CS231 Project5

Checkout Line Simulation

Summary

The goal of this project was to create and implement queues in order to simulate grocery store checkout lanes. This was an extremely time consuming and difficult project and I am happy to have completed it successfully.

Within this simulation, we were given the task to create customer objects, each having a specific method of choosing a given checkout queue. Each customer also had a given number of items in their “basket.”

Customers used one of three strategies to pick a given checkout queue:

1) Picking a random queue

2) Picking 2 random queues and then choosing the shorter queue

3) Scanning each queue then choosing the shortest queue

These three strategies affected the time in which it would take a customer to choose a given line. The fist strategy took the shortest amount of time (1 time step), the second strategy took 2 time steps and the third strategy took 4 time steps.

Below I have outline the classes I created in order to implement this simulation.

Customer

This class is an extension of my Agent class from Project 4. The fields of the customer class included: items (int number of items in customer basket), phase (int value of -1, 0, 1, or 2 indicating which selection phase), waitTime (int value of 1, 2, o2 4) and strategy (int value of 1, 2 or 4 indicating which strategy). The customer class also included a private variable “turns” which would be responsible for counting the number of time steps it would take a customer from spawning to the completion of their checkout phase.

The following methods were implemented in this class:

- Customer
  - parameters included strategy, and (x,y) position
  - sets initial number of items to 1
  - sets initial phase to -1
  - sets wait time depending on given strategy
- updateState
  - takes ArrayList of queues as parameter
  - increments the “turns” variable
  - if in selection phase and wait time is not 0, customer remains in selection phase and wait time decreases
  - if in selection phase and wait time is 0, phase changes and customer chooses a given line to join based on their strategy
- getItems
  - returns integer number of items in basket
- setItems
  - sets number of items in basket to a random integer value between 1 and 20
- getPhase
  - returns string indicating which phase customer is in
- getIntPhase
  - returns integer value of phase
- setPhase

    - sets the phase of the customer
- getStrat
    - returns string indicating which strategy the customer is using
- numTurns
    - returns integer value of the “turns” variable
- removeItems
    - removes items from customer basket
- toString
    - returns string indicating phase and number of items
- draw
    - if customer is waiting in line, they are orange
    - if customer is choosing a line, they appear blue

Checkout

This class was an extension of the Agent class as well. The only field of this class was a MyQueue object (from lab) named checkoutQueue. This queue holds Customer objects.

The following are the class methods:

- constructor
    - parameters include (x,y) position
    - sets field variable
- getSize
    - returns size of queue
- addCustomer
    - adds customer object to queue
- removeCustomer
    - removes customer object from head of queue
- toString
    - returns string indicating number of items first customer in line has
- draw
    - draws cashier
    - draws each customer in the given queue (calling Customer draw method)
- updateState
    - if there exists a customer at the head of a given queue, remove an item from the customer’s basket
    - if there are not any customers in a given queue, do nothing
    - if the customer at the head of the list no longer has any items, remove them from the queue
        - do this by changing their phase
        - when a customer has this given phase, the draw method is not called
        - move the remaining customers in line down towards cashier

```
//updates state of Queue based on states of customers in queue
public void updateState(){
    //if customer at head is present, remove their items
    if( this.checkoutQueue.peek() != null ){
        this.checkoutQueue.peek().removeItems();
    }
    //if there arent any customers in queue, do nothing
    else if( this.getSize() == 0 ){
        return;
    }
    //if customer at head does not have any items, remove them from queue
    if( this.checkoutQueue.peek().getItems() == 0 ){
        this.removeCustomer().setPhase(2);
        int i=0;
        for( Customer cust: this.checkoutQueue ){
            // int y = cust.getY();
            i++;
            //change position of remaining customers in queue
            cust.setY( this.getY() - 20*i );
```

Spawner

This class includes two fields: a random object and an ArrayList of integers indicating the possible strategies of a customer.

Below are the methods:

- constructor
  - initializes random object and array list
  - adds 1,2, and 4 to arraylist
- makeCustomer
  - makes a single customer object
    - passes in a random (x,y) position and a random strategy chosen from strategy array list EXTENSION

Landscape

The fields in this class included an int width, int height, ArrayList of Checkout objects, and an ArrayList of customer objects.

