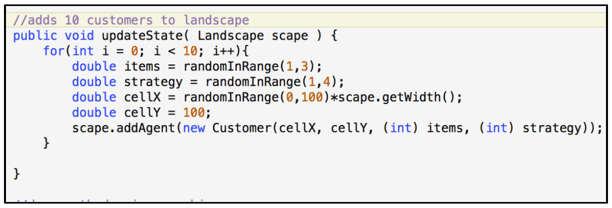
Project 6

PROJECT 6

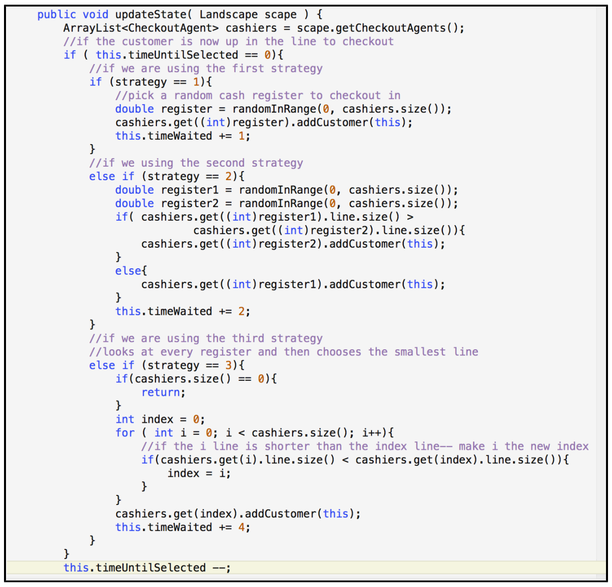
In this project, we started using a new type of data structure called a "Queue" that creates a list of items and implements a first-in-first-out behavior. The goal of this project was to design a multi-station checkout simulation. The simulation had to include Customers and CheckoutAgents. To create the Customers, we implemented a Spawner class. In this project, we continued to use our Landscape and Cell classes to organize our simulation but this week we used a DoubleLinkedList Queue to create checkout stations.

Task 1. The goal of this first task was to design and create a Spawner class that creates Customer agents. The purpose of this Spawner class is that it will be written in the final CheckoutSimulation and take care of creating the Customer agents. Here is the code snippet for the updateState method of my Spawner class:



```
//adds 10 customers to landscape
public void updateState( Landscape scape ) {
    for(int i = 0; i < 10; i++){
        double items = randomInRange(1,3);
        double strategy = randomInRange(1,4);
        double cellX = randomInRange(0,100)*scape.getWidth();
        double cellY = 100;
        scape.addAgent(new Customer(cellX, cellY, (int) items, (int) strategy));
    }

}
```

