Project 6

Added by Christopher Hoder, last edited by Christopher Hoder on Oct 31, 2011

Cs231 Project 6 Write Up

The goal of this project was to implement a social simulation in a way similar to previous weeks. In this weeks project we investigated how discrete social interactions can evolve to build a tightly-connected social network. The project consisted of populating a landscape with a set number of Users. And initially a user, User A has only 1 other user, User B, it is trying to friend. With each iteration User A moves closer to User B. Once it reaches User B, User A will become friends with User B. Additionally, User A will look at all the friends of User B and find a new target User it is not already a friend with. With these simple rules we can observe how the social interaction unfolds.

My solution implemented this simulation using the same basic framework as in previous projects with SimObjects inhibiting a Landscape. This project however focused more intently on being able to search and sort lists with reasonable time complexity. The search method I implemented was a binary search instead of a less complicated linear search. The linear search would have to go down every element in the list and therefore has a time complexity of O(n). The binary search however, uses the fact that our lists are all sorted by User id's to reduce this time complexity to O(log n) by dividing the indexes that could potentially hold the given id by 1/2 with each loop. This method was implemented recursively. The sorting algorithm I implemented was a version of the common quick-sort algorithm. I modeled mine after the one on the Wikipedia page, http://en.wikipedia.org/wiki/Quicksort. This algorithm uses a pivot point to to put all id's less than it on the left part of the array and all the ids greater than it on the right part of the array. It then recursively does this for progressively smaller and smaller sections of the array. By dividing the array into sections we can get an average time complexity of O(n log n). This algorithm also has been written to sort the array in place so it does not require any additional code to save a new, sorted array.

Before I answer the questions proposed in the project handout, I am going to discuss the extensions i undertook because they are present in the discussion of these questions. First, I implemented a quicksort sorting algorithm which brings the time complexity of the sorting algorithm to O(n log n) on the average case. I also made it so that the colors of the Users depends on the number of friends that they have. The users will be black, gray, blue, orange green , or pink if they have 0,1, 2-5, 6-10, 11-50, or 50 or more friends respectively. This allows us to get a grasp of the number of friends different users have. I also made the program customizable from the command line using the jargs package again. The syntax is below

```
"Usage: NetworkSimulation [-u,--usersNum] [-w,--width] [-h, --height] [-p,--pauseTime]\n" +
"                         [-s,--save] [-n, --saveNum] [-l,--saveLocation] [-r,--randomize] [-m,--randMove]");
```

Where we have the following:

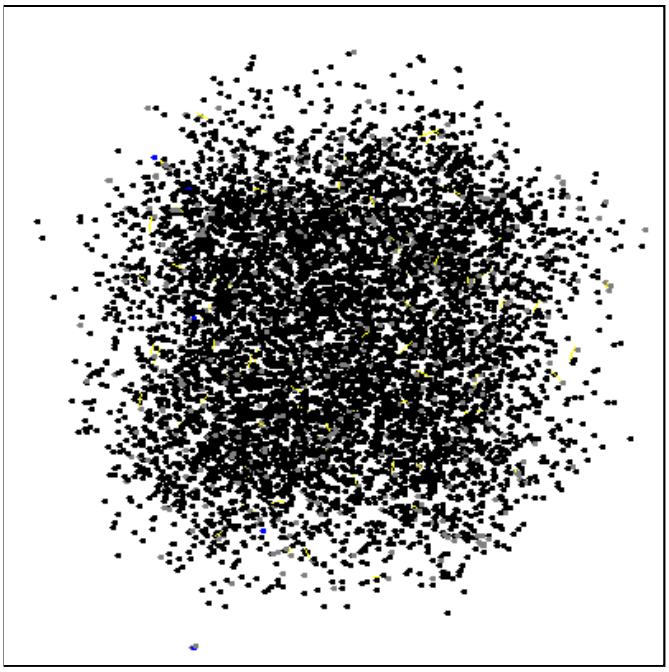
usersNum is the number of users on the landscape
width is the width of the landscape
heigh tis the height of the landscape
pauseTime is the time the program will puase between iterations
save is a boolean to decide if you want to save the average number of friends every saveNum iterations (true to save)
saveNum is an integer to know after every saveNum iterations the program will save the average number of friends to the given path

