Project 4

Added by Christopher Hoder, last edited by Christopher Hoder on Oct 16, 2011

The goal of Project 4 was to create yet another simulation and in this case we simulating a landscape that has resource which an agent (broccoliFiend) needs to eat inorder to survive. These BroccoliFiends are nomadic harvesters that are seeking out the best Broccoli plants to eat. The Broccoli plants live on the landscape, do not move or interact with each other but grow with every day (rate of growth depends on location and season, to be discussed later). In this, more complicated simulation the gird is now a continuous, bounded region on which both the Broccoli and BroccoliFiends can live anywhere and can occupy the same space.

In my solution to this project, the Broccoli and BroccoliFiends are not stored on the grid themselves, as they were in the past. Instead these objects were stored in their own respective singly linked list. I created a class that stores SimObjects in a linked list that could handle these two objects, as well as the SocialAgents and Cells as part of the extension. By removing the grid from the simulation, we simplified the code to storing the data because we were able to remove the duplication of storing each agents position in both the object and on the grid. However, searching for neighbors and surrounding objects becomes a bit more computation heavy because we are forced to sort down the entire linked list, taking on the O(n) scale versus finding neighbors in past projects which only required that you check 9 spots (O(1)). When it came to implementing the SocialSimulation and the Game of Life this reduction in speed became very apparent. The Game of Life ran very slow, especially with large grids. However, the removal of the grid allows us to make our simulation much more complicated with not very much extra code because we can now store more than one object at each location in the grid without running into a confusing problem of storing this data in the grid. Instead we can allow each object to store the data themselves.

My simulations can be run and customized based on command line inputs. As part of an extension I looked into and used the jargs package to help me with my command line parsing. The code and documentation for the jargs package was found at http://jargs.sourceforge.net/. This package allowed for much simpler implementation of advanced commandline parsing. The command line options for running the harvest simulation are as follows

```
java HarvestSimulation [-g,--gameType] [-w,--width] [-h,--height] [-d,--denstiy] [-t,--types]
                       [-f,--fiendFood] [-r,--resourceFood] [-a,--alliances] [-p,--pauseTime]
```

Where these parameters are the following:

gameType is the type of simulation (1 = harvest simulation, 2 = social simulation, 3 = game of life)
width and height are the size of the landscape
density is the density of the landscape
fiendFood is the max amount a fiend can start with
resourceFood is the max amount a plant can start with
alliances is only for the social simulation and it is a boolean (true = user wants to set alliances, false = user does not)
pauseTime is the tie to pause between updates
All of these options are optional and if they are not included an appropriate default value will be chosen.

A sample input could be

java HarvestSimulation -g 1 -w 100 -h 80 -d 0.5

This would start a Harvest Simulation with width 100 , height 80 and density 0.5

As part of another extension I refactored my old code for the SocialSimulation and for the Game of Life simulation, allowing these programs to work in this landscape format. I did this by creating 2 new classes SocialAgent and Cell that are both extensions of the SimObject class. The SocialSimulation was easier than the game of Life because the agents are not updated simultaneously but instead one at a time, similar to the harvest simulation. The initialization for these two simulations, simply makes sure to place these objects only on integer location, thus creating a grid in our continuous landscape. The Game of Life simulation places a cell at every Grid location, and then determines whether it is alive or dead based on the density. The getNeighbors methods for these lists needed to also be redesigned so that they only find neighbors that are in the 3x3 grid. These 2 simulations are slower than the earlier simulations because finding neighbors is much more computationally heavy due to the necessity of traversing down the list of all agents on the landscape and checking each one to see if it is a neighbor. In the case of the Game of Life simulation, these appear to have a considerable effect, especially on large landscapes. Additionally the repaint issue is still evident. I am not going to write much on these simulations as they have already been discussed in previous projects. They can be initiated via the command line by changing the gameType parameter. I also, changed a bit of the coding format for the Social Simulation for better OO design. The Alliance information is now stored in the SocialAgent class, and i wrote the code so that the HarvestSimulation and landscape classes will always call the same update methods and the only thing that matters is how the landscape has been initialized in the begining (i.e. which SimObjectLists are populated).