The following methods were implemented:

- constructor
  - parameters include width and height
  - initializes ArrayLists of customer and checkout objects
- getHeight
  - returns height
- getWidth
  - returns width
- addQueue
  - adds a checkout object to the ArrayList of checkout objects
- toString
  - returns string indicating number of checkout objects are in the arraylist
- draw
  - draws header “Welcome to Hannaford” EXTENSION
  - loops through each checkout object in array list and draws each
  - loops through each customer in array list and draws each

- updateState
  - creates Spawner object
  - makes a customer using the spawner object and adds it to ArrayList of customers
  - loops through customer array list and updates state of each customer using customer update state method
  - loops through each checkout object in array list and updates state of each checkout object
- calcAvg
  - this calculates the average of an array list with integer objects
- stDev
  - this calculates the standard deviation of an array list with integer objects

Landscape Display

This code was borrowed from previous projects.

CheckoutSimulation

- creates Landscape object
- creates 10 Checkout objects and appends them to ArrayList of Landscape object
- creates LandscapeDisplay object
- iterates a given number of times, updating the state of the landscape object each time and then repainting the landscape display and saving an image of the given iteration
- creates a different array list for customers who have used different strategies
  - these array lists contain the time steps of customers from when they spawned to their deletion
  - we then calculated the standard deviation and average of each of these lists

Extensions

For my extensions, I decided to make my project visually interesting. While customers are in the selection phase, they are blue circles and when they move to a given queue they turn into orange circles. I also added "$" signs to each cash register and added text to my Landscape saying "Welcome to Hannaford" and "let's buy some groceries".

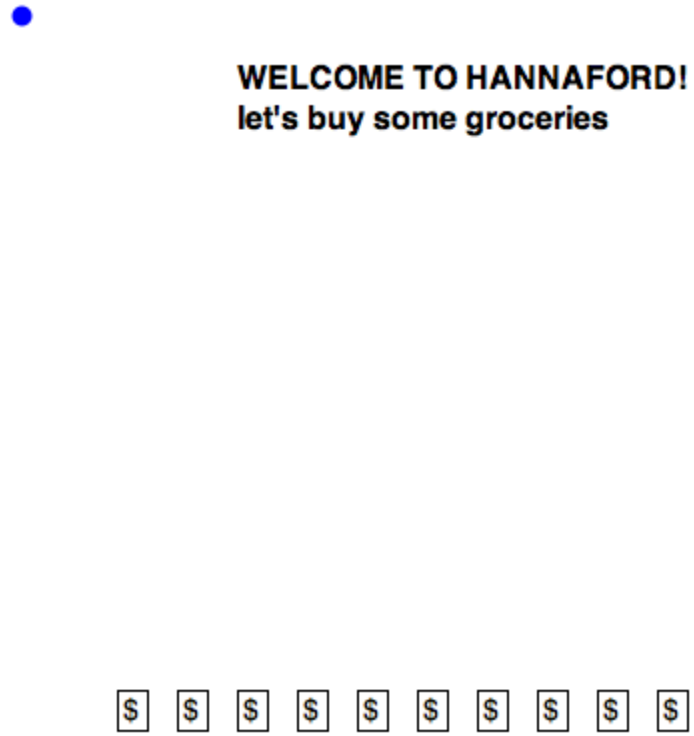
```
//draws a circle of radius 5 in x,y position
public void draw( Graphics g ){
    //if customer has finished checking out, they are not drawn
    if( this.phase == 2 ){
        return;
    }
    //if customer is waiting in line they are orange
    else if( this.phase == 0 || this.phase == 1 ){
        g.setColor(Color.orange);
    }
    //if customer is selecting, they appear blue
    else {
        g.setColor(Color.blue);
    }
    g.fillOval(this.getX(),this.getY(),10,10);
```

For a second extension, I allowed each customer to have a strategy randomly assigned to them. This is explained above. After calculating the Average Times for each of these strategies, I found that the person using strategy 4 used the least amount of time steps.

```
Average number of time steps for strategy 1: 119.81632653061224
Standard Deviation of time steps for strategy 1: 46.41413994146205
Average number of time steps for strategy 2: 121.828125
Standard Deviation of time steps for strategy 2: 46.11044983498182
Average number of time steps for strategy 4: 110.15384615384616
Standard Deviation of time steps for strategy 4: 52.008277675246674
```

GIF

Below is a GIF animation of my simulation in action:



Conclusion

This was definitely the most difficult project thus far. I feel very accomplished having finished this project and know that I learned a lot about the structure of a complete project through trial and error. I learned the power of a queue and how they have the ability to make a simulation like this much more efficient.

I would not have been able to complete this project without the help of LiWei (my tutor), Adam Carlson, Matt Martin and Stephanie!! THANK YOU!