintion
            def counter = 1
            def tempXYAngle = xyAngle
            def tempXZAngle = xzAngle
            def tempSpeed = speed

            // Make a decision.
            for (friend in (new repast.simphony.query.AndQuery(new
repast.simphony.query.space.continuous.ContinuousWithin(space, this, 20), new
repast.simphony.query.NotQuery(context, new
repast.simphony.query.space.graph.NetworkSuccessor(friends, this))).query()))
{

                // New Friend
                CreateEdge("Boids/Friends", this, friend, 1.0)

            }


            // Make a decision.
            for (friend in (new repast.simphony.query.AndQuery(new
repast.simphony.query.NotQuery(context, new
repast.simphony.query.space.continuous.ContinuousWithin(space, this, 40)),
new repast.simphony.query.space.graph.NetworkSuccessor(friends,
this)).query())) {

                // remove friend
                RemoveEdge("Boids/Friends", this, friend)

            }


            // Make a decision.
            for (influence in (new repast.simphony.query.OrQuery(new
repast.simphony.query.space.graph.NetPathWithin(family,this, 2), new
repast.simphony.query.space.graph.NetPathWithin(friends, this, 3)).query()))
{

                // This is a task.
                counter = counter + 1
                tempXYAngle = tempXYAngle + influence.xyAngle
                tempXZAngle = tempXZAngle + influence.xzAngle
```

```
                    tempSpeed = tempSpeed + influence.speed

                }

                // This is a task.
                tempXYAngle = tempXYAngle / counter
                tempXZAngle = tempXZAngle / counter
                setSpeed min (max(tempSpeed / counter + RandomDraw(-0.5, 0.5), -
2), 2)
                space.moveByDisplacement(this, speed * tempXYAngle, speed *
tempXYAngle, speed * tempXZAngle)

                // Exit this scope.
                return

            }

            // End the method.
            return

        }

        /**
         *
         * This method provides a human-readable name for the agent.
         * @method toString
         *
         */
        @ProbeID()
        public String toString() {

            // Define the return value variable.
            def returnValue

            // Note the simulation time.
            def time = GetTickCountInTimeUnits()

            // Use the Repast Simphony Groovy math tools.
            use (MathOperations.mathCategories()) {

                // Set the default agent identifier.
                returnValue = this.agentID

            }

            // Return the results.
            return returnValue

        }


}
```

Figure 5: Boids agent source code (in Groovy)

As can be seen from a comparison of Figure 4 and Figure 5, model building with the GUI is easier. Finally, here are the visual results of the Boids simulation compiled as a Java/Repast application, with the model data initialized. Note the connecting lines between Boids represent affinity (either Friend or Relative) with other agents in the simulation.

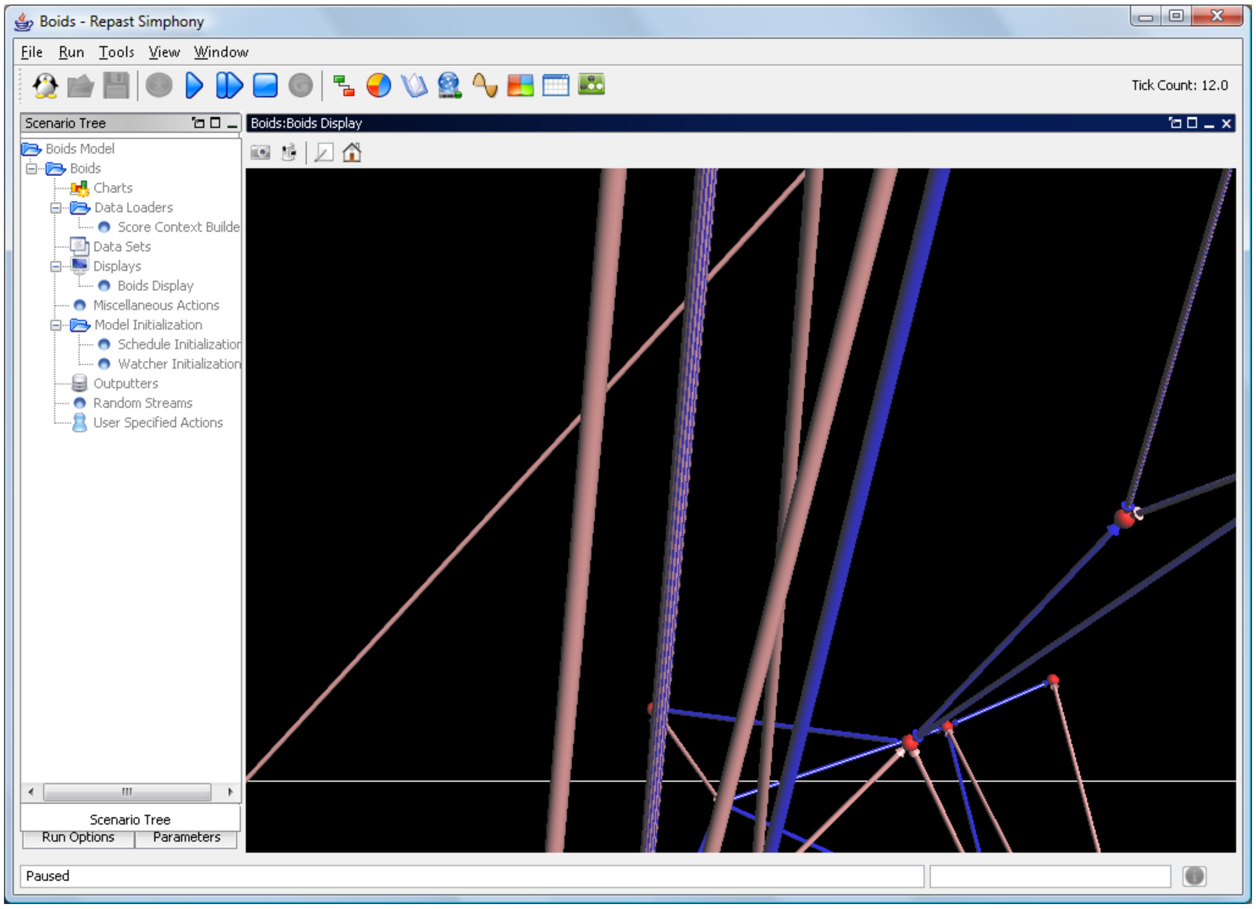


Figure 6: Boids model with network graph representing affinity between various Boids.

MetaABM – After encountering the various Repast Simphony issues, MetaABM looked like a nice, elegant solution. It suffered some of the same types of Eclipse issues as Repast Simphony, as MetaABM is actually loaded on top of Repast (as it can emit models for use with the Repast framework). Getting a working version could only be done using the all-in-one bundled distribution, which then suffered maintenance issues with Eclipse itself. The model building is very heavily GUI oriented, which is a disadvantage when subtle bugs were discovered in the documentation (e.g., the basic test model would not actually work). Troubleshooting these issues meant taking screenshots of the GUI and posting them to a discussion board, which ironically had quite severe limits on both the data and pixel size of graphics. In the end, MetaABM required too much setup/troubleshooting time, and to complete work on time efforts focused on other toolkits.

Breve – The author wishes to have found this earlier. Although the modeling is strictly a 3D simulation for multi-agents (hence lacks some of the more sophisticated network or more general mathematical models that can be done with Repast), what it does it does well. Scripts can be separately loaded, in a proprietary scripting language, steve, or much better, using Python (though only v2.3 at present). Scripting environment limited compared to other Python IDEs, but it is efficient in loading.

Breve has some limitations. As mentioned, it is only suitable for 3D modeling. It also has certain magic methods, init() and iterate() that must be called to initialize agents and perform actions. Unlike Repast, you cannot set variable steps to do actions, they always happen each time. To get around computational issues, there is a separate post-iterate() method.

