Patrick Coleman

Period 6

COMPUTER SYSTEMS RESEARCH

Fall/Spring 2007-2008

Research Paper Second Draft - January 2008

The Sugarscape

1. Abstract

This project studies artificial societies, especially the Sugarscape and the Schelling segregation model. To implement the Sugarscape, a display of the sugar-filled environment with agents is outputted, the simulation allows agents to harvest sugar, consume sugar, die of starvation, migrate during plagues, reproduce, and combat each other and allows the environment to grow back at a given rate and undergo plagues. To implement the Schelling segregation model, two or three distinct groups of agents are added to the environment with a preference for neighbors of their own kind to determine the effects of the individual preferences on the society at large. The reasons these two projects are being implemented is because while both are often compared, the two models in their original forms have not been combined and analyzed in a single simulation. In addition to displaying the environment graphs showing the population growth and wealth distribution are displayed. These graphs analyze what is occurring in the simulation. The program code is broken up into files: a main file, an environment file, an agent file, a location file, a display file, and a simulation file. The conclusions show that the the model conforms to Axtell and Epstein's models in the areas which were implemented. But more importantly, it shows that the simulation conforms to real world phenomena reasonably well.

2. Introduction

The program implements several aspects of Axtell and Epstein's Sugarscape model. An environment with locations holding various amounts of sugar, which grows back over time is populated by a heterogeneous group of agents, different with respect to vision, metabolism (rate at which sugar is consumed), starting wealth, and age limit. The agents move to the locations with the highest concentration of sugar. Over time agents die and new agents are added to the environment according to a logistic function. Two graphs are displayed: one showing population growth over time and the other showing wealth distribution. The information shown by these two graphs will be analyzed.

3. Background

As of yet the Sugarscape society has not been implemented in Ruby and it would be valuable for this code to be available because of the scope of the Sugarscape research. Sugarscape has inspired further research concerning agent-based modeling and artificial societies. The Schelling segregation model was one of the first artificial societies to be implemented on a computer and has defined the area of study. The combination of these two models can provide valuable insight into human culture. Perhaps 3 different groups could be put into the Sugarscape instead of the usual two different groups. Lastly, combat between different groups will be implemented, as this has not yet been done by Tony Bigbee at George Mason.

Growing Artificial Societies: Social Sciences from the Bottom Up written by Joshua M. Epstein and Robert Axtell and *Micromotives and Macrobehavior* by

Thomas Schelling define Sugarscape and the Schelling segregation model. Tony Bigbee from George Mason University has written the Sugarscape in Java and his code will be used for reference along with the first book primarily. In the book by Axtell and Epstein Schelling's segregation model is mentioned and the Sugarscape is built with two separate groups (tribes) which combat against each other. The results should mirror those of the Sugarscape models in *Growing Artificial Societies*. However, once Schelling segregation is implemented with possibly more than two different colored populations the results will differ. In all likelihood only two groups will survive in the long run. The final results will be presented with screenshots of the running program along with graphs of relationships of variables. It will perform like previous Sugarscape models. *Growing Artificial Societies* and *Micromotives and Macrobehavior* are two books which are used as references to develop this project. The article "Seeing Around Corners" shows how both the Schelling segregation model and the Sugarscape compare to various other artificial societies in the field of generative social sciences. "The Theoretical Basics of Popular Inequality Measures" examines various ways of determining inequality in a population. The measures used are: simple range, the McLoone index, the coefficient of variation, the Gini coefficient, and Theil's T statistic. Gigliotta's article "Groups of Agents with a Leader" examines how a leader affects a group of agents attempting to reach a goal location. The leader was effective in small groups without communication, and especially effective when he had increased vision.

4. Development

I. Theory. The algorithm driving the move method of the agents is at the core of the simulation. The agents look out in the four cardinal directions as far as their vision allows and move one square in the direction of the closest location with the most sugar. If more than one location is optimum, a random direction is chosen. See the section on agent movement in Appendix A. Agents are added to the environment according to a logistic function which follows the form of: $P * (1 - \frac{P}{K})$ where P is the current population level and K is the carrying capacity. The section on population growth in Appendix A shows how the population growth is graphed. The inequality of the population is found using the Gini coefficient. The Gini coefficient is calculated according to the formula: $1 - 2 * L$ where L is the area under the Lorenz curve, which is calculated using trapezoidal Reiman sums. The section on wealth distribution in Appendix A shows how the Gini coefficient is calculated and how the Lorenz curve is graphed.