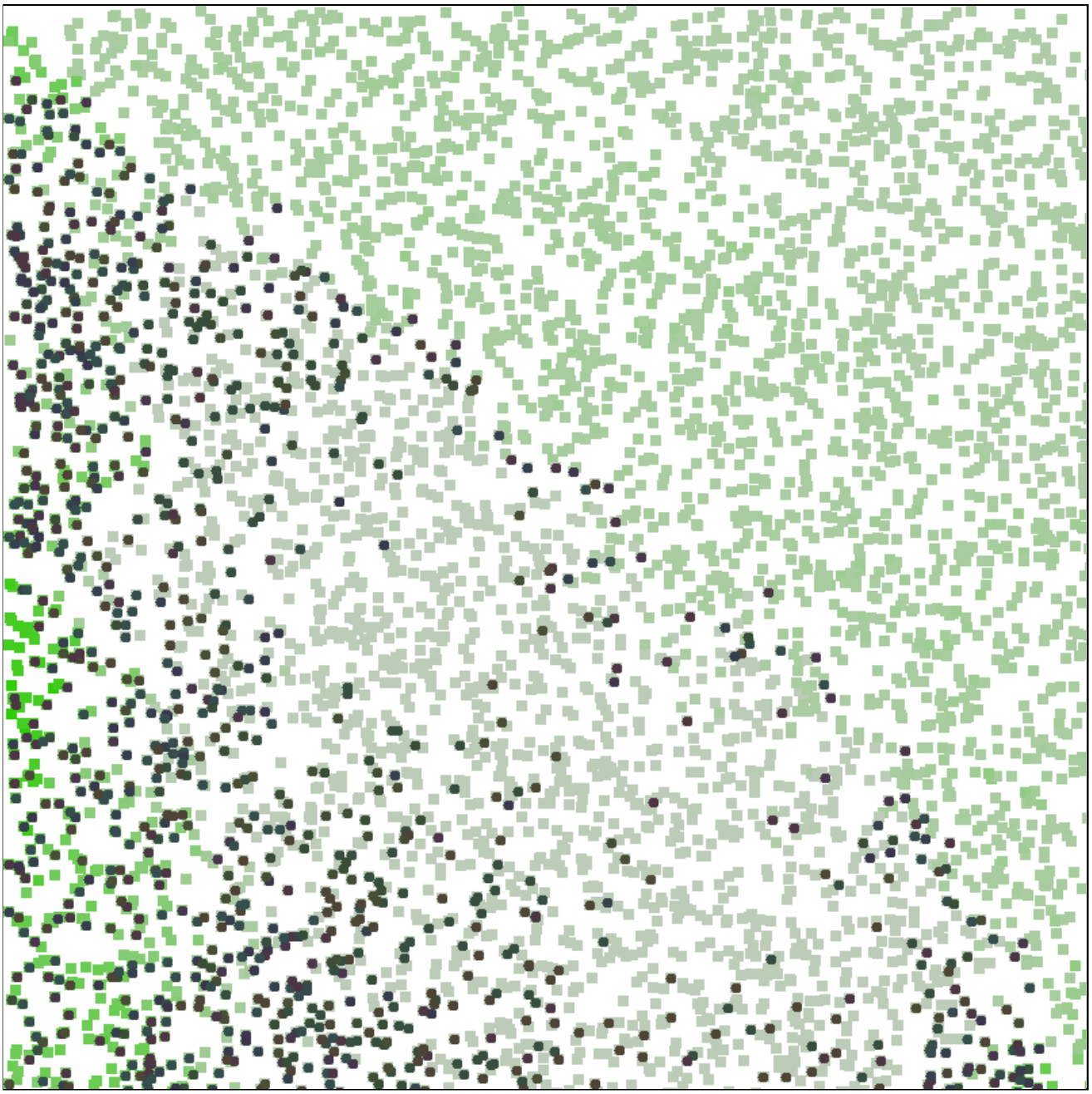
Extensions i coded for this simulation were to implement seasons and to allow the fiends to reproduce. I created 4 different seasons. I divided the grid into 2 sections, above the diagonal and below. In the first season, say summer. The plants below the diagonal grow very

fast and those above grow slowly. In the fall, All plants grow slower but the plants near the diagonal grow faster than those far away. In the Winter the plants below the diagonal grow very fast and the plants above grow slowly. These seasons repeat forever. I did this by storing an integer data called day which is incremented every iteration and it resets at 364. The seasons are split evenly at 91 days each.