```
import breve

class HelloWorld( breve.Control ):

        def __init__( self ):

                breve.Control.__init__( self )

        def iterate( self ):

                print '''Hello, world!'''

                breve.Control.iterate( self )

breve.HelloWorld = HelloWorld

# Create an instance of our controller object to initialize the simulation

HelloWorld()
```

Figure 7: Basic breve controller/agent model structure using Python

```
@include "Control.tz"

Controller HelloWorld.

Control : HelloWorld {

  + to iterate:

    print "Hello, world!".

        super iterate.

}
```

Figure 8: Basic breve controller/agent model structure using steve

On the plus side, it supports model serialization to XML, and networking to other servers. This allows a web interface to a breve simulation, using a very simple RESTful type method calls:

http://myserver:<port>/set-agent-color .2 .4 .6

Breve has a nice intuitive user interface, and comes with dozens of models with viewable source code. The UI is slightly more powerful on MacOS X due to inclusion of a Cocoa application which allows one to view and edit model settings for particular agents on the fly; this capability is not available in other versions of Breve. Documentation on the system is extensive, but the best form of documentation is the models themselves, which mostly come in both steve and Python versions for inspection. As mentioned, the text editor is serviceable but somewhat limited; it is possible to edit files in other programs, although they aren't able to be developed in breve by a simple save settings and run inherent with the basic editor. Still, breve was extremely powerful, easy to learn, and productive – a refreshing contrast to most of the other systems. The author was able to modify most of the simulations to save movie clips of the resulting runs with very little effort

If the model being developed is best represented using a 3D graphical display, breve is probably the best tool for the job. Future versions of breve will concentrate on the use of Python as the standard modeling language going forward. As can be seen from a comparison of Figure 7 and 8, at present steve has an advantage in brevity and expressiveness.

At last, onto doing some actual modeling!

The first model shows the action of gravity and 3D collisions upon a randomly generated collection of spherical agents.