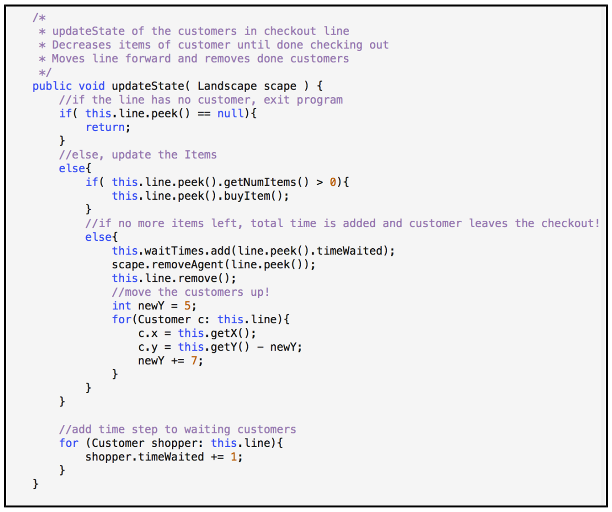
Task 2. In this task, we had to implement our Customer class. To do this, I had to give my class a lot of fields to keep track of the phase, the time until selected, the total wait time, the number of items, and the strategy type! Then, in the constructor I had to write conditionals that added different amount of time until selected increments based on the strategy. Then, for each Customer, once the customer has been selected and the time until selected is 0, then each strategy has its own unique updateState rules. Here is the updateState method for my Customer class:



```
    public void updateState( Landscape scape ) {
        ArrayList<CheckoutAgent> cashiers = scape.getCheckoutAgents();
        //if the customer is now up in the line to checkout
        if ( this.timeUntilSelected == 0){
            //if we are using the first strategy
            if (strategy == 1){
                //pick a random cash register to checkout in
                double register = randomInRange(0, cashiers.size());
                cashiers.get((int)register).addCustomer(this);
                this.timeWaited += 1;
            }
            //if we using the second strategy
            else if (strategy == 2){
                double register1 = randomInRange(0, cashiers.size());
                double register2 = randomInRange(0, cashiers.size());
                if( cashiers.get((int)register1).line.size() >
                            cashiers.get((int)register2).line.size()){
                    cashiers.get((int)register2).addCustomer(this);
                }
                else{
                    cashiers.get((int)register1).addCustomer(this);
                }
                this.timeWaited += 2;
            }
            //if we are using the third strategy
            //looks at every register and then chooses the smallest line
            else if (strategy == 3){
                if(cashiers.size() == 0){
                    return;
                }
                int index = 0;
                for ( int i = 0; i < cashiers.size(); i++){
                    //if the i line is shorter than the index line-- make i the new index
                    if(cashiers.get(i).line.size() < cashiers.get(index).line.size()){
                        index = i;
                    }
                }
                cashiers.get(index).addCustomer(this);
                this.timeWaited += 4;
            }
        }
        this.timeUntilSelected --;
```

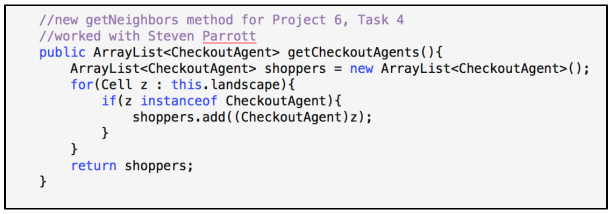
Task 3. This task called for us to create a CheckoutAgent class. This class maintains a queue of Customers. The updateState method of CheckoutAgent needs to implement the following rules:

- There is no one in line: do nothing.
- There is a Customer at the head of the line with more than zero items in their basket: decrement the number of items in the Customer's basket.
- There is a Customer at the head of the line with zero items in their basket: remove this Customer from the checkout queue and update the positions of all other Customer agents in the queue.Here is the updateState snippet:

- 

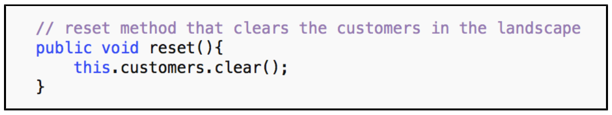
```
/*
 * updateState of the customers in checkout line
 * Decreases items of customer until done checking out
 * Moves line forward and removes done customers
 */
public void updateState( Landscape scape ) {
    //if the line has no customer, exit program
    if( this.line.peek() == null){
        return;
    }
    //else, update the Items
    else{
        if( this.line.peek().getNumItems() > 0){
            this.line.peek().buyItem();
        }
        //if no more items left, total time is added and customer leaves the checkout!
        else{
            this.waitTimes.add(line.peek().timeWaited);
            scape.removeAgent(line.peek());
            this.line.remove();
            //move the customers up!
            int newY = 5;
            for(Customer c: this.line){
                c.x = this.getX();
                c.y = this.getY() - newY;
                newY += 7;
            }
        }
    }

    //add time step to waiting customers
    for (Customer shopper: this.line){
        shopper.timeWaited += 1;
    }
}
```

Task 4. This task asked me to create a new getNeighbors method in Landscape that returns a list of checkout agents instead of neighbors. Here is my code for my getCheckoutAgents method:



