Maddy's CS231 Project4

Project 4

Summary

The goal of this project was to learn more about LinkedLists, Nodes and Iterations. In order to do this, we created Agent objects to be held in our nodes that had a specific location, color and size (this was part of my extension).

Below I will outline my solution through each class implementation.

LinkedList

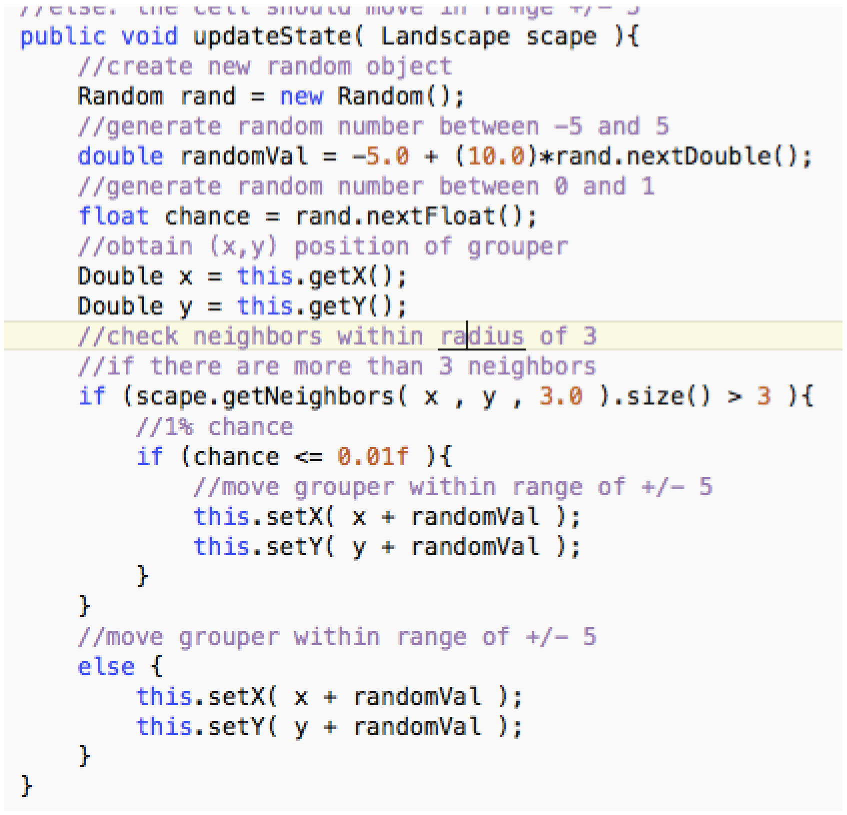
This class was created in lab and implements a singly linked list with an iterator. This class included a method which allowed us to convert the Linked List into an ArrayList both ordered and shuffled.

Agent

Within this class, we created our agent objects. Each agent held a specific (x,y) position on our specified grid. We were able to obtain these locations using our getX() and getY() methods. In this class, our updateState and draw methods do nothing – they will serve a purpose in our extensions of this class.

Grouper

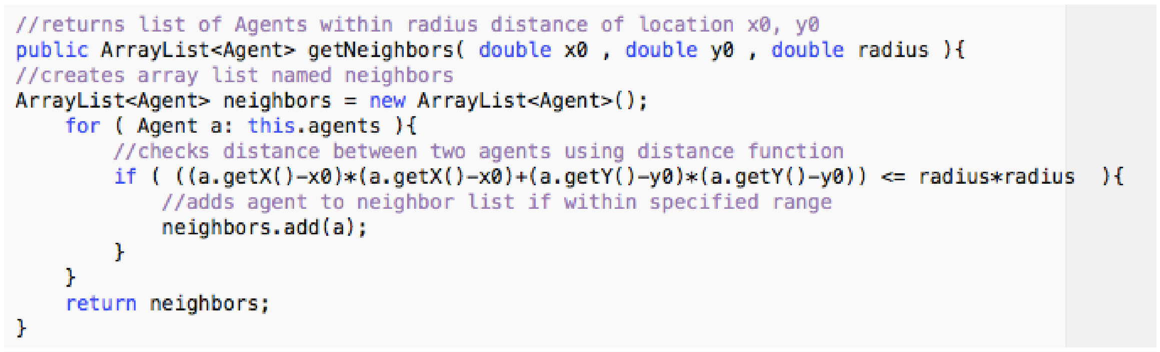
This class is responsible for creating a group of agents and is an extension of our Agent class. In this class, we implemented an updateState method that was responsible for moving the position of a group of specified agents. This method checked to see how many neighbors a given agent had (the neighbor method will be explained later) and moved the cell in a range of +/- 5 with a 1% chance if the given agent had more than 3 neighbors within a radius of 3. If this was not the case for a given agent, it moved in a range of +/- 5 with 100% chance.