```
# Note: this file was automatically converted to Python from the
# original steve-language source code.  Please see the original
# file for more detailed comments and documentation.


import breve

class Gravity( breve.PhysicalControl ):
      def __init__( self ):
            breve.PhysicalControl.__init__( self )
            self.theBall = None
            Gravity.init( self )

      def init( self ):
            self.setIntegrationStep( 0.000100 )
            breve.createInstances( breve.Step, 1 ).create( breve.vector( -
0.500000, 0, 0 ), breve.vector( 1.000000, 0.020000, 1 ) )
            breve.createInstances( breve.Step, 1 ).create( breve.vector(
0.200000, -0.200000, 0 ), breve.vector( 0.200000, 0.020000, 1 ) )
```

```
            breve.createInstances( breve.Step, 1 ).create( breve.vector( 
0.400000, -0.400000, 0 ), breve.vector( 0.200000, 0.020000, 1 ) )
            breve.createInstances( breve.Step, 1 ).create( breve.vector( 
0.600000, -0.600000, 0 ), breve.vector( 0.200000, 0.020000, 1 ) )
            breve.createInstances( breve.Step, 1 ).create( breve.vector( 
0.800000, -0.800000, 0 ), breve.vector( 0.200000, 0.020000, 1 ) )
            breve.createInstances( breve.Step, 1 ).create( breve.vector( 
1.000000, -1.000000, 0 ), breve.vector( 0.200000, 0.020000, 1 ) )
            breve.createInstances( breve.Step, 1 ).create( breve.vector( 
2.000000, -1.200000, 0 ), breve.vector( 2, 0.020000, 1 ) )
            breve.createInstances( breve.Balls, 5 )
            self.pointCamera( breve.vector( 1.000000, -0.800000, -0.600000 ), 
breve.vector( 3.500000, 1.100000, 5.000000 ) )
            self.enableShadowVolumes()
            self.addMenu( '''Reset Ball''', 'resetBall' )

      def resetBall( self ):
            self.theBall.reset()


breve.Gravity = Gravity
class Step( breve.Stationary ):
      def __init__( self ):
            breve.Stationary.__init__( self )

      def create( self, location, sizeVector ):
            stepShape = None

            stepShape = breve.createInstances( breve.Cube, 1 ).initWith( 
sizeVector )
            self.setShape( stepShape )
            self.move( location )


breve.Step = Step
class Ball( breve.Mobile ):
      def __init__( self ):
            breve.Mobile.__init__( self )
            Ball.init( self )

      def init( self ):
            self.setShape( breve.createInstances( breve.Sphere, 1 ).initWith( 
( 0.050000 + breve.randomExpression( 0.150000 ) ) ) )
            self.enablePhysics()
            self.reset()

      def iterate( self ):
            if ( self.getLocation().y < -2.000000 ):
                  self.reset()


      def reset( self ):
            self.setColor( breve.randomExpression( breve.vector( 1, 1, 1 ) ) 
)
            self.move( ( breve.vector( -0.900000, 0.500000, -1 ) + 
breve.randomExpression( breve.vector( 1.500000, 0.800000, 2 ) ) ) )
```

```
            self.setVelocity( breve.vector( ( 1 + breve.randomExpression(
1.000000 ) ), ( 1.000000 + breve.randomExpression( 1.000000 ) ), 0 ) )


breve.Ball = Ball
# Add our newly created classes to the breve namespace

breve.Balls = Ball


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

Figure 9: Python source code for Gravity agent simulation

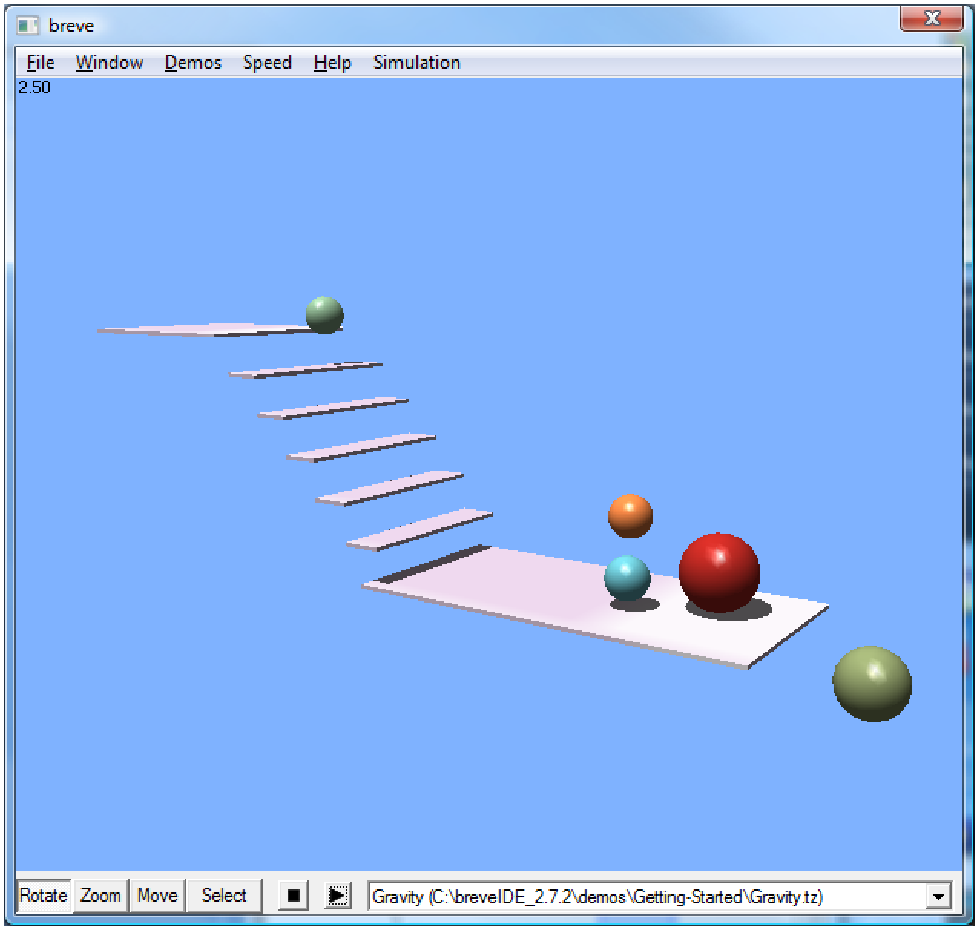
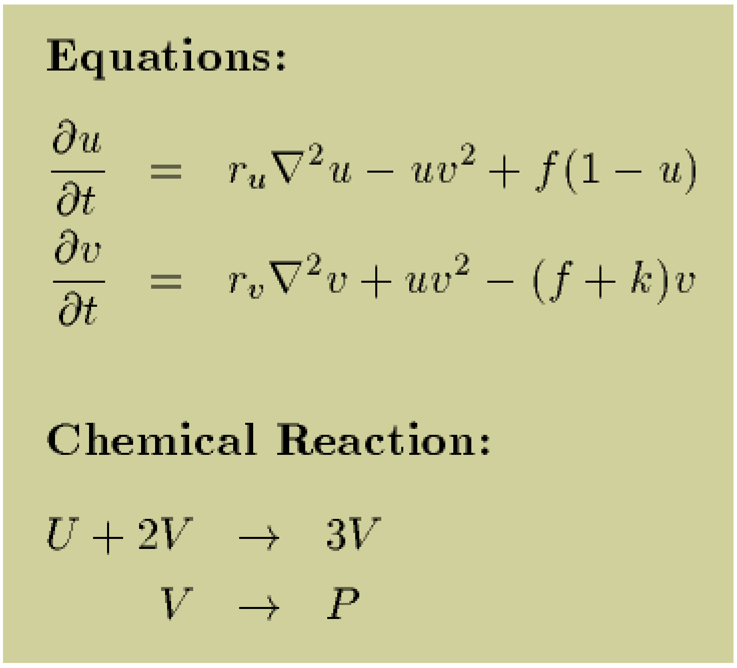