II. Design Criteria. The goal of the project is to accurately represent the models it is implementing. It follows the Sugarscape design from *Growing Artificial Societies* by Axtell and Epstein and the Schelling segregation design from Schelling's book *Micromotives and Macrobehavior*. The agents and the environment behave as they should with respect to the aspect implemented so far. The Schelling segregation model will accurately represent Schelling's model as best as possible, but will not be perfect because concessions will need to be made

to allow it to run in the Sugarscape. The information shown in the graphs and the display of the environment will be compared to the results found by the authors.

III. Materials. The program code was written in Ruby (see http://ruby-lang.org/). Tk toolkit is used for the GUI representation and graphics in the program. A text file which represents the maximum capacities of sugar in various locations in the environment was used from GMU's Tony Bigbee's files (he wrote a Java version).

IV. Procedures. Currently the program displays the environment, and has the agents move and harvest sugar. The display draws each location in the matrix using a circle whose radius increases based on the amount of sugar at that location. The display draws the agents as a red circle with the same radius as a location with the maximum amount of sugar. The display also shows the current time step. The GUI window has a frame containing the canvas and buttons to play, pause, and step the simulation and to quit the program. The agents themselves choose the closest location with the greatest amount of sugar. If more than one location matches these requirements, one of them is randomly chosen. Then the agent harvests the sugar and consumes from his own supply of sugar. At each time step the sugar in the environment grows back by one. The program begins with a small number of agents and adds to the population using a logarithmic function so that it reaches carrying capacity. Modifications in the individual agents include an improved move method, a random age limit, and a variable for red or blue color. The GUI window has been modified to include buttons to change the graph and change the refresh rate. There is an input box to change the refresh rate in the display of the environment and of the graphs. the two graphs which are now displayed are the population growth over time, and the percent of total wealth over the percent of the population (Lorenz curve). To get the population graph, it keeps track of the length of the array of agents at each time step in the simulation file and cycles through the array of population values in the display file. To get the wealth graph, it cycles through the array of agents and stores the wealth of each individual agent. Then it sorts this array and cycles through it keeping a running total to determine percents.

5. Results

It has been determined that the program meets the design criteria in the areas in which it was implemented. The graphs are what answer many of the experimental questions. Descriptions of the population growth graph refer to the section on population growth in Appendix B. In general it follows the shape of logistic graphs which are proven to be a fairly accurate representation of population growth. Growth is slow when the population is close to zero and close to the carrying capacity, and growth is highest at half of the carrying capacity. The few anomalies reveal certain aspects of the simulation. The initial portion of slow growth is smaller than the final portion because the population begins with three individuals instead of one (but starting with one agent would not

completely remedy this). The oscillations near carrying capacity come from the age limit of agents. It takes longer for the population to decrease due to dead agents than it does for it to react to the added agents. The oscillations decrease over time and will eventually disappear. At about half of carrying capacity the line begins to become jagged instead of fairly straight like it was earlier in the simulation. This is a result of the heterogeneous population. In the beginning even agents with low vision and high metabolism (less fit agents) have room to survive in the regions of abundant sugar. As the environment fills up only better fit agents can survive on the fringes, areas with less sugar, so many added agents die quickly. The effects are even more pronounced as population approaches carrying capacity. Descriptions of population inequality refer to the graph in the wealth distribution section of Appendix B. At first a bar graph was used to represent wealth distribution, but it was replaced with the Lorenz curve. Both conform to the graphs in Axtell and Epstein's book. They show that there are very few wealthy agents (agents with a lot of harvested sugar stored) and many poor agents. In this sense the population is pretty unequal. The Gini coefficient is a numerical representation of this phenomenon. A coefficient of zero represents perfect equality and one represents perfect inequality (one agent has all the wealth). The number is just over .5 showing that the Sugarscape population is closer to perfect inequality than to perfect equality.