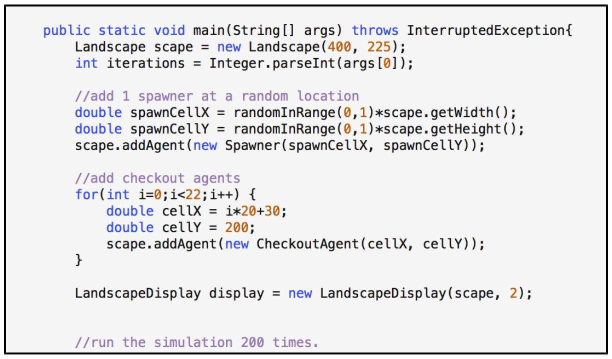
```
//new getNeighbors method for Project 6, Task 4
//worked with Steven Parrott
public ArrayList<CheckoutAgent> getCheckoutAgents(){
    ArrayList<CheckoutAgent> shoppers = new ArrayList<CheckoutAgent>();
    for(Cell z : this.landscape){
        if(z instanceof CheckoutAgent){
            shoppers.add((CheckoutAgent)z);
        }
    }
    return shoppers;
}
```

Also, I changed the reset method in landscape to clear the customers in the landscape in a list of customers:



```
// reset method that clears the customers in the landscape
public void reset(){
    this.customers.clear();
}
```

Task 5. This task wrapped up the project with a CheckoutSimulation class. This class required the Spawner and the CheckoutAgent classes. Here is a snippet from the main method:

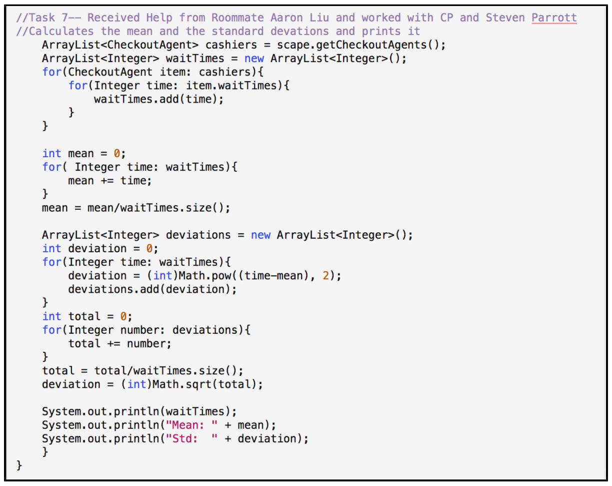


```
public static void main(String[] args) throws InterruptedException{
    Landscape scape = new Landscape(400, 225);
    int iterations = Integer.parseInt(args[0]);

    //add 1 spawner at a random location
    double spawnCellX = randomInRange(0,1)*scape.getWidth();
    double spawnCellY = randomInRange(0,1)*scape.getHeight();
    scape.addAgent(new Spawner(spawnCellX, spawnCellY));

    //add checkout agents
    for(int i=0;i<22;i++) {
        double cellX = i*20+30;
        double cellY = 200;
        scape.addAgent(new CheckoutAgent(cellX, cellY));
    }

    LandscapeDisplay display = new LandscapeDisplay(scape, 2);


    //run the simulation 200 times.
```

When I ran my simulation, here is what I got. Here is the GIF:
(**note**-- i am not sure why the first couple images in my GIF are funky and skewed... the 000.png and the 001.png are totally normal...)

