Zena's CS231 Project 6

This project involved simulating lines representing queues at grocery stores and such. A customer would appear, choose a line (taking a certain amount of time to do so), and join it, then wait until it was at the end to have its items decremented. After all its items were gone, it would leave the line.

This project did not really have the usual "tasks", and was instead just a general guide with hints on how to build the project. This made it very different and more challenging than most previous projects. I will attempt to explain how I organized my code instead.

My lowest class was the customer, which was drawn as a circle. In the constructor, I checked which strategy integer was passed in as a parameter, and set up a waiting time for each kind. This would be used in update state, which is where I tell the customer how to choose a line, then add it to that line and associate them. Every iteration of the simulation, this update would be called, so the time to wait to get in line would be decremented.

I made a checkout class next, which represented the queue system that the customers would wait in. Once inside a queue, a customer would passively update as a part of it. In the line's update state, it checks if it has customers. If it does, it will decrement the amount of items of the one in front, given they have above 0. If they have 0, the customer will be removed from the overall arraylist of customers as well as the line itself. There was also a method to initially add the customer to the line, which would set the x and y of the customers; the y would depend on the queue's current length, of course.

The next class, as usual, was the landscape, which contained both lines and newly-spawned customers. It had an overall method to update everything, and also a method to add customers. It created some customers and the queues to start, as well. Note that I didn't use a spawn class, instead having a method to add customers in the landscape class. The landscape would be created in the simulation top-level class, along with a landscape display class, like always. The landscape display is based on all of the past ones, and updated to handle this project's classes.

In my simulation class, I had my main loop have an integer representing the strategy for customers to join lines, and changed it there if I needed to. If the timestep was over 0 (because some are already there to start with), it would create more customers. Then, I would repaint and sleep to show that customers are there before they switch into line and never get shown spawning first. I updated all customers after that, and then the lines.

For the last task, it said to find the mean and standard deviation. For this, I created various methods in landscape that took the list of all of the times to get out of line for each customer, then did things like counting them, totaling them and dividing by the number of customers, getting a list of the distances of each from the mean leaving-time, square rooting that, etc. I had the stats print out about every 100 customers, which I checked in a method in landscape that checked the amount of finished customers, and had them print if the number was close to a multiple of 100. My results seemed to be mathematically incorrect, because they were increasing the more customers checked out, but nevertheless, I can discuss the results.

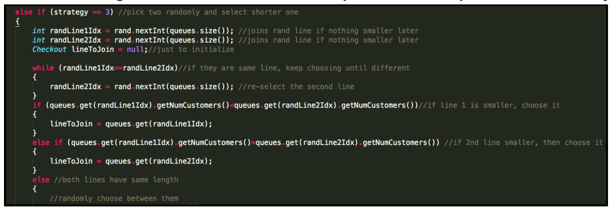
It seemed that choosing randomly, the first strategy, was efficient for time, but would result in the most varying times to check out (ie, bigger standard deviation). The other two (scanning all lines for the shortest and choosing from 2 lines randomly) would probably be about the same. The scanning would take more time steps so it would wouldn't be as efficient, but in the long run it might be because it takes the time to choose the optimal lines to join, getting smaller standard deviations. Choosing the smaller of 2 randomly selected ones would probably be the best in terms of a good choice and speed, because it takes half as long but has half the chance to join the right one, resulting in a standard deviation perhaps a bit larger than the scan-all method.

Below is a gif of the randomly-chosen lines being joined.

Below this is are my two extra strategies as an extension. I made the first one choose the shortest line in terms of number of items, which would be more accurate, probably even more so than just by line length alone. The drawback of this is how much time it would theoretically take. For the second strategy, I chose the line from which the least number of customers had been checked out at any given time. This would depend very heavily on the amount of items each customer had (from 1 to 6), and would also take a while to scan. It turns out that it is not very efficient.

I also did the extension to make it visually interesting, as evidenced by my random-color customers with rectangles next to them representing how many items they had. This was useful for me, as well, because I could track which one was where. To place the item bars in the right spots, I added a part to the customer's draw method that drew items in relation to its position.

Here is an example of the way I selected a customer's line for strategy 3, which was to choose 2 randomly then take the shorter one. It was based off of the strategy from gatherer in the previous project, where we would start at 0 (or in this case, a ridiculously high number because we want to get lines that are smaller than that. I had a default random line to join for when the lines were all size 0 at the start. If the lines were different enough, it would pick two randomly. Then, it would see which one is bigger and store it. I also had a condition for them being the same length, in which I assigned a number to each (either 0 or 1), then randomly chose it and made it join that corresponding one.



```
else if (strategy == 3) //pick two randomly and select shorter one
{
    int randLine1Idx = rand.nextInt(queues.size()); //joins rand line if nothing smaller later
    int randLine2Idx = rand.nextInt(queues.size()); //joins rand line if nothing smaller later
    Checkout lineToJoin = null;//just to initialize

    while (randLine1Idx==randLine2Idx)//if they are same line, keep choosing until different
    {
        randLine2Idx = rand.nextInt(queues.size()); //re-select the second line
    }
    if (queues.get(randLine1Idx).getNumCustomers()<queues.get(randLine2Idx).getNumCustomers())//if line 1 is smaller, choose it
    {
        lineToJoin = queues.get(randLine1Idx);
    }
    else if (queues.get(randLine1Idx).getNumCustomers()>queues.get(randLine2Idx).getNumCustomers()) //if 2nd line smaller, then choose it
    {
        lineToJoin = queues.get(randLine2Idx);
    }
    else //both lines have same length
    {
        //randomly choose between them
```

This project taught me a lot. First of all, it taught me the importance of organization and planning, because it lacked the usual walkthrough style instructions and gave us free reign. It was strange to use my own discretion for many parts, but in a way, this taught me a lot about my strengths and weaknesses and how much I know. It was challenging dividing up the jobs into classes and passing in various things as parameters all over, but I eventually got it and was able to figure out my own style.

As usual, special thanks to Melody for helping and encouraging me during this ?project.