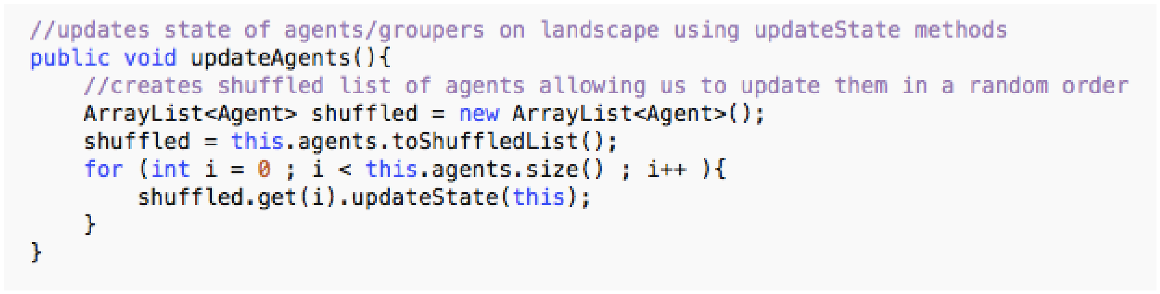
```
//else: the cell should move in range +/- 5
public void updateState( Landscape scape ){
    //create new random object
    Random rand = new Random();
    //generate random number between -5 and 5
    double randomVal = -5.0 + (10.0)*rand.nextDouble();
    //generate random number between 0 and 1
    float chance = rand.nextFloat();
    //obtain (x,y) position of grouper
    Double x = this.getX();
    Double y = this.getY();
    //check neighbors within radius of 3
    //if there are more than 3 neighbors
    if (scape.getNeighbors( x , y , 3.0 ).size() > 3 ){
        //1% chance
        if (chance <= 0.01f ){
            //move grouper within range of +/- 5
            this.setX( x + randomVal );
            this.setY( y + randomVal );
        }
    }
    //move grouper within range of +/- 5
    else {
        this.setX( x + randomVal );
        this.setY( y + randomVal );
    }
}
```

Landscape

Our landscape class was responsible for creating a field in which the agents would appear. Within this class, we were able to add agents and check to see how many agents were present. This class also included a “getNeighbors” method which would return an ArrayList of a given agent’s neighbors. The number of neighbors returned was determined by the specify radius.



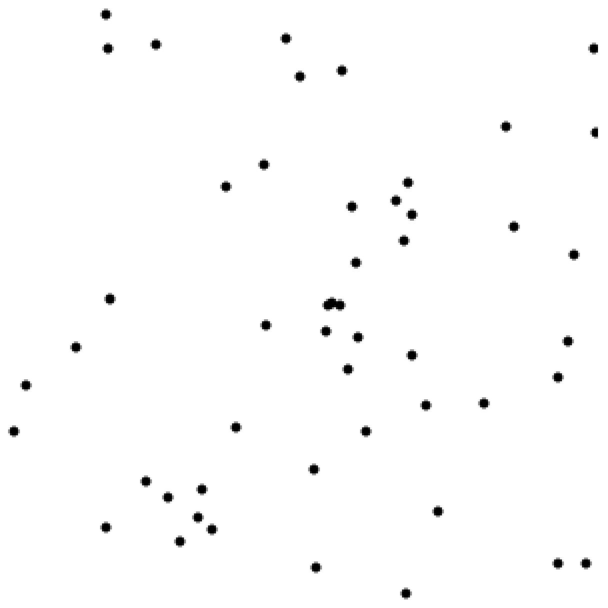
```java
//returns list of Agents within radius distance of location x0, y0
public ArrayList<Agent> getNeighbors( double x0 , double y0 , double radius ){
//creates array list named neighbors
ArrayList<Agent> neighbors = new ArrayList<Agent>();
    for ( Agent a: this.agents ){
        //checks distance between two agents using distance function
        if ( ((a.getX()-x0)*(a.getX()-x0)+(a.getY()-y0)*(a.getY()-y0)) <= radius*radius  ){
            //adds agent to neighbor list if within specified range
            neighbors.add(a);
        }
    }
    return neighbors;
}
```



```java
 //updates state of agents/groupers on landscape using updateState methods
 public void updateAgents(){
     //creates shuffled list of agents allowing us to update them in a random order
     ArrayList<Agent> shuffled = new ArrayList<Agent>();
     shuffled = this.agents.toShuffledList();
     for (int i = 0 ; i < this.agents.size() ; i++ ){
         shuffled.get(i).updateState(this);
     }
 }
```

GrouperSimulation

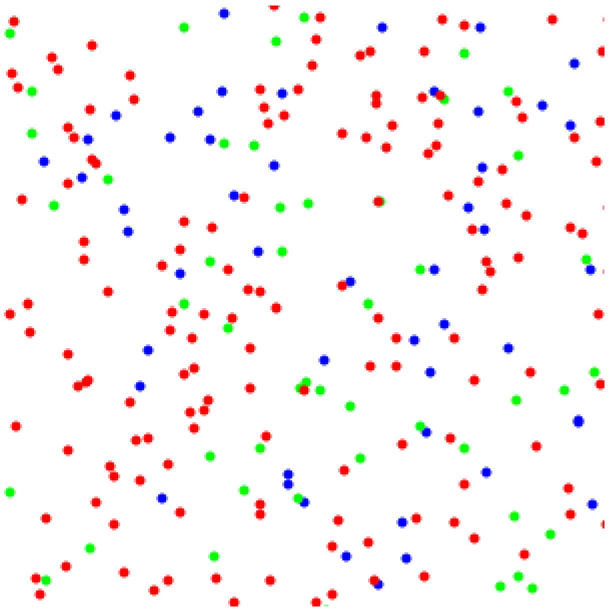
In this class, I chose to control the Landscape size, and number of agents by the command line. I modeled this simulation after LifeSimulation.