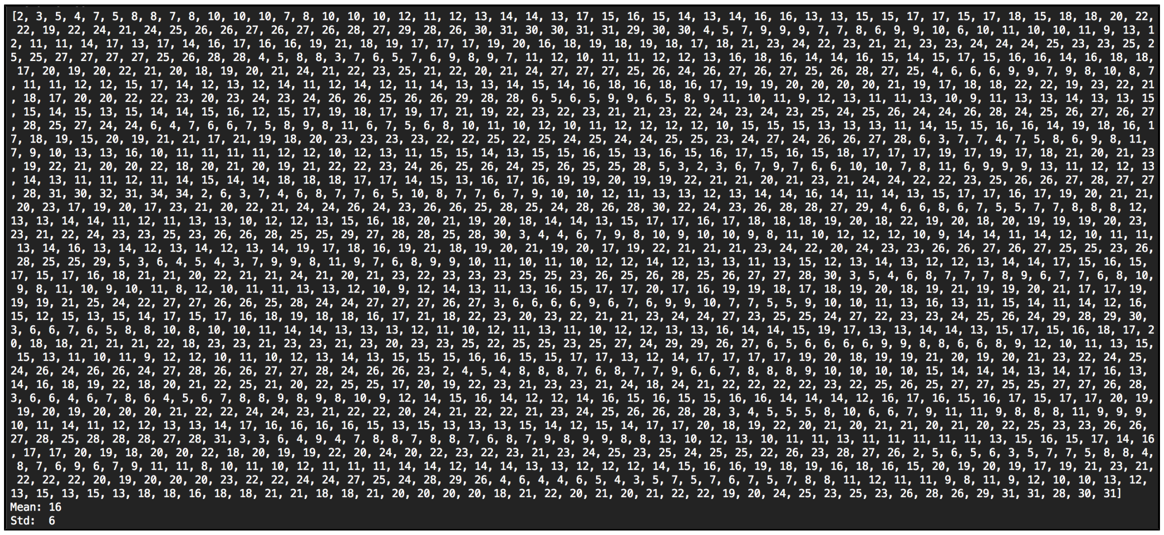
please ignore this minor problem of the first two images in my GIF

Task 6. This task asked us to calculate the standard deviation and the mean of 2000 customers ran through the simulation. I recieved guidance and help from CP Majgaard and Aaron Liu for this task. Here is my code for the math:



```java
//Task 7-- Received Help from Roommate Aaron Liu and worked with CP and Steven Parrott
//Calculates the mean and the standard devations and prints it
    ArrayList<CheckoutAgent> cashiers = scape.getCheckoutAgents();
    ArrayList<Integer> waitTimes = new ArrayList<Integer>();
    for(CheckoutAgent item: cashiers){
        for(Integer time: item.waitTimes){
            waitTimes.add(time);
        }
    }

    int mean = 0;
    for( Integer time: waitTimes){
        mean += time;
    }
    mean = mean/waitTimes.size();

    ArrayList<Integer> deviations = new ArrayList<Integer>();
    int deviation = 0;
    for(Integer time: waitTimes){
        deviation = (int)Math.pow((time-mean), 2);
        deviations.add(deviation);
    }
    int total = 0;
    for(Integer number: deviations){
        total += number;
    }
    total = total/waitTimes.size();
    deviation = (int)Math.sqrt(total);

    System.out.println(waitTimes);
    System.out.println("Mean: " + mean);
    System.out.println("Std:  " + deviation);
    }
}
```