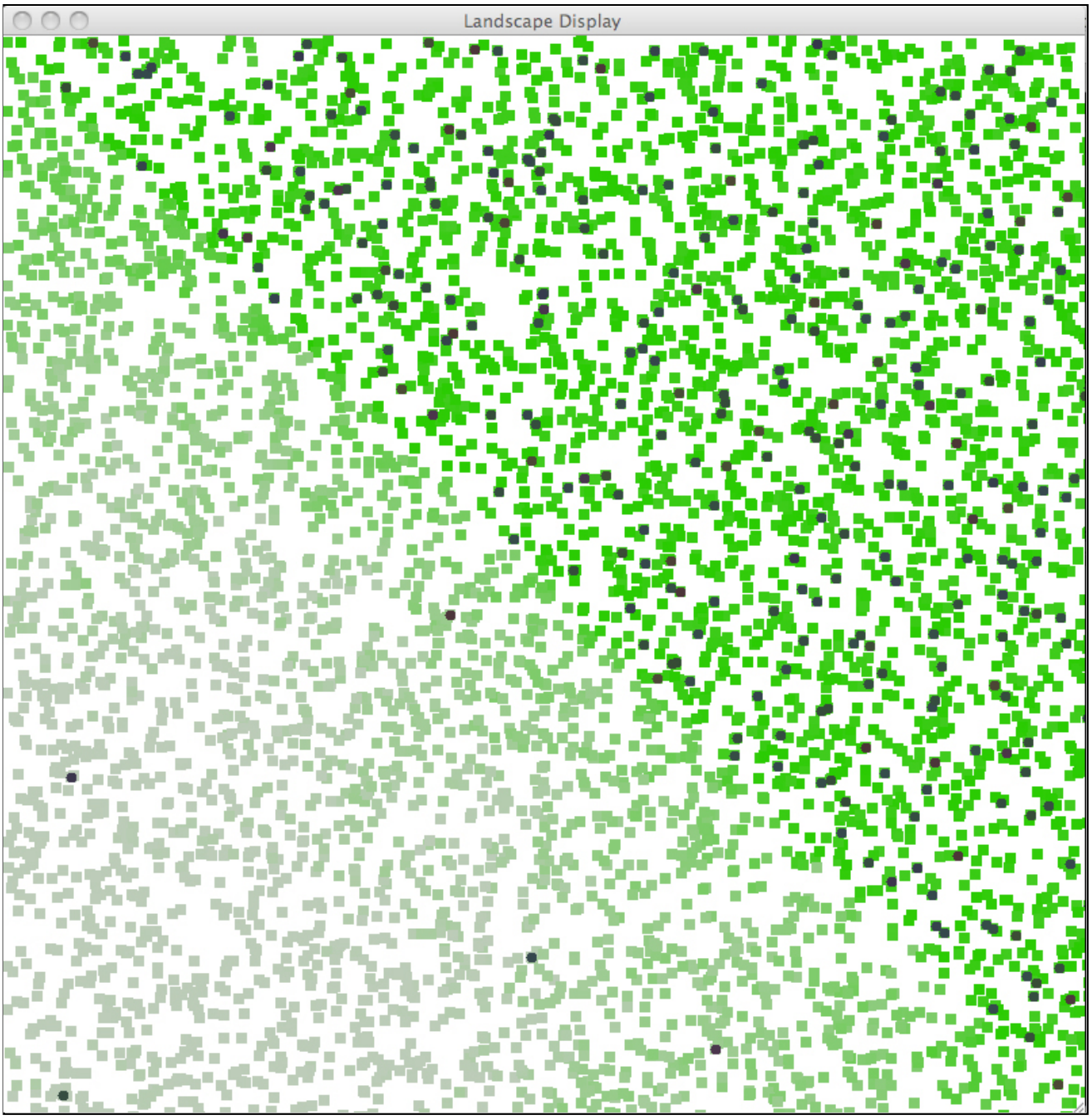
The next extension was to allow the BroccoliFiends to reproduce. I implemented this the same way that it worked in the Game Of Life simulation. When a BroccoliFiend is updating its State it also checks to see if it has any fiends within a radius of 1 unit from it. If it has exactly 2 or 3 neighbors it will produce a new BroccoliFiend at its position. However, the BroccoliFiend needs to supply this new fiend with food (from its own store) and this could kill the fiend after reproduction.