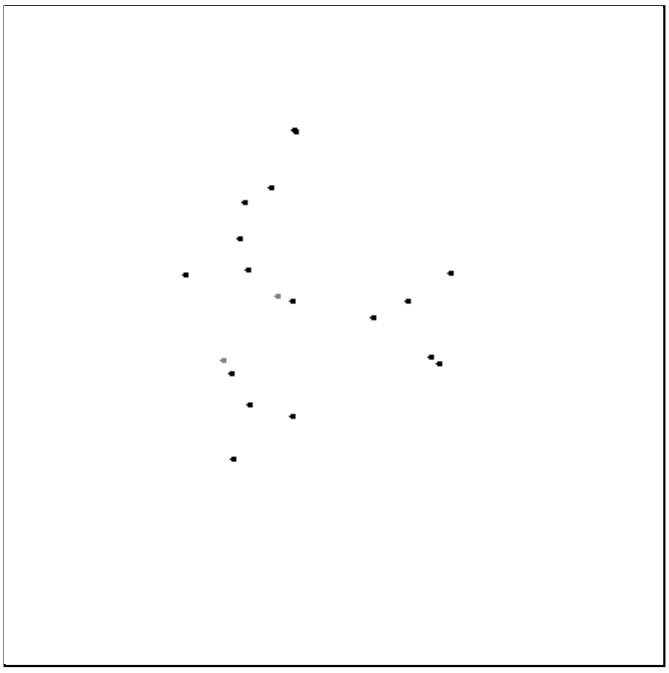
saveLocation is the path to save the text file to

randomize will randomize the updating of the Users, when false it will update them in order of ids.
randMove will when true move the User to a random position after friending another User

All of these options are completely optional and reasonable defaults will be chosen if they are left out. Additionally, I made it so that every 10th user will draw yellow lines between it and all of its friends. Finally, I added code that allows the user to save the average number of friends for each User after every iteration to a text file. This text file can then be easily loaded into Excel or Matlab for data processing. The command line options for this can be read above. In the analysis of my simulation below there are examples of the graphs produced from this data. Help with the syntax for writing to data was found on the following website http://www.daniweb.com/software-development/java/threads/115931.

In the first version of this program, I did not have the Users reset to a random location after making a friend. I ran several simulations with this setup. Below is first a gif of a simulation that started with 8000 users and the second was a simulation started with 20 users.

