Patrick Coleman

Period 6

COMPUTER SYSTEMS RESEARCH

Fall/Spring 2007-2008

Research Paper First Draft - November 2007

The Sugarscape

1. Abstract

This project will study artificial societies, especially the Sugarscape and the Schelling segregation model. To implement the Sugarscape, a display of the sugar-filled environment with agents will be outputted, the simulation will allow agents to harvest sugar, consume sugar, die of starvation, migrate during plagues, reproduce, and combat each other and will allow 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 will be 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. The program code will be broken up into files: a main file, an environment file, an agent file, a location file, a display file, and a simulation file. Study questions would be: Does the prgram accurately represent Epstein's and Axtell's model? Do the results of Schelling segregation in the Sugarscape match what has been done by Schelling in an environment without sugar? Does the hetergogeous population of agents mimic the basic patterns of a society? When concluded the product will accurately implement the models as described by Schelling and Axtell and Epstein.

2. Introduction and 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 Schellingu2019s 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 I am using as references to develop this project. The article "Seeing Around Corners" has let me see how both the Schelling segregation model and the Sugarscape compare to various other artificial societies in the field of generative social sciences.

Study questions would be: Does the prgram accurately represent Epstein's and Axtell's model? Do the results of Schelling segregation in the Sugarscape match what has been done by Schelling in an environment without sugar? Does the hetergogeous population of agents mimic the basic patterns of a society?

3. Developments:

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. See Appendice A. 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. Developing my project according to the published rules, I will be able to answer my first question (Does the prgram accurately represent Epstein's and Axtell's model?) in the affirmative. See Appendice B. So far the question about Schelling's segregation model and the accuracy of this model compared wiht actual society cannot be answered because the program is in its beginning stages.

4. Results:

The program will be checked to see if it corresponds to the results obtained by Axtell and Epstein. Mathematical formulas are listed in the back of the book, and displays of charts and graphs showing the relationships between various variables are shown throughout the book. These can be used in conjunction with the version implemented in Ruby which will eventually display graphs that can be compared to said graphs and mathematical formulas. The program will meet as many of the specifications of Sugarscape as defined by Axtell and Epstein as possible. As there is a large amount of data on the results of certain variations of the Sugarscape in the book by Axtell and Epstein, versions of the project can be compared to the results in the book to see if it is running as it should. The testing I have done so far has been verifying whether or not my running program matches the one described by Axtell and Epstein in Growing Artificial Societies. I have done various tests displaying to the command line different information such as the possible choices of an agent or the amount of sugar at each location. This information is used to track down the problem in the code so that the program will run correctly. See Appendice C.

5. Discussion

The program worked as was expected for the first quarter. It currently allows for basic agent movement, consumption, and dying which was my goal. The agents move towards the areas of higher sugar concentration because it is more beneficial for them provided that they have enough vision to see the increase in sugar. See Appendice C. Those with poor vision and high metabolism will die the fastest because they will go through their initial wealth quickly and will not see the higher concentrations of sugar if they start on the outside. This shows that in a heterogenous society, the agents better equipped from the start due to their innate characteristics will be more successful. I have created a successful first implementation of the Sugarscape in Ruby but I have not yet implemented the Schelling segregation.

Bibliography

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

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

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

Appendices

Appendice A

Act method of agent:

#Gets possible locations, chooses one, harvests sugar at that location, and consumes sugar

def act

k = @vision

sug = 0

choices = []

#Check spaces vision number spaces away from the agent in the 4 cardinal directions

@vision.times do

arr = []

#Checks to see if possible location is in the environment and is unoccupied

arr << @@env[@posY+k][@posX] if @posY+k < @@env.length and @@env[@posY+k][@posX].hasAgent == false

```
arr << @@env[@posY-k][@posX] if @posY-k >= 0 and @@env[@posY-k][@posX].hasAgent == false

arr << @@env[@posY][@posX+k] if @posX+k < @@env.length and @@env[@posY][@posX+k].hasAgent == false

arr << @@env[@posY][@posX-k] if @posX-k >= 0 and @@env[@posY][@posX-k].hasAgent == false

#Does not continue if no locations are valid

if arr != nil

 frst = 0

 #Cycles through possible locations

 arr.each do |a|

  #If the quantity of sugar in one location is greater than previous greatest location the choices are reset

  if a.sugarquant > sug

   sug = a.sugarquant

   choices.clear

   choices << a

  #If the quantity of sugar in one location is equal to the previous greatest location the location is saved

  elsif a.sugarquant == sug

   #Resets the choices if the equal amount is closer to the agent

   choices.clear if frst == 0

   choices << a

  end

  frst = 1

 end

end

k -= 1

end

#Chooses a random choice and moves agent to that location

if choices != nil

i = rand(choices.length)
```