breve
File Window Demos Speed Help Simulation
2.50
Rotate Zoom Move Select
Gravity (C:\breveIDE_2.7.2\demos\Getting-Started\Gravity.tz)

Figure 10: breve gravity and 3D collision simulation. Note, a video of this model has been generated and is available at the following location: http://insecure.ucdavis.edu/Members/adam/physics

The next model shows the interesting Gray Scott model of reaction diffusion [30], which shows that even chemical reactions can generate chaotic patterns in contrast to the conventional intuition that chemical reactions proceed forward to equilibrium.



Equations:

$$\frac{\partial u}{\partial t} = r_u \nabla^2 u - uv^2 + f(1-u)$$

$$\frac{\partial v}{\partial t} = r_v \nabla^2 v + uv^2 - (f+k)v$$

Chemical Reaction:

$$U + 2V \rightarrow 3V$$

$$V \rightarrow P$$

Figure 11: Gray Scott mathematical model

```
# Note: this file was automatically converted to Python from the
# original steve-language source code.  Please see the original
# file for more detailed comments and documentation.


import breve

class GS( breve.Control ):
      def __init__( self ):
            breve.Control.__init__( self )
            self.U = None
            self.V = None
            self.cube = None
            self.delta = None
            self.deltareact = None
            self.inflow = None
            self.texture = None
            GS.init( self )

      def init( self ):
            n = 0
            m = 0

            self.setBackgroundColor( breve.vector( 1, 1, 1 ) )
            self.setIterationStep( 1.000000 )
            self.setIntegrationStep( 1.000000 )
            self.pointCamera( breve.vector( 0, 0, 0 ), breve.vector( 0, 0, 70
) )
            self.U = breve.createInstances( breve.Matrix2D, 1 )
            self.V = breve.createInstances( breve.Matrix2D, 1 )
            self.inflow = breve.createInstances( breve.Matrix2D, 1 )
            self.delta = breve.createInstances( breve.Matrix2D, 1 )
            self.deltareact = breve.createInstances( breve.Matrix2D, 1 )
            self.U.setSize( 128, 128 )
            self.V.setSize( 128, 128 )
            self.delta.setSize( 128, 128 )
            self.deltareact.setSize( 128, 128 )
            self.inflow.setSize( 128, 128 )
            self.texture = breve.createInstances( breve.MatrixImage, 1 )
            self.texture.initWith( self.U, 1.000000 )
            self.texture.setRed( self.V )
            self.texture.setBlue( self.U )
            self.cube = breve.createInstances( breve.Mobile, 1 )
            self.cube.setShape( breve.createInstances( breve.Cube, 1
).initWith( breve.vector( 400, 400, 1 ) ) )
            self.cube.setTextureImage( self.texture )
            self.cube.setTextureScale( 40 )
            n = 0
            while ( n < 128 ):
                  m = 0
                  while ( m < 128 ):
                        self.U.setValue( ( ( 0.500000 +
breve.breveInternalFunctionFinder.sqrt( self, breve.length( ( 0.250000 - ( (
0.010000 * ( 1.000000 + ( 0.040000 / 0.010000 ) ) ) * ( 1.000000 + ( 0.040000
/ 0.010000 ) ) ) ) ) ) ) + ( 0.020000 * ( breve.randomExpression( 1.000000 )
- 0.500000 ) ) ), m, n )
                        self.V.setValue( ( ( ( 1.000000 - self.U.getValue( m,
n ) ) / ( 1.000000 + ( 0.040000 / 0.010000 ) ) ) + ( 0.020000 * (
breve.randomExpression( 1.000000 ) - 0.500000 ) ) ), m, n )
                        self.inflow.setValue( 0.010000, n, m )

                        m = ( m + 1 )

                  n = ( n + 1 )


      def iterate( self ):
            self.deltareact.copy( self.U )
            self.deltareact.multiplyWithValues( self.V )
            self.deltareact.multiplyWithValues( self.V )
            self.delta.computePeriodicDiffusionMatrix( self.U, 0.078000 )
            self.delta.addValues( self.deltareact, -1.000000 )
            self.delta.addValues( self.U, ( -0.010000 ) )
            self.delta.addValues( self.inflow )
            self.U.addValues( self.delta, 1.000000 )
```

```
                self.delta.computePeriodicDiffusionMatrix( self.V, 0.022000 )
                self.delta.addValues( self.deltareact )
                self.delta.addValues( self.V, ( -( 0.040000 + 0.010000 ) ) )
                self.V.addValues( self.delta, 1.000000 )
                breve.Control.iterate( self )


breve.GS = GS


# Create an instance of our controller object to initialize the simulation

GS()
```