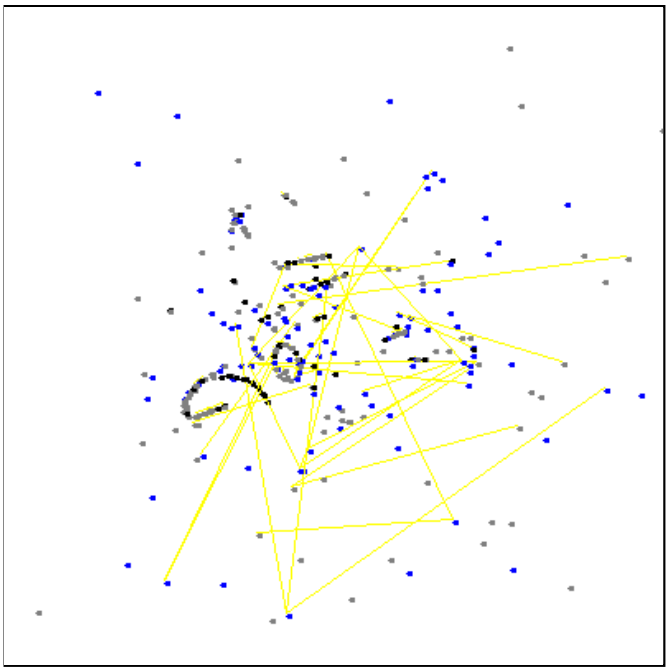




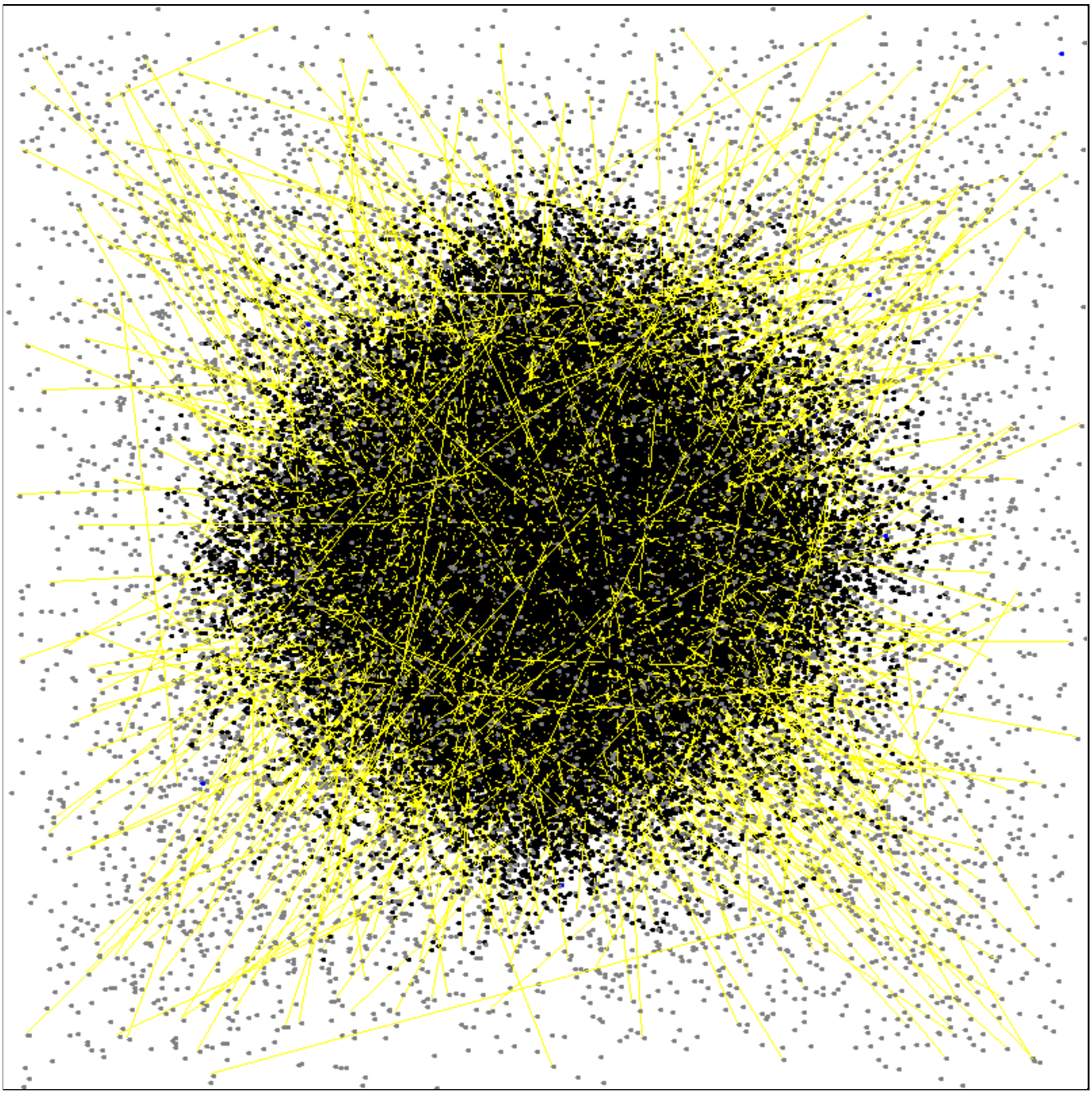
From these two gifs there are some immediate observations that the simulation quickly reaches a steady state with several small clusters. As the simulation gets larger the number of iterations increases as well as the friends made before reaching this situation but ultimately a steady state is achieved. The number of clusters varies with every simulation because they are somewhat dependant on the starting target id's that each user picks. There is nothing in my code that makes sure that you can always travel from friend to friend and go through every User (such as using TargetId's in the beginning). Therefore we can get clusters that form were there are no mutual friends between the clusters. It is also noticeable how the Users immediately migrate toward the middle with almost every simulation. In the larger simulation we can also see "trains" of Users develop. Most likely this is when each User is targeting the user infront of it. I liked the long trains that could form, especially when there were large numbers of Users on the landscape I tried implementing it such that the updating of the Users happens in order of the targetIds but this caused no visible difference.