@posX = choices[i].posX

@posY = choices[i].posY

end

#Tells the location it has an agent

@@env[@posY][@posX].hasAgent= true

#Adds harvested sugar to wealth

@wealth += @@env[@posY][@posX].harvest

#Consumes sugar from the amount of sugar possessed

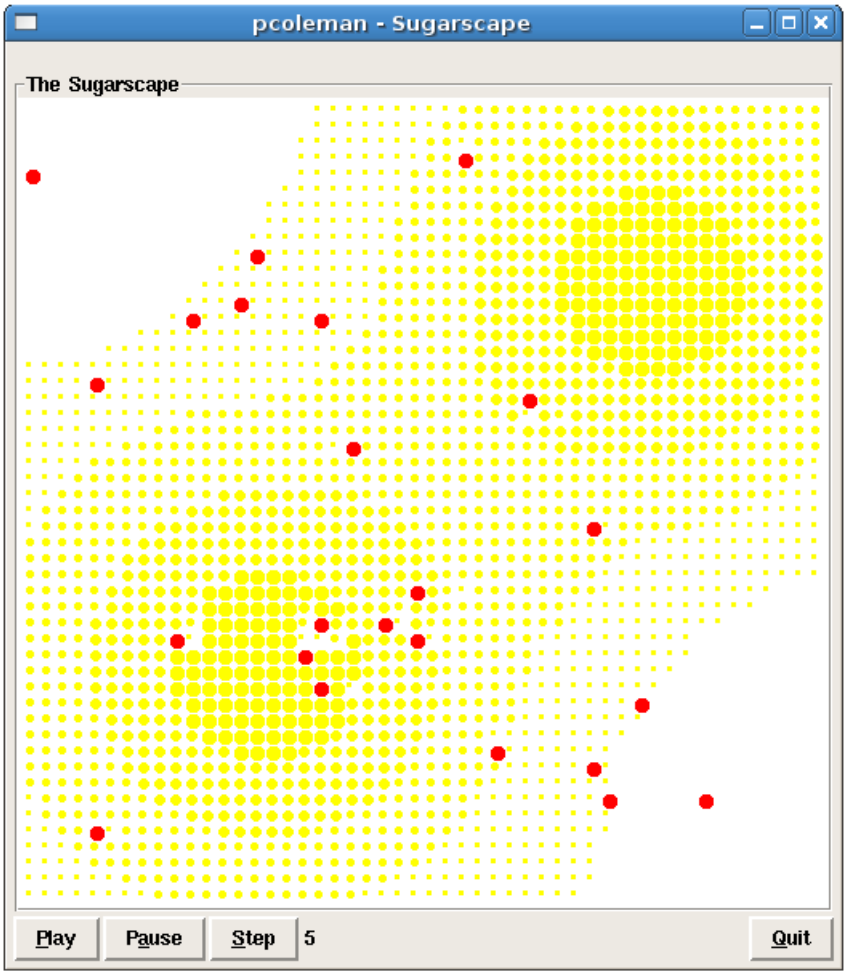
@wealth -= @metabolism

#Agent dies if it does not have enough sugar to consume

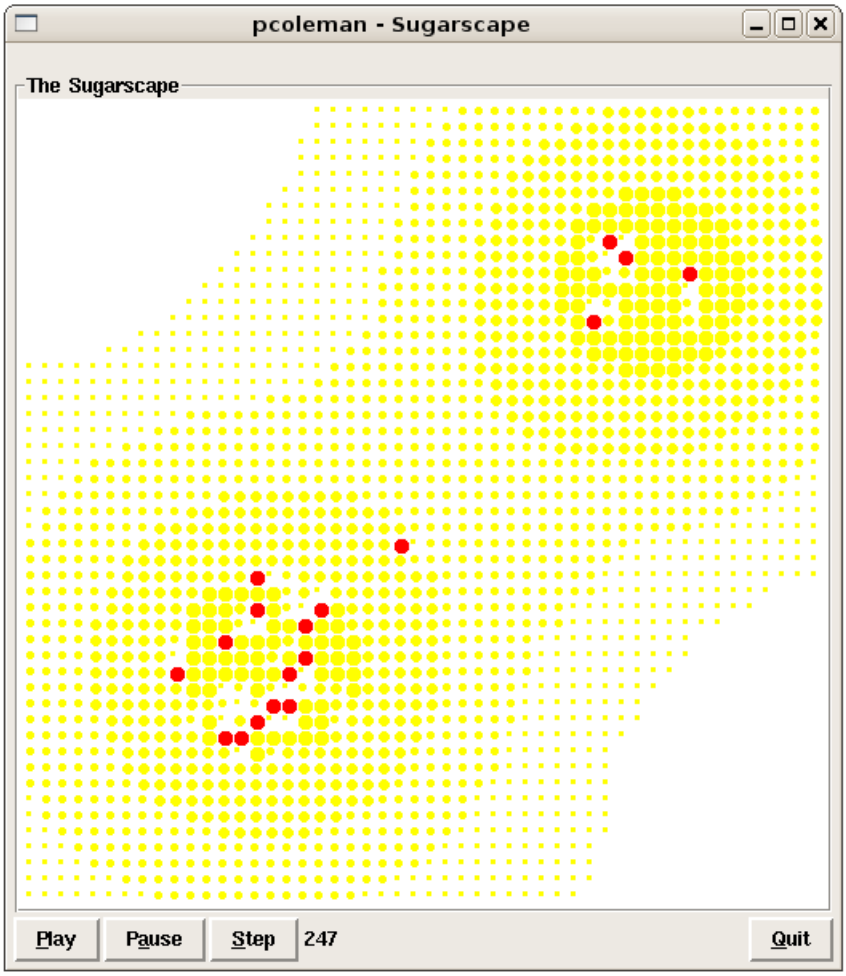
@dead = true if @wealth < 0

end

Appendice B



Appendice C


pcoleman - Sugarscape
The Sugarscape
Play
Pause
Step
247
Quit