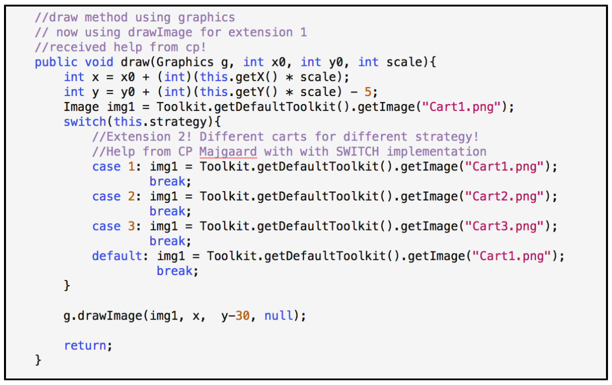
Finally, here is what this code generated and printed to the terminal. Standard Deviation of 6 and a Mean of 16 :



```
[2, 3, 5, 4, 7, 5, 8, 8, 7, 8, 10, 10, 10, 7, 8, 10, 10, 10, 12, 11, 12, 13, 14, 14, 13, 17, 15, 16, 15, 14, 13, 14, 16, 16, 13, 13, 15, 15, 17, 17, 15, 17, 18, 15, 18, 18, 20, 22,
 22, 19, 22, 24, 21, 24, 25, 26, 26, 27, 26, 27, 26, 28, 27, 29, 28, 26, 30, 31, 30, 30, 31, 31, 29, 30, 30, 4, 5, 7, 9, 9, 9, 7, 7, 8, 6, 9, 9, 10, 6, 10, 11, 10, 10, 11, 9, 13, 1
2, 11, 11, 14, 17, 13, 17, 14, 16, 17, 16, 16, 19, 21, 18, 19, 17, 17, 17, 19, 20, 16, 18, 19, 18, 19, 18, 17, 18, 21, 23, 24, 22, 23, 21, 21, 23, 23, 24, 24, 24, 25, 23, 23, 25, 2
5, 25, 27, 27, 27, 27, 25, 26, 28, 28, 4, 5, 8, 8, 3, 7, 6, 5, 7, 6, 9, 8, 9, 7, 11, 12, 10, 11, 11, 12, 12, 13, 16, 18, 16, 14, 14, 16, 15, 14, 15, 17, 15, 16, 16, 14, 16, 18, 18,
 17, 20, 19, 20, 22, 21, 20, 18, 19, 20, 21, 24, 21, 22, 23, 25, 21, 22, 20, 21, 24, 27, 27, 27, 25, 26, 24, 26, 27, 26, 27, 25, 26, 28, 27, 25, 4, 6, 6, 6, 9, 9, 7, 9, 8, 10, 8, 7
, 11, 11, 12, 12, 15, 17, 14, 12, 13, 12, 14, 11, 12, 14, 12, 11, 14, 13, 13, 14, 15, 14, 16, 18, 16, 18, 16, 17, 19, 19, 20, 20, 20, 20, 21, 19, 17, 18, 18, 22, 22, 19, 23, 22, 21
, 18, 17, 20, 20, 22, 22, 23, 20, 23, 24, 23, 24, 26, 26, 25, 26, 26, 29, 28, 28, 6, 5, 6, 5, 9, 9, 6, 5, 8, 9, 11, 10, 11, 9, 12, 13, 11, 11, 13, 10, 9, 11, 13, 13, 14, 13, 13, 15
, 15, 14, 15, 13, 15, 14, 14, 15, 16, 12, 15, 17, 19, 18, 17, 19, 17, 21, 19, 22, 23, 22, 23, 21, 21, 23, 22, 24, 23, 24, 23, 25, 24, 25, 26, 24, 24, 26, 28, 24, 25, 26, 27, 26, 27
, 28, 25, 27, 24, 24, 6, 4, 7, 6, 6, 7, 5, 8, 9, 8, 11, 6, 7, 5, 6, 8, 10, 11, 10, 12, 10, 11, 12, 12, 12, 12, 10, 15, 15, 15, 13, 13, 13, 11, 14, 15, 15, 16, 16, 14, 19, 18, 16, 1
7, 18, 19, 15, 20, 19, 21, 21, 17, 21, 19, 18, 20, 23, 23, 23, 23, 22, 22, 25, 22, 25, 24, 25, 24, 24, 25, 25, 23, 24, 27, 24, 26, 26, 27, 28, 6, 3, 7, 7, 4, 7, 5, 8, 6, 9, 8, 11,
7, 9, 10, 13, 13, 16, 10, 11, 11, 11, 11, 12, 12, 10, 12, 13, 11, 15, 15, 14, 13, 15, 15, 16, 15, 13, 16, 15, 16, 17, 15, 16, 15, 18, 17, 17, 17, 19, 17, 19, 17, 18, 21, 20, 21, 23