6. Further Research

Further research could include implementing other aspects of the Sugarscape, as described by Axtell and Epstein. Possible topics include reproduction or the trade of spice. In addition, other studies of artificial societies (like Schelling's segregation model) could be analyzed using the Sugarscape as the base environment. Changing the range of values in the heterogeneous aspects of the agents yields different results in the graphs. This could be attempted to be quantified. Combat could be implemented, a desire expressed by Tony Bigbee. Genocide, as described in "Seeing Around Corners" could be implemented. The manner in which agents are added to the environment could be done according to other functions to show other phenomena, like exponential growth.In addition to changing the environment, the method of determining social equality could be determined using some of the different methods described in "The Theoretical Basics of Popular Inequality Measures." Lastly, the Sugarscape could be implemented in other languages, like assembly for example.

Bibliography

Epstein, Joshua M. and Robert Axtell. *Growing Artificial Societies: Social Sciences from the Bottom Up.* Washington, D.C.: The Brookings Institute Press, 1996.

Gigliotta, Onofrio. "Groups of Agents with a Leader." Journal of Artificial Societies and Social Simulation 10 (2007). 30 Jan. 2007

$< http://jasss.soc.surrey.ac.uk/10/4/1.html >$.

Hale, Travis. "The Theoretical Basics of Popular Inequality Measures." University of Texas. $< http://utip.gov.utexas.edu/tutorials/theo_basic_ineq_measures.doc >$

Rauch, Jonathan. "Seeing Around Corners." The Atlantic Monthly. Apr. 2002. 35-48.

Schelling, Thomas C. *Micromotives and Macrobehavior.* New York: W. W. Norton & Company, Inc., 1978.

Appendices

Appendice A

Code

agent move method

```
def move2
choices = []
4.times {choices << [@@env[@posY][@posX],-1,0]}
choices[0] = nil if @posY+1 >= @@env.length or
@@env[@posY+1][@posX].hasAgent != -1
choices[1] = nil if @posY-1 < 0 or @@env[@posY-1][@posX].hasAgent != -1
choices[2] = nil if @posX+1 >= @@env.length or
@@env[@posY][@posX+1].hasAgent != -1
choices[3] = nil if @posX-1 < 0 or @@env[@posY][@posX-1].hasAgent != -1

return if choices == [nil,nil,nil,nil]

for k in 1..@vision do
if choices[0] != nil and @posY+k < @@env.length and
@@env[@posY+k][@posX].sugarquant > choices[0][1]
choices[0] = [@@env[@posY+1][@posX],@@env[@posY+k][@posX].sugarquant,k]
end
if choices[1] != nil and @posY-k >= 0 and
@@env[@posY-k][@posX].sugarquant > choices[1][1]
choices[1] = [@@env[@posY-1][@posX],@@env[@posY-k][@posX].sugarquant,k]
end
if choices[2] != nil and @posX+k < @@env.length and
@@env[@posY][@posX+k].sugarquant > choices[2][1]
choices[2] = [@@env[@posY][@posX+1],@@env[@posY][@posX+k].sugarquant,k]
end
if choices[3] != nil and @posX-k >= 0 and
@@env[@posY][@posX-k].sugarquant > choices[3][1]
choices[3] = [@@env[@posY][@posX-1],@@env[@posY][@posX-k].sugarquant,k]
end
end

choices = choices.compact
choices.sort! {|x,y| y[1] <=> x[1]}

while choices[-1][1] != choices[0][1] and choices[-1][2] != choices[0][2]
choices.pop
end

i = rand(choices.length)
```

```
return if choices[i][0] == nil
@posX = choices[i][0].posX
@posY = choices[i][0].posY
end
```

population graphing method

```
def drawPop
$graph.delete(:all)
TkcLine.new($graph,40,40,40,$w-40)
TkcLine.new($graph,40,$w-40,$w-40,$w-40)
TkcText.new($graph,$w/2,30,:text=>"Population levels over time",
:font=>['Helvetica',15,'bold'])
TkcText.new($graph,10,$w/2,:text=>"P\no\np\n \nl\ne\nv\ne\nl",
:font=>['Helvetica',10,'bold'])
TkcText.new($graph,$w/2,$w-10,:text=>"Time",:font=>['Helvetica',10,'bold'])
for n in 1...popLength do
break if getPop[n] == nil
TkcLine.new($graph,50+((n-1)*400/$popLength),
$w-40-($w-100)*getPop[n-1]/maxPop,50+n*400/$popLength,$w-40-($w-100)*getPop[n]/maxPop)
end
TkcOval.new($graph,50+n*400/$popLength-2,$w-40-
($w-100)*getPop[-1]/maxPop-2,50+n*400/$popLength+2,$w-40-($w-100)*getPop[-1]/maxPop+2)
tinit = getStep-getPop.length
TkcText.new($graph,40,$w-30,:text=>"#{tinit}")
TkcText.new($graph,50+n*400/$popLength,$w-30,:text=>"#{getStep}")
TkcText.new($graph,25,60,:text=>"#{maxPop}")
TkcText.new($graph,75+n*400/$popLength-2,
$w-40-($w-100)*getPop[-1]/maxPop,:text=>"#{getPop[-1]}")
TkcText.new($graph,25,$w-40,:text=>"0")
end
```