Figure 12: Python source code for Gray Scott simulation

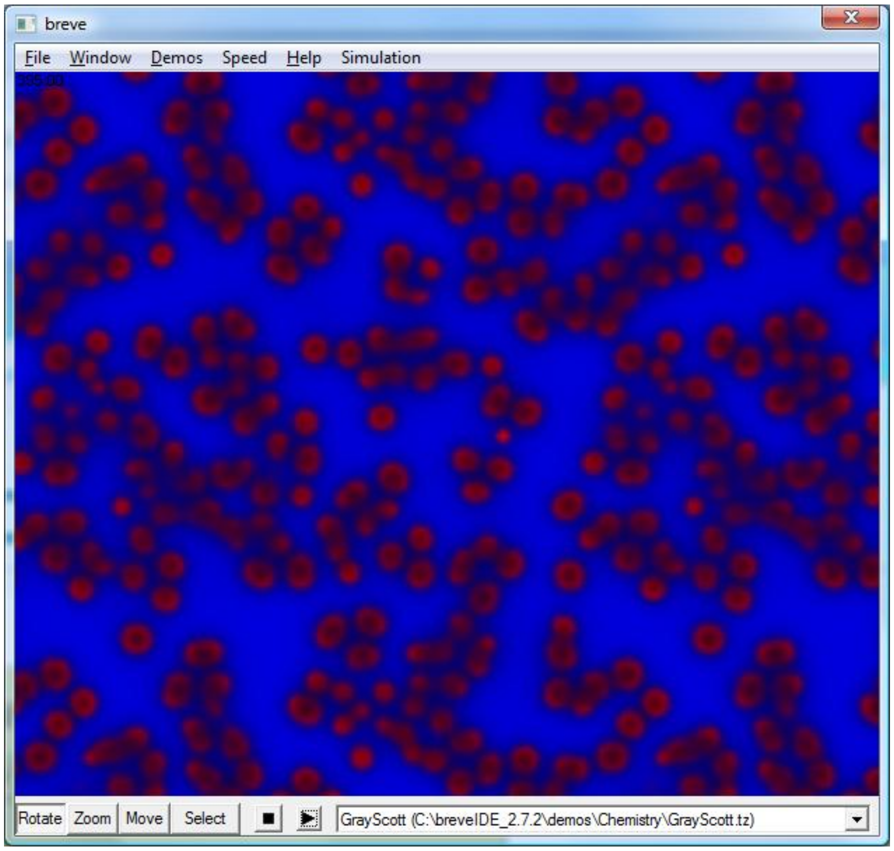

breve
File Window Demos Speed Help Simulation
Rotate Zoom Move Select
GrayScott (C:\breveIDE_2.7.2\demos\Chemistry\GrayScott.tz)

Figure 13: breve Gray Scott simulation. Note, a video of this model has been generated and is available at the following location: http://insecure.ucdavis.edu/Members/adam/physics

The third model examined was a fairly sophisticated agent behavior simulation playing the game Capture the Flag. Breve has a separate section dedicated to the programming and development of sophisticated winning strategies for this game, which involves the following simple rules:

1. Ten agents per side

2. If an agent touches an opposing agent on its home side, the opposing agent is teleported to a jail on the home side and cannot play until freed
3. An agent that touches the opposing flag moves that flag. If the agent moves the flag to its own side, it wins the game for that side
4. An agent that touches the jail on the opposing side frees all agents in that jail

The interesting aspects to the game derive from the limited intelligence of the agents themselves; in particular, as mentioned previously, they cannot see the entire board, and cannot depend on the flag (goal) having a fixed location.