, 19, 22, 21, 20, 20, 22, 18, 20, 21, 20, 19, 21, 22, 22, 23, 24, 26, 25, 26, 24, 25, 26, 25, 25, 28, 5, 3, 2, 3, 6, 7, 9, 7, 6, 6, 10, 10, 7, 8, 11, 6, 9, 9, 9, 13, 11, 12, 12, 13
, 14, 13, 11, 11, 12, 11, 14, 15, 14, 14, 18, 18, 18, 17, 17, 14, 15, 13, 16, 17, 16, 19, 19, 20, 19, 19, 22, 21, 21, 20, 21, 23, 21, 24, 24, 22, 22, 23, 25, 26, 26, 27, 28, 27, 27
, 28, 31, 30, 32, 31, 34, 34, 2, 6, 3, 7, 4, 6, 8, 7, 7, 6, 5, 10, 8, 7, 7, 6, 7, 9, 10, 10, 12, 11, 13, 13, 12, 13, 14, 14, 16, 14, 11, 14, 13, 15, 17, 17, 16, 17, 19, 20, 21, 21,
 20, 23, 17, 19, 20, 17, 23, 21, 20, 22, 21, 24, 24, 26, 24, 23, 26, 26, 25, 28, 25, 24, 28, 26, 28, 30, 22, 24, 23, 26, 28, 28, 27, 29, 4, 6, 6, 8, 6, 7, 5, 5, 7, 7, 8, 8, 8, 12,
13, 13, 14, 14, 11, 12, 11, 13, 13, 10, 12, 12, 13, 15, 16, 18, 20, 21, 19, 20, 18, 14, 14, 13, 15, 17, 17, 16, 17, 18, 18, 18, 19, 20, 18, 22, 19, 20, 18, 20, 19, 19, 19, 20, 23,
23, 21, 22, 24, 23, 23, 25, 23, 26, 26, 28, 25, 25, 29, 27, 28, 28, 25, 28, 30, 3, 4, 4, 6, 7, 9, 8, 10, 9, 10, 10, 9, 8, 11, 10, 12, 12, 12, 10, 9, 14, 14, 11, 14, 12, 10, 11, 11,
 13, 14, 16, 13, 14, 12, 13, 14, 12, 13, 14, 19, 17, 18, 16, 19, 21, 18, 19, 20, 21, 19, 20, 17, 19, 22, 21, 21, 21, 23, 24, 22, 20, 24, 23, 23, 26, 26, 27, 26, 27, 25, 25, 23, 26,
 28, 25, 25, 29, 5, 3, 6, 4, 5, 4, 3, 7, 9, 9, 8, 11, 9, 7, 6, 8, 9, 9, 10, 11, 10, 11, 10, 12, 12, 14, 12, 13, 13, 11, 13, 15, 12, 13, 14, 13, 12, 12, 13, 14, 14, 17, 15, 16, 15,
17, 15, 17, 16, 18, 21, 21, 20, 22, 21, 21, 24, 21, 20, 21, 23, 22, 23, 23, 23, 25, 25, 23, 26, 25, 26, 28, 25, 26, 27, 27, 28, 30, 3, 5, 4, 6, 8, 7, 7, 7, 8, 9, 6, 7, 7, 6, 8, 10,
 9, 8, 11, 10, 9, 10, 11, 8, 12, 10, 11, 11, 13, 13, 12, 10, 9, 12, 14, 13, 11, 13, 16, 15, 17, 17, 20, 17, 16, 19, 19, 18, 17, 18, 19, 20, 18, 19, 21, 19, 19, 20, 21, 17, 17, 19,
19, 19, 21, 25, 24, 22, 27, 27, 26, 26, 25, 28, 24, 24, 27, 27, 27, 26, 27, 3, 6, 6, 6, 6, 9, 6, 7, 6, 9, 9, 10, 7, 7, 5, 5, 9, 10, 10, 11, 13, 16, 13, 11, 15, 14, 11, 14, 12, 16,
15, 12, 15, 13, 15, 14, 17, 15, 17, 16, 18, 19, 18, 18, 16, 17, 21, 18, 22, 23, 20, 23, 22, 21, 21, 23, 24, 24, 27, 23, 25, 25, 24, 27, 22, 23, 23, 24, 25, 26, 24, 29, 28, 29, 30,
3, 6, 6, 7, 6, 5, 8, 8, 10, 8, 10, 10, 11, 14, 14, 13, 13, 13, 12, 11, 10, 12, 11, 13, 11, 10, 12, 12, 13, 13, 16, 14, 14, 15, 19, 17, 13, 13, 14, 14, 13, 15, 17, 15, 16, 18, 17, 2