CategorizedGrouper

This class allowed us to determine the behavior of the agents based on their “category”. Each category was denoted with a different color. I chose to create 3 categories: 0, 1, and 2. Within this class, we counted the number of agents with the same category and chose to move them via this measurement. If more agents occurred with the same category they moved with a 1% chance, if not, they moved with an 100% chance.

```
//scans through neighbors ArrayList
for (int i = 0 ; i < neighbors.size() ; i++ ){
    //converts agents to CategorizedGroupers
    CategorizedGrouper grouper = (CategorizedGrouper)neighbors.get(i);
    //checks category of given CategorizedObject to see if it matches specified category
    if ( grouper.getCategory() == this.category ){
        catSame++;
    }
    else {
        catDiff++;
    }
}
```



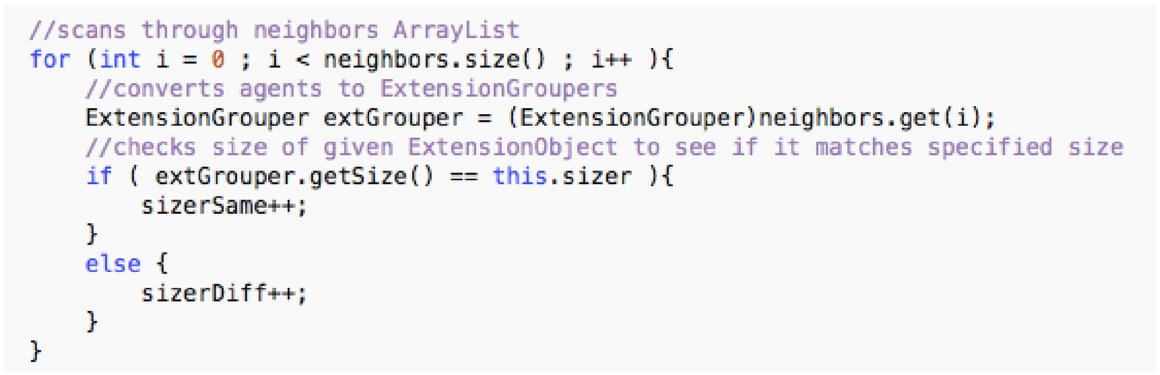
Extension 1

For my first extension I chose to make a mastersimulation class which allowed me to execute all of my grouper types in one simulation. In order to do this, I created an additional command line argument which took the value 0 , 1, or 2 determining the type of grouper I would be using. I then used if and else if statements to determine which type of simulation should be executed, according to the standard input.

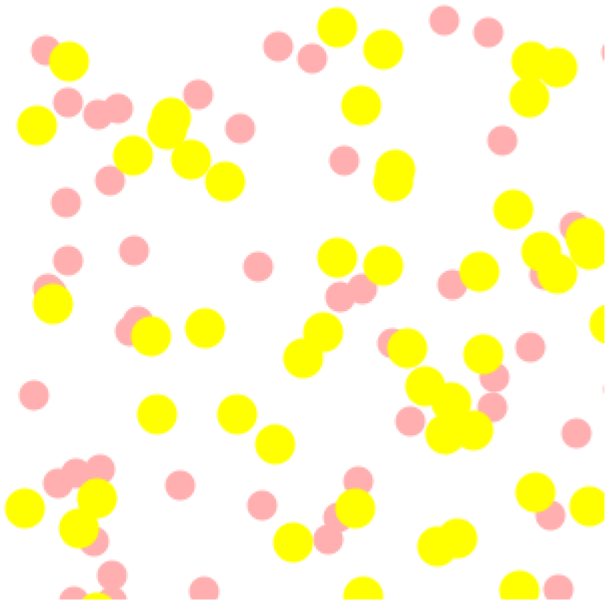
```
//specifies type of simulation
//type either 0 or 1
//0 == categorized grouper
//1 == reg grouper
//2 == extension grouper
int grouperType = Integer.parseInt(args[3]);
```

Extension 2

For my second extension, I created an addition grouper subclass which moved agents based on their sizes.



```
//scans through neighbors ArrayList
for (int i = 0 ; i < neighbors.size() ; i++ ){
    //converts agents to ExtensionGroupers
    ExtensionGrouper extGrouper = (ExtensionGrouper)neighbors.get(i);
    //checks size of given ExtensionObject to see if it matches specified size
    if ( extGrouper.getSize() == this.sizer ){
        sizerSame++;
    }
    else {
        sizerDiff++;
    }
}
```



Conclusion

This project taught me a lot about LinkedLists and how they can be a helpful way to implement various extensions of classes. Through the use of nodes and iterators we were able to sift through a collection of Agents.