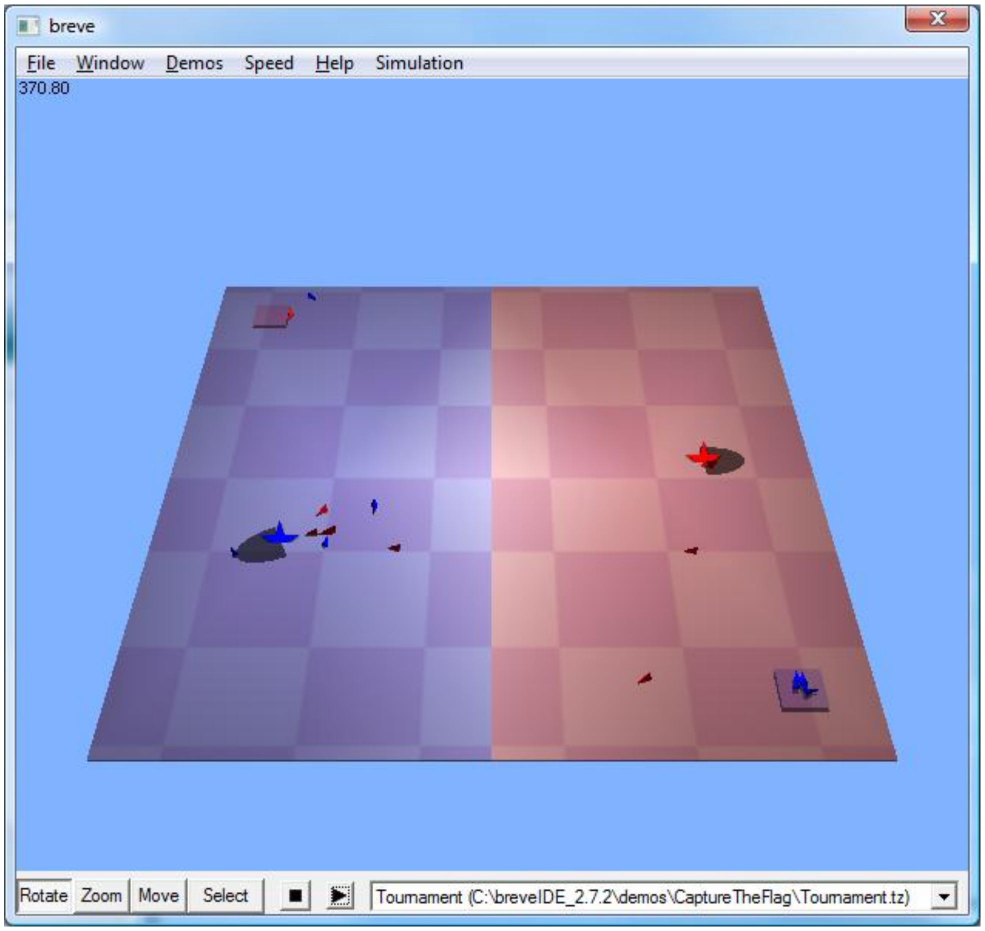

breve
File
Window
Demos
Speed
Help
Simulation
370.80
Rotate
Zoom
Move
Select
Tournament (C:\breveIDE_2.7.2\demos\CaptureTheFlag\Tournament.tz)

Figure 14: breve Capture the Flag

Due to the length of the source code and size of the simulation, results will not be explicitly included in this paper, but the source code is freely available. [31]

Capture the Flag showcases the challenges and limits of traditional agent-based rules for governing behavior. In particular, the agents cannot self-modify their behavior, and react with a fixed strategy every time. So how does one introduce efficient, self-modification in agents?

The techniques of genetic programming provide an answer. In breve, this is implemented using Push, a stack-based genetic programming language. [32]

Non-genetic programming languages are constrained by their rigid syntax. In general, mixing up the order of tokens and operators typically results in garbage, as shown by the following list comprehension in Python which shows randomization of language tokens.

This makes sense:

```
L = [math.exp(val) for val in eigenvalues]
```

This does not:

```
eigenvalues ] in math.exp(val) = L ] for
```

In Push (which are not typically written by hand), programs are composed of instructions, literals, or sublists of the previous two. Push programs are expressions, placed entirely on the stack and evaluated recursively according to these rules:

1. If P is an instruction then execute it
2. Else if P is a literal then push it on to the stack
3. Else (P must be a list) sequentially execute each of the Push programs in P

Let’s look at a simplified Push example program:

(2 3 INTEGER. * 4.1 5.2 FLOAT.+ TRUE FALSE BOOLEAN.OR)

Each type of operation (Push is necessarily strongly-typed) gets placed on its own stack. Evaluating this from left to right we have:

2, 3, INTEGER.* (integer operator multiply) gets placed on an Integer stack.

4.1, 5.2, FLOAT.+ (floating point operator addition) gets placed on a Float stack.

TRUE, FALSE, BOOLEAN.OR (Boolean operator OR) gets placed on a Boolean stack.

Pushing onto the stack from left to right, we then pop the stack right to left :

- First run is: BOOLEAN.OR FALSE TRUE = (TRUE) (BOOLEAN stack)
- Next we have: FLOAT.+ 5.2 4.1 = (9.3) (FLOAT stack)
- Finally we have INTEGER.* 2 3 = (6) (INTEGER stack)

Note that each stack has its own type, and the stack-based typing system puts each instruction appropriately. The result is that any combination remains semantically valid, and wee could re-order all of the instructions (stacks) without issue.