From my simulation, shown below as a gif, the addition of reproduction and seasons created some very interesting social behavior. First it is very clear that the seasons will make our nomadic BroccoliFiends migrate across the landscape when the seasons change and the resource starts growing fast somewhere else. Additionally, we can see that the BroccoliFiends will roproduce around where there is plenty of food but then at somepoint there will be too many fiends and the resources will get depleted and most of the fiends will die off. In the simulation i noticed that the population (which prints after each iteration) went as low as around 45 and to as high as over 1000 BroccoliFiends on the landscape. This is a huge variation in fiends. This was for a 100 by 100 landscape.

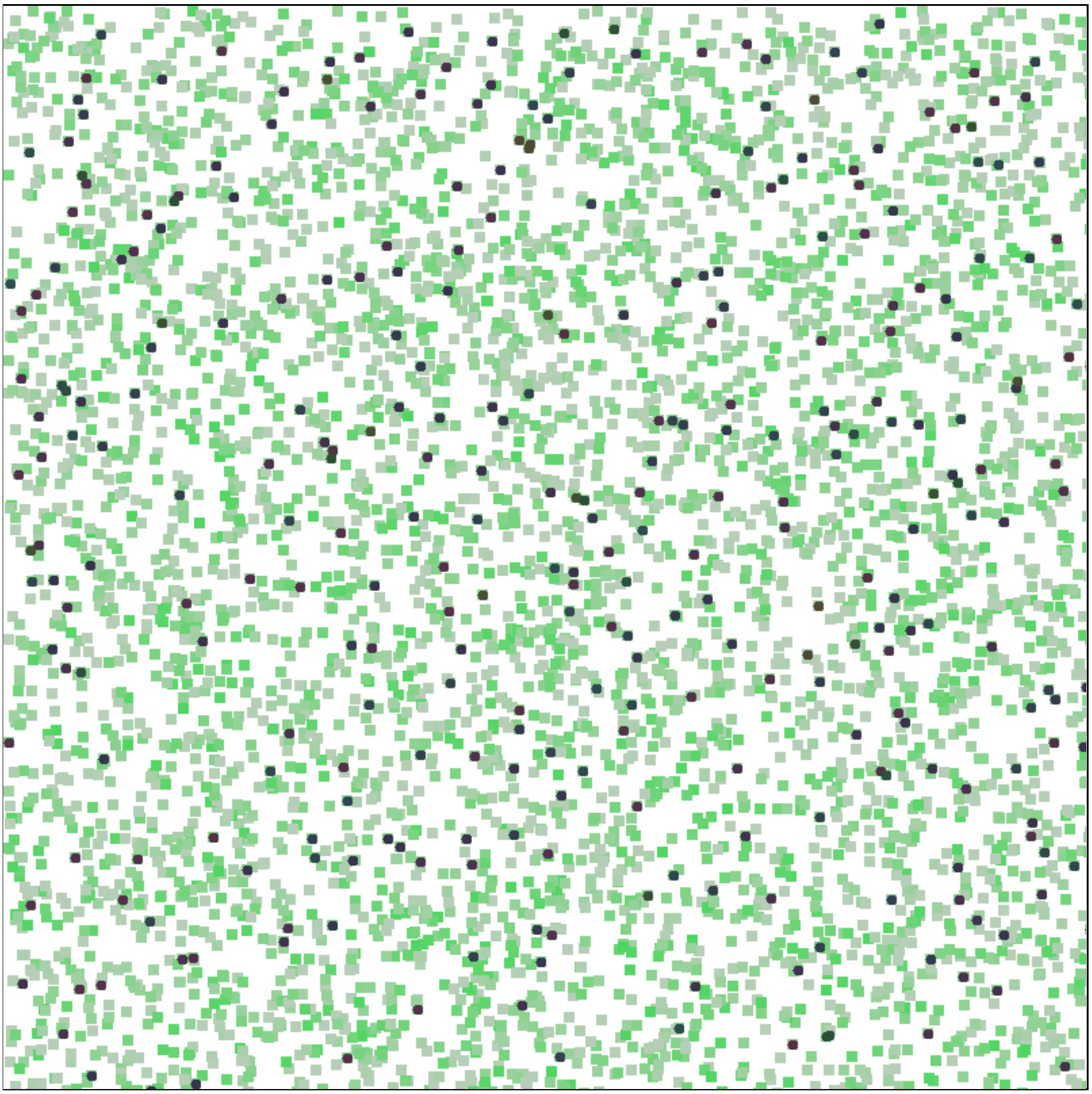
Below is a gif of my Harvest simulation working. These images are taken 100 "days" apart.