The simulation below shows the NetworkSimulation at work with 1000 Users initially at work. We can see the connection lines slowly growing. Each separate image is 100 iterations from the previous one, and the yellow lines are coming from only every

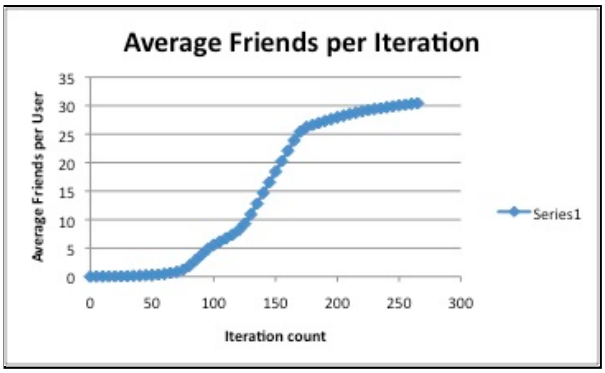
10 Users.



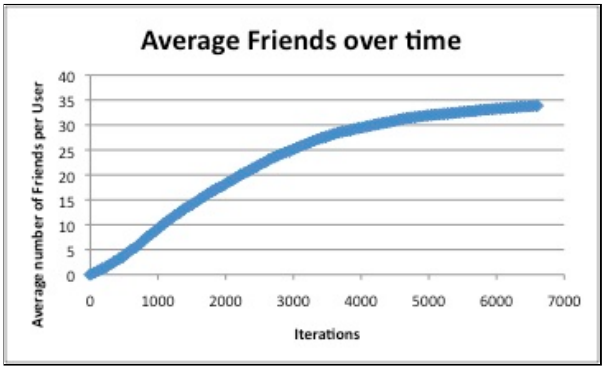
You can also see the colors of the individual users changing as they add more friends to their friends list. The lines of users will become longer and more pronounced when more users are put onto the landscape. Simulations with 100,000 Users provide some interesting trails as well as some circles develop that appear to hold for several hundred iterations before breaking. The gif below shows a simulation where I saved every 100th iteration and started a 200x200 landscape with 100,000 Users and allowed random movements. Very evident in this simulation are the short and long strings of Users that quickly develop. There is even evidence of one of the circles i mentioned in the lower right center of the Landscape.



As part of one of the extensions I created the option which allowed the user to save the average friends per user for each iteration to a text file. Using this I was able to load the data into Excel and make a few graphs of my simulation. The graph below was created by running a simulation with 1000 users and no random repositioning after a user friends another. We can see that this has an S shape graph. This makes sense because at first the users need to get closer together. And since there is no random repositioning after befriending a User, all of the users stay close together, this leads to the rapid growth in the middle. Then after all of the users in each cluster are friends with each other the addition of new friends dies off fast, into a steady state.



This other graph was taken using the same simulation parameters but this time a User repositions itself randomly in the Landscape after befriending a User. We can see here that it requires a much longer time to reach the same average number of users because each user needs to go and find its new target user every time. We can see that the average number of users is beginning to level off at the same size in both simulations at around 35-40. This seems very low considering there are 1000 Users on the Landscape, but again the choice of targetIds at the beginning of the simulation is made completely randomly.



In conclusion I learned a lot about sorting and searching algorithms for this project. I coded several different sorting algorithms before ending with my final version of the quicksort algorithm. Additionally, I was able to look into the different types of search algorithms. Using a binary search method versus a linear search algorithm is probably why my program is able to run at a reasonable speed with 100,000 Users on the Landscape. Running the simulations showed some interesting and unexpected emergent behavior of our Users. The long trains of Users are very interesting and a bunch of fans trailing a popular person comes to mind. Additionally, by reusing old projects Landscape and SimObject code we are able to create more detailed simulations without the tedious repeating of similar code between the simulations.

Labels

cs231f11project6