The main trick is to devise programs that actually produce changeable behaviors in the agents, so they can be selected for or against.

In breve, one can do a sophisticated version of the Boids simulation, using evolving swarms and the following additional rules:

- Seek out food, which randomly teleports around
- Feed their friends with excess food
- Reproduce when energy (food) hits certain threshold
- Die when they run out of energy, or reach maximum age
- Land on the ground, rest, fly around again
- Mutate in such a way as to improve/reduce reproduction

The breve model initially starts off with 10 Boids, which reproduce and die according to the given rules above. After 6000 iterations, the simulation stops, although depending upon the state of the model this takes an extended amount of time.

Repeated runs of the model showed widely varying results in Boid populations; some models capped at 10 (the default model auto-generates a new Boid whenever the count drops below 10), or zero when the auto-growth code was removed. Other models generated flocks of 50, 60, 100, 200 or more Boids with high levels of “evolution”.

In general, this is quite a sophisticated simulation that would be incredibly difficult to reproduce via other means. In the author’s opinion, this evolving swarms really demonstrate the utility and value of agent-based modeling and toolkits.

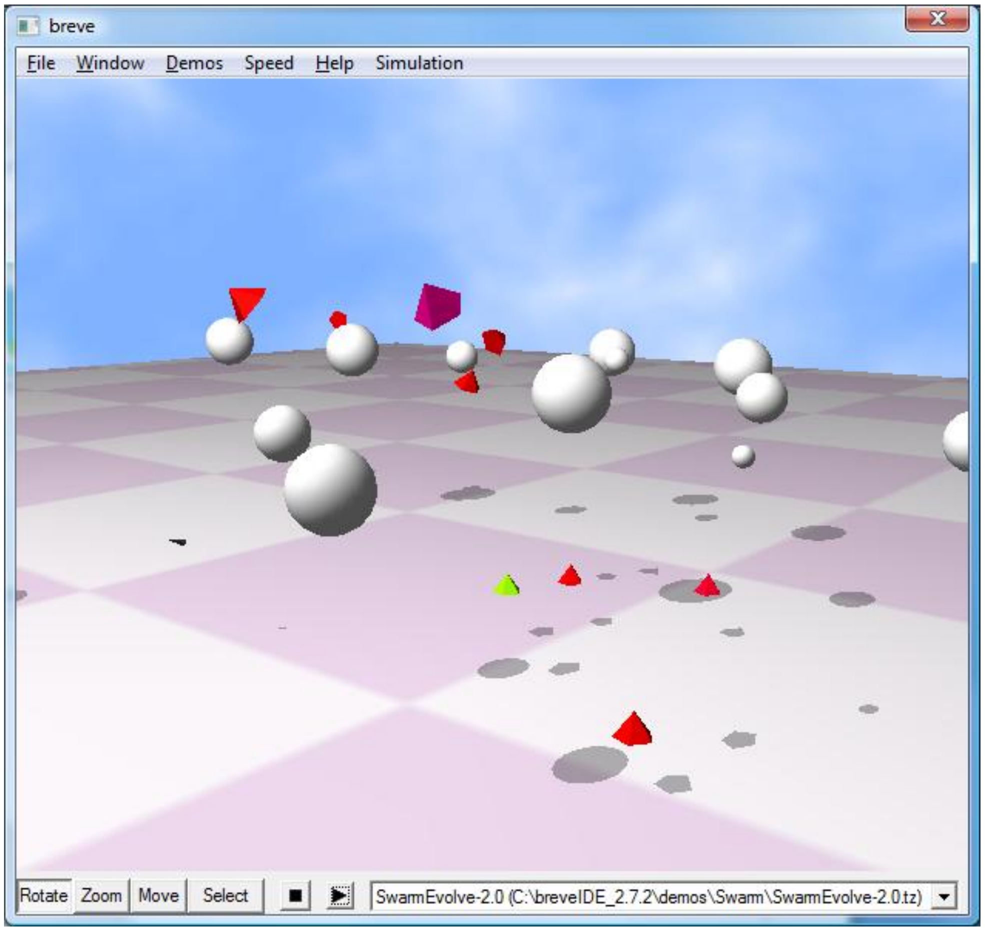

breve
File Window Demos Speed Help Simulation
Rotate Zoom Move Select
SwarmEvolve-2.0 (C:\breveIDE_2.7.2\demos\Swarm\SwarmEvolve-2.0.tz)

Figure 15: Evolving Boid swarms using breve

Again, due to the size of the simulation, source code will not be explicitly detailed in this paper. However, it is available for inspection in breve, and a video of this model has been generated and is available at the following location: http://insecure.ucdavis.edu/Members/adam/physics

SECTION IV – CONCLUSION

This paper has explored the theory and practice of agent-based modeling and tools. The toolkits themselves were found to be, at the time of this writing, of varying degrees of usefulness and power, and their use generated a greater-than-anticipated learning curve during the development of this paper. Agent-based modeling toolkits are subtle and sophisticated, and the freely available ones involve quite a bit of setup, fine-tuning, and tinkering before even a single line of model code is written. The modeling frameworks evaluated all used open-source languages and libraries, usually either Java or Python, though Python was preferred for its simplicity, expressiveness, and ease of use. Particular toolkits had decided advantages depending upon the type of simulation desired. For 3-D modeling, breve was far and away the best tool in terms of setup, ease of use, and sophistication of agent behaviors. For network and grid models with visualization of non spatio-temporal data, Repast Simphony was the best solution evaluated, though not without a steep learning curve and setup issues. Finally, for prototyping of network/grid models (which could be redone in a more complex tool), RepastPy was the clear winner.