0, 18, 18, 21, 21, 21, 22, 18, 23, 23, 21, 23, 23, 21, 23, 20, 23, 23, 25, 22, 25, 25, 23, 25, 27, 24, 29, 29, 26, 27, 6, 5, 6, 6, 6, 6, 9, 9, 8, 8, 6, 6, 8, 9, 12, 10, 11, 13, 15,
 15, 13, 11, 10, 11, 9, 12, 12, 10, 11, 10, 12, 13, 14, 13, 15, 15, 15, 16, 16, 15, 15, 17, 17, 13, 12, 14, 17, 17, 17, 17, 19, 20, 18, 19, 19, 21, 20, 19, 20, 21, 23, 22, 24, 25,
24, 26, 24, 26, 26, 24, 27, 28, 26, 26, 27, 27, 28, 24, 26, 26, 23, 2, 4, 5, 4, 8, 8, 8, 7, 6, 8, 7, 7, 9, 6, 6, 7, 8, 8, 8, 9, 10, 10, 10, 10, 15, 14, 14, 14, 13, 14, 17, 16, 13,
14, 16, 18, 19, 22, 18, 20, 21, 22, 25, 21, 20, 22, 25, 25, 17, 20, 19, 22, 23, 21, 23, 23, 21, 24, 18, 24, 21, 22, 22, 22, 22, 23, 22, 25, 26, 25, 27, 27, 25, 25, 27, 27, 26, 28,
3, 6, 6, 4, 6, 7, 8, 6, 4, 5, 6, 7, 8, 8, 9, 8, 9, 8, 10, 9, 12, 14, 15, 16, 14, 12, 12, 14, 16, 15, 16, 15, 15, 16, 16, 14, 14, 14, 12, 16, 17, 16, 15, 16, 17, 15, 17, 17, 20, 19,
 19, 20, 19, 20, 20, 20, 21, 22, 22, 24, 24, 23, 21, 22, 22, 20, 24, 21, 22, 22, 21, 23, 24, 25, 26, 26, 28, 28, 3, 4, 5, 5, 5, 8, 10, 6, 6, 7, 9, 11, 11, 9, 8, 8, 8, 11, 9, 9, 9,
10, 11, 14, 11, 12, 12, 13, 13, 14, 17, 16, 16, 16, 16, 15, 13, 15, 13, 13, 13, 15, 14, 12, 15, 14, 17, 17, 20, 18, 19, 22, 20, 21, 20, 21, 21, 20, 21, 20, 22, 25, 23, 23, 26, 26,
27, 28, 25, 28, 28, 28, 27, 28, 31, 3, 3, 6, 4, 9, 4, 7, 8, 8, 7, 8, 8, 7, 6, 8, 7, 9, 8, 9, 9, 8, 8, 13, 10, 12, 13, 10, 11, 11, 13, 11, 11, 11, 11, 11, 13, 15, 16, 15, 17, 14, 16
, 17, 17, 20, 19, 18, 20, 20, 22, 18, 20, 19, 19, 22, 20, 24, 20, 22, 23, 22, 23, 21, 23, 24, 25, 23, 25, 24, 25, 25, 22, 26, 23, 28, 27, 26, 2, 5, 6, 5, 6, 3, 5, 7, 7, 5, 8, 8, 4,
 8, 7, 6, 9, 6, 7, 9, 11, 11, 8, 10, 11, 10, 12, 11, 11, 11, 14, 14, 12, 14, 14, 13, 13, 12, 12, 12, 14, 15, 16, 16, 19, 18, 19, 16, 18, 16, 15, 20, 19, 20, 19, 17, 19, 21, 23, 21,
 22, 22, 22, 20, 19, 20, 20, 20, 23, 22, 22, 24, 24, 27, 25, 24, 28, 29, 26, 4, 6, 4, 4, 6, 5, 4, 3, 5, 7, 5, 7, 6, 7, 5, 7, 8, 8, 11, 12, 11, 11, 9, 8, 11, 9, 12, 10, 10, 13, 12,
13, 15, 13, 15, 13, 18, 18, 16, 18, 18, 21, 21, 18, 18, 21, 20, 20, 20, 20, 18, 21, 22, 20, 21, 20, 21, 22, 22, 19, 20, 24, 25, 23, 25, 23, 26, 28, 26, 29, 31, 31, 28, 30, 31]
Mean: 16
Std:  6
```