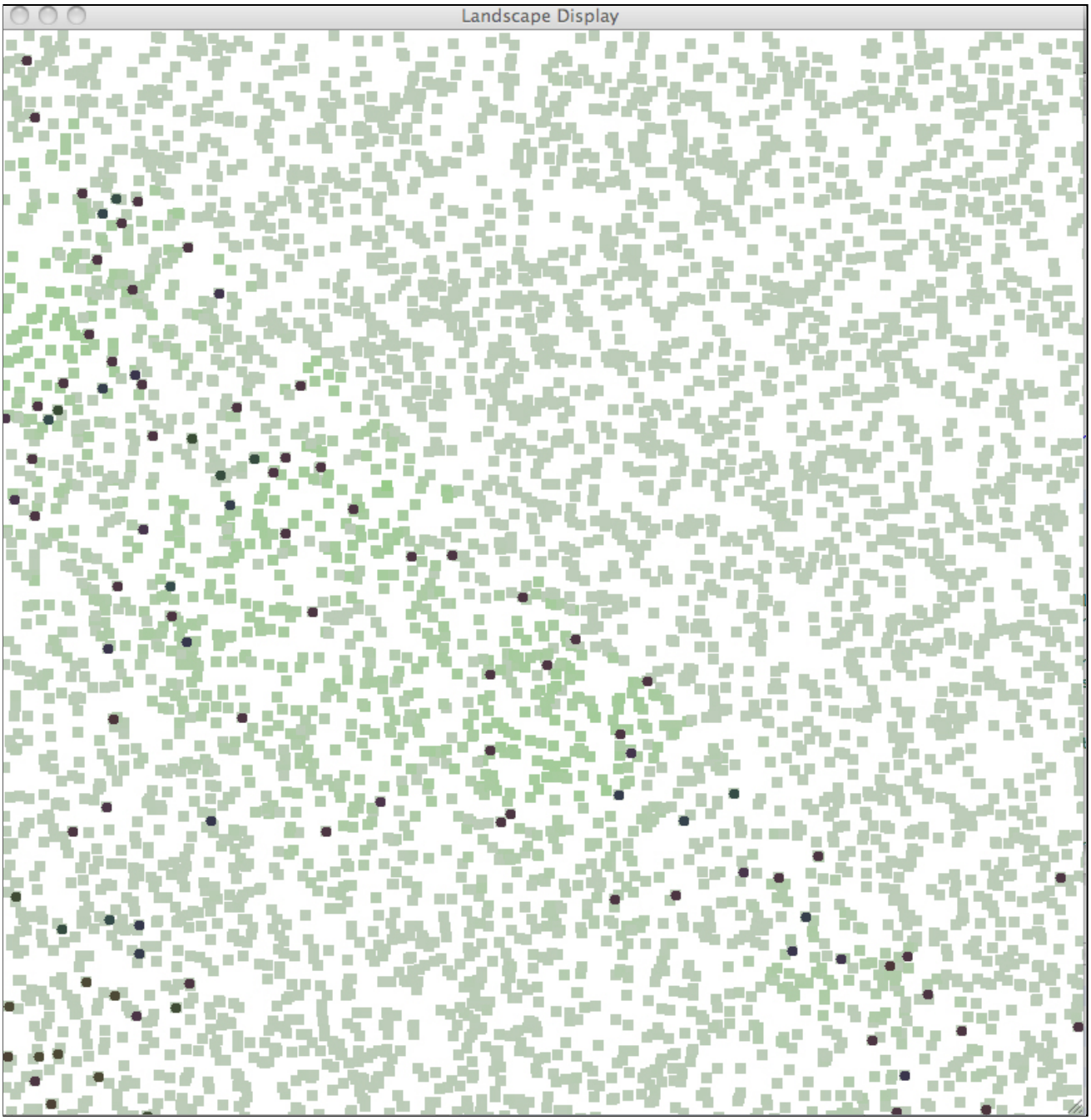


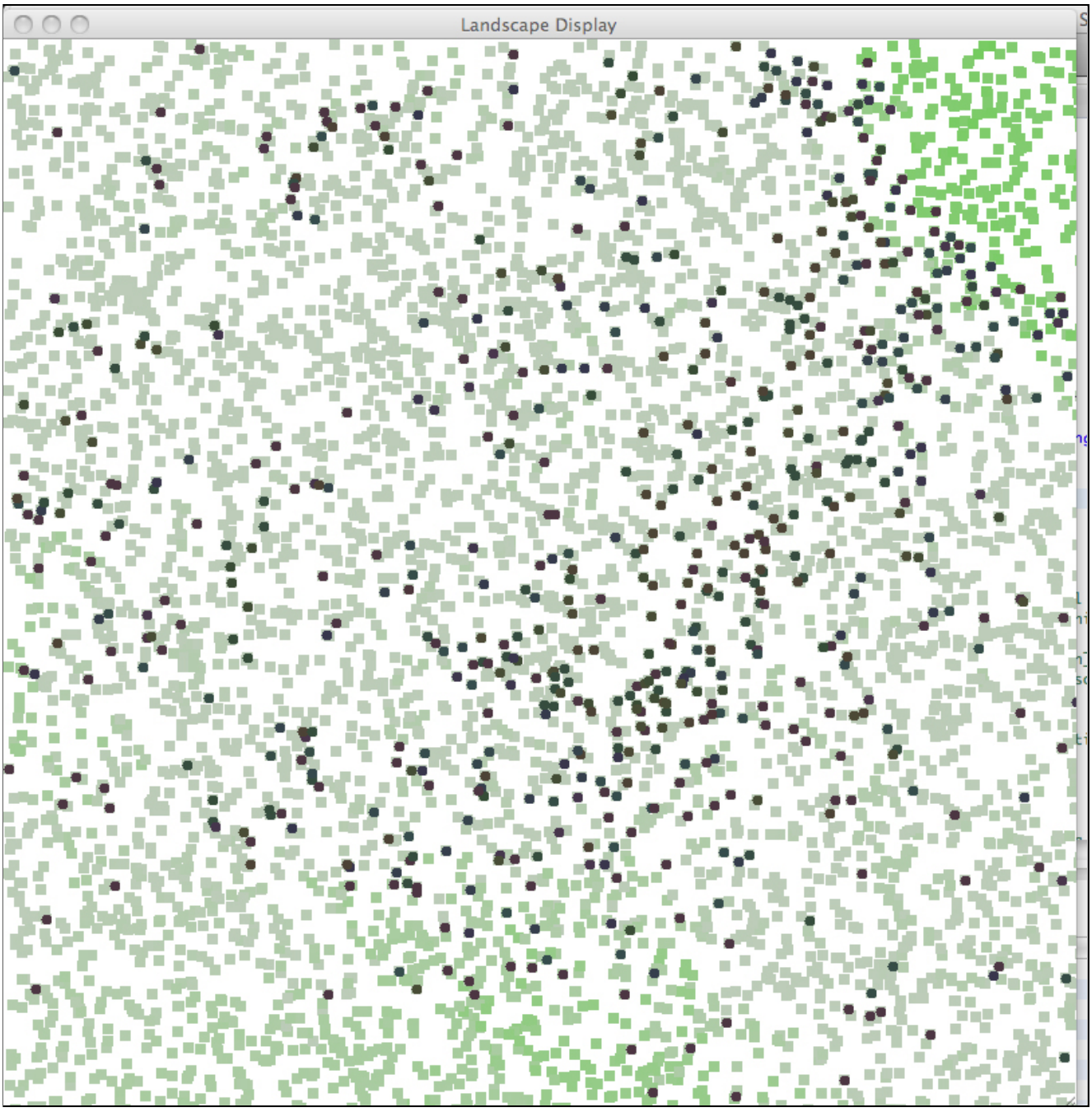
Here is another image showing how the migration looks:



Some more examples of how it changes. The first picture below is the initial landscape. The second shows how the population can get quite small, after the population gets too big and then eats all the food and the fiends die out. The last image shows how the population can then regrow back to huge numbers:




Landscape Display



In this project i learned alot about using Linked LIsts to store data. I can see how this way of data storing allowed us to design more complicated social simulations, by storing all the location data in the objects and then having all the objects on the landscape stored in a list versus in the array grid. However, the limitations are felt when I tried to implement the SocialSimulation and the Game of Life into to this setup. Being forced to traverse down the entire list of objects instead of just the 9 block neighborhood was noticeably slower in large grids. Additionally, I spent a while learning about the jargs package that allowed me to do more advanced command line parsing. In probably a 10th of the coding i did last week in command line parsing i was able to do more intricate parsing using jargs package.

Labels

cs231f11project4