Future work entails the development of specific breve and Repast models in the author's area of interest, particularly modeling of 4D general relativity.

Bibliography

[1] Z. Duan, J. Wang, and L. Huang, “Attraction/repulsion functions in a new class of chaotic systems,” *Physics Letters A*, vol. 335, Feb. 2005, pp. 139-149.

[2] “ABM for Systems Biology”; http://www.abmsystemsbiology.info/.

[3] L. Tesfatsion, “Agent-Based Computational Economics”; http://www.econ.iastate.edu/tesfatsi/ace.htm.

[4] P. Davidsson , “Agent Based Social Simulation: A Computer Science View,” Jan. 2002; http://jasss.soc.surrey.ac.uk/5/1/7.html.

[5] A. Crooks, “GIS and Agent-Based Modelling: January 2007,” *GIS and Agent-Based Modelling*, Jan. 2007; http://gisagents.blogspot.com/2007_01_01_archive.html.

[6] C. Castle and A. Crooks, *Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations*, UCL Centre for Advanced Spatial Analysis, 2006; http://www.casa.ucl.ac.uk/working_papers/paper110.pdf.

[7] “Agent-based model,” *Wikipedia, the free encyclopedia*; http://en.wikipedia.org/wiki/Agent_based_modeling.

[8] S.H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology*, Westview Press, 2001.

[9] Joshua M. Epstein, “Agent-based computational models and generative social science,” *Complexity*, vol. 4, 1999, pp. 41-60.

[10] “Multi-agent system,” *Wikipedia, the free encyclopedia*; http://en.wikipedia.org/wiki/Multi-agent_system.

[11] "Multiagent Systems | Where game theory, artificial intelligence, distributed programming, and the semantic web meet."; http://www.multiagent.com/.

[12] "Distributed artificial intelligence," *Wikipedia, the free encyclopedia*; http://en.wikipedia.org/wiki/Distributed_artificial_intelligence.

[13] "Distributed Artificial Intelligence"; http://www.info.fundp.ac.be/~pys/ModelAge/prop/subsection2_7_0_4.html.

[14] "Swarm intelligence - Scholarpedia"; http://www.scholarpedia.org/article/Swarm_intelligence.

[15] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University PressUS, 1999.

[16] "Swarm Intelligence - Artificial Intelligence (incl. Robotics) Journals, Books & Online Media | Springer"; http://www.springer.com/computer/artificial/journal/11721.

[17] S.F. Railsback, S.L. Lytinen, and S.K. Jackson, "Agent-based Simulation Platforms: Review and Development Recommendations," Sep. 2005; http://www.humboldt.edu/~ecomodel/documents/ABMPlatformReview.pdf.

[18] M.J. de Smith, M.F. Goodchild, and P.A. Longley, "Geospatial Analysis - a comprehensive guide. 2nd edition © 2006-2008 de Smith, Goodchild, Longley," Feb. 2008; http://www.spatialanalysisonline.com/output/.

[19] "Main Page - SwarmWiki"; http://www.swarm.org/index.php?title=Main_Page.

[20] *Swarm main page - SwarmWiki*; http://www.swarm.org/index.php?title=Swarm_main_page.

[21] S. Luke, G.C. Balan, and L. Panait, *MASON Multiagent Simulation Toolkit*; http://www.cs.gmu.edu/~eclab/projects/mason/.

[22] U. Wilensky, “NetLogo Home Page,” *Center for Connected Learning and Computer-Based Modeling*; http://ccl.northwestern.edu/netlogo/.

[23] *Repast Agent Simulation Toolkit*; http://repast.sourceforge.net/.

[24] M. Parker, *metaABM*; http://metaabm.org/docs/index.html.

[25] J. Klein, *breve: a 3d Simulation Environment for Multi-Agent Simulations and Artificial Life*, Hampshire College, ; http://www.spiderland.org/.

[26] “Groovy - Home”; http://groovy.codehaus.org/.

[27] *SourceForge.net: VisAD Java Visualization*; http://sourceforge.net/projects/visad/.

[28] “SourceForge.net: Weka---Machine Learning Software in Java”; http://sourceforge.net/projects/weka/.

[29] C. Reynolds, “Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model),” *Boids, Backround and Update*; http://www.red3d.com/cwr/boids/.

[30] Abelson et al., “Gray Scott Home Page,” *Gray Scott Model of Reaction Diffusion*; http://www.swiss.ai.mit.edu/projects/amorphous/GrayScott/.

[31] J. Klein, “Capture the Flag in breve | breve,” *Capture the Flag in breve*; http://www.spiderland.org/CaptureTheFlag.

[32] J. Klein, “"Push": a Language for Evolutionary Computation Integrated With breve | breve”; http://www.spiderland.org/node/2759.