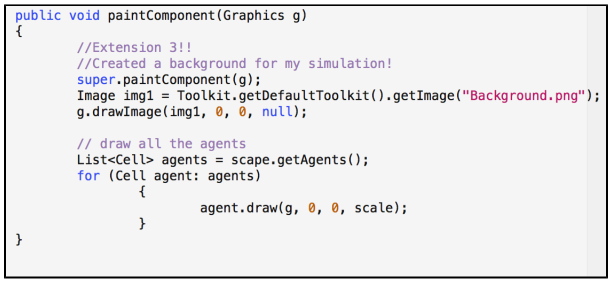
*Extension 1. *For this extension, I created a more interesting visualization of my simulation by using icon .png images. I did this last project as well. I updated my draw methods to draw images.

Extension 2. For this extension, I created a different shopping cart icon for each strategy. CP helped me implement a switch. Here is the code:



```
    //draw method using graphics
    // now using drawImage for extension 1
    //received help from cp!
    public void draw(Graphics g, int x0, int y0, int scale){
        int x = x0 + (int)(this.getX() * scale);
        int y = y0 + (int)(this.getY() * scale) - 5;
        Image img1 = Toolkit.getDefaultToolkit().getImage("Cart1.png");
        switch(this.strategy){
            //Extension 2! Different carts for different strategy!
            //Help from CP Majgaard with with SWITCH implementation
            case 1: img1 = Toolkit.getDefaultToolkit().getImage("Cart1.png");
                    break;
            case 2: img1 = Toolkit.getDefaultToolkit().getImage("Cart2.png");
                    break;
            case 3: img1 = Toolkit.getDefaultToolkit().getImage("Cart3.png");
                    break;
            default: img1 = Toolkit.getDefaultToolkit().getImage("Cart1.png");
                     break;
        }

        g.drawImage(img1, x,  y-30, null);

        return;
    }
```

Extension 3. Finally, I added a background to my Simulation making it look more interesting. The code for this is in Landscape display:



```
public void paintComponent(Graphics g)
{
        //Extension 3!!
        //Created a background for my simulation!
        super.paintComponent(g);
        Image img1 = Toolkit.getDefaultToolkit().getImage("Background.png");
        g.drawImage(img1, 0, 0, null);

        // draw all the agents
        List<Cell> agents = scape.getAgents();
        for (Cell agent: agents)
                {
                        agent.draw(g, 0, 0, scale);
                }
}
```

What I learned. I learned how to implement a Queue data structure in this project. We continued our work with Cells and Landscape to visualize a Checkout counter situation. Overall, I gained experience with DoubleLinkedLists, Cells, Landscapes, and drawing Images and learned new skills with using Queues!

Who helped me. In this project, I worked alongside my fellow classmate Steven Parrott throughout the entire project. I received help from Professor Bruce Maxwell, CP Majgaard, and Aaron Liu.