Lorenz curve graphing method

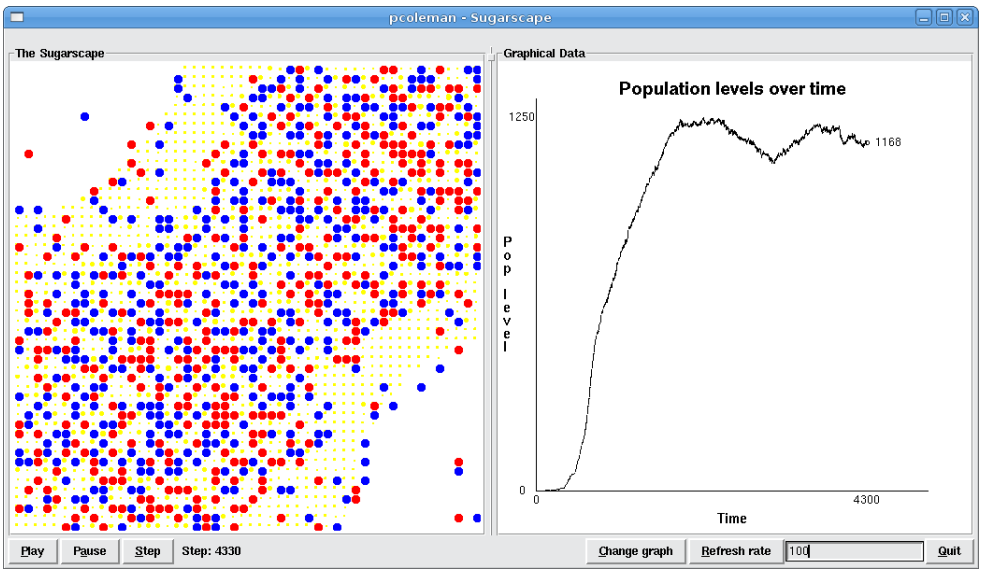
```
def drawWealth
$graph.delete(:all)
TkcLine.new($graph,40,40,40,$w-40,:arrow=>:first)
TkcLine.new($graph,40,$w-40,$w-40,$w-40,:arrow=>:last)
TkcText.new($graph,$w/2,30,:text=>"Wealth Distribution",
:font=>['Helvetica',15,'bold'])
TkcText.new($graph,10,$w/2,:text=>"%\n \no\nf\n \nW\ne\na\nl\nt\nh",
:font=>['Helvetica',10,'bold'])
TkcText.new($graph,$w/2,$w-10,:text=>"% of Population",
:font=>['Helvetica',10,'bold'])
```

```
wealths = [0]
totalW = 0

for a in $env.agents do
w = a.wealth
totalW += w if w > 0
wealths << w if w > 0
end
wealths.sort!

x = 400.0/(wealths.length-1)
y = 400.0/totalW
dx = 1.0/(wealths.length-1)
dy = 1.0/totalW

giniAr = 0

count = [0,0]
for n in 1...wealths.length do
count = [count[1],count[1]+wealths[n]]
TkcLine.new($graph,40+(n-1)*x,$w-40-count[0]*y,40+n*x,$w-40-count[1]*y)
giniAr += (count[0] + 0.5 * (count[1]-count[0]))*dx*dy
end

TkcText.new($graph,30,$w-30,:text => "0%")
TkcText.new($graph,440,$w-25,:text => "100% (#{$env.agents.length})")
TkcText.new($graph,40,15,:text => "100%")
TkcText.new($graph,40,30,:text => "(#{totalW})")
TkcText.new($graph,150,150,:text=>"Gini coefficient:\n%f" % (1.0-2*giniAr),
:font=>['Helvetica',10,'bold'])
end
```

Appendice B

Graphs

population growth



wealth distribution