Agent-based Modeling

Adam Getchell

Nonlinear Physics: Modeling Chaos and Complexity
What is an Agent?

Historically related to the Von Neumann machine, as later improved by Stanislaw Ulam into the first agent-based device – the cellular automata.

Agents have:
- Activity
- Autonomy
- Heterogeneity
Agent Activity

- Goal-direction
- Reactivity/Perceptivity to its surroundings (model)
- Mobility: Able to roam the model space independently
- Bounded Rationality (imperfect information)
- Interacts/exchanges information with other agents, which may in turn cause:
- Adaptation: Change in behavior based on interactions with the model or other agents
What is a Model?

import breve

class myControl( breve.Control ) :
    def __init__( self ) :
        breve.Control.__init__( self )
        self.walkerShape = None
        myControl.init( self )

    def getWalkerShape( self ) :
        return self.walkerShape

    def init( self ) :
        print "Setting up the simulation."
        self.pointCamera( breve.vector( 0, 0, 0 ), breve.vector( 0, 60, 0 ) )
        self.walkerShape = breve.createInstances( breve.Sphere, 1 ).initWith( 1 )
        breve.createInstances( breve.RandomWalker, 200 )

breve.myControl = myControl
class RandomWalker( breve.Mobile ) :
    def __init__( self ) :
        breve.Mobile.__init__( self )
        RandomWalker.init( self )

    def init( self ) :
        self.setShape( self.controller.getWalkerShape() )
        self.setColor( breve.randomExpression( breve.vector( 1.000000, 1.000000, 1.000000 ) ) )
        self.move( breve.randomExpression( breve.vector( 0.100000, 0.100000, 0.100000 ) ) )

    def iterate( self ) :
        self.setVelocity( ( breve.randomExpression( breve.vector( 60, 60, 60 ) ) - breve.vector( 30, 30, 30 ) ) )

breve.RandomWalker = RandomWalker

# Create an instance of our controller object to initialize the simulation
myControl()
# Tools and Languages in 2008

<table>
<thead>
<tr>
<th>Program</th>
<th>Language(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swarm</td>
<td>Objective-C, Java</td>
<td>Agent modeling library, dated, last release version 2.2 February 2005</td>
</tr>
<tr>
<td>RepastJ</td>
<td>Java</td>
<td>Based on Swarm, written specifically in Java</td>
</tr>
<tr>
<td>RepastPy</td>
<td>Python</td>
<td>Friendly GUI, uses Python for scripting, limited</td>
</tr>
<tr>
<td>Repast.NET</td>
<td>C#, VB.NET</td>
<td>Leverages .NET framework, doesn't work with Visual Studio 2008</td>
</tr>
<tr>
<td>Repast Simphony</td>
<td>Java</td>
<td>Full-featured, uses Eclipse IDE, can be difficult to setup</td>
</tr>
<tr>
<td>MetaABM</td>
<td>Java</td>
<td>Full-featured, uses Eclipse IDE plus own GUI, designed to use standard model file that can work with other tools (Repast, Weka, VisAD, MatLAB), can be difficult to setup</td>
</tr>
<tr>
<td>Breve</td>
<td>Steve, Python, Push</td>
<td>Fast, easy to use 3d simulation environment targeted towards physics and artificial life</td>
</tr>
</tbody>
</table>
RepastPy -- Model

- Simple GUI which generates Java classes
RepastPy – Agent

Simple Python scripting for behaviors
Repast Simphony

- Uses Java + Groovy to compile an application
Boids

- A kind of 3D Life simulation producing chaotic behavior. The rules are:
  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 near boids (perceived velocity of neighbors)
A Simphony of Boids
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()
breve – basic Controller/Agent structure (steve)

@include "Control.tz"

Controller HelloWorld.

Control : HelloWorld {
  + to iterate:
    print "Hello, world!".
    super iterate.
}
breve – Gravity & 3D collisions
breve – Gray Scott model of reaction diffusion

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:

\[
U + 2V \rightarrow 3V
\]

\[
V \rightarrow P
\]
brevé – Capture the Flag
breve – boids to evolving swarms
Evolving swarms

In addition to the behaviors of boids, swarm agents:
- 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

So how to you mutate code that must be pre-defined?
Push

Genetic programming – random crossover and mutation of computer programs

- Doesn’t work for most computer languages, since they typically have rigid syntax:
- This makes sense:
  \[ L = [\text{math.exp}(\text{val}) \text{ for } \text{val} \text{ in } \text{eigenvalues}] \]
- This does not:
  \[ \text{eigenvalues } \text{ in } \text{math.exp(val)} = L \text{ for} \]
Push

- Programs made up of: instructions, literals, and sublists
- Push program is an expression, entirely placed 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
Sample Push program and execution

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

- 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, the stack-based typing system puts each instruction on its own type of stack, so that any combination remains semantically valid. We could re-order all of these 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
References

- Jon Klein, breve: a 3d Simulation Environment for Multi-Agent Simulations and Artificial Life (Hampshire College), http://www.spiderland.org/.