CS 1181 Project 3: Data Science (Grocery Store Simulation)

PURPOSE: This assignment explores simulation of a real world situation which leads to meaningful actions to solve a problem. A very common use of computers is to run simulations to answer practical questions with no easy closed-form mathematical solution. A simulation is a program designed to model a real system of interest. By setting parameters and selecting an appropriate model, results are produced that model the behavior of the real system. If the model is a valid representation of the real system, then the simulation can even predict the future performance of the real system.

PROBLEM: In this project, you will construct a program to optimize efficiency of a grocery store. Your goal is to use your simulation to answer a specific question. A grocery store currently has 12 checkout lanes. Some of those lanes allow customers to purchase any number of items. Some of the lanes are 'express lanes', which only allow customers who are purchasing twelve items or less. If our goal is to minimize the overall time that customers wait in line, how many of our 12 lanes should be express lanes?

Assume that we collected data from a 'normal/average' set of customers during a normal/average day of operation. We observe the customers and determine that they are characterized by:

(1) Time of arrival at the store (in minutes relative from the store opening)
(2) Their order size (the number of items that they purchase)
(3) The average time it takes for them to select each grocery item (their total time spent browsing divided by their order size)

We provide this information in a text file. Each customer is one line of the data file, characterized by these three measurements in that order.

For example, the line "2.0   15   0.5" in the data file represents a customer that arrives at the store 2.0 minutes after the store opens. The customer picks up 15 items and on average takes 0.5 minutes to pick up each item. The customer therefore spends 15 * 0.5 = 7.5 minutes shopping and then is ready to check out at 9.5 minutes after the store opens.

The customer then moves to the checkout area. We will assume that customers will always select the checkout area with the shortest number of other customers in line (no matter how many items those customers are purchasing). Customers with more than 12 items can only use the regular checkout lanes. Customers with 12 or fewer items can use any lane. If there is a tie in line length, then customers will select any of the lines of equal length. Customers with 12 or fewer items will select an express lane over a regular lane if the two lanes are tied in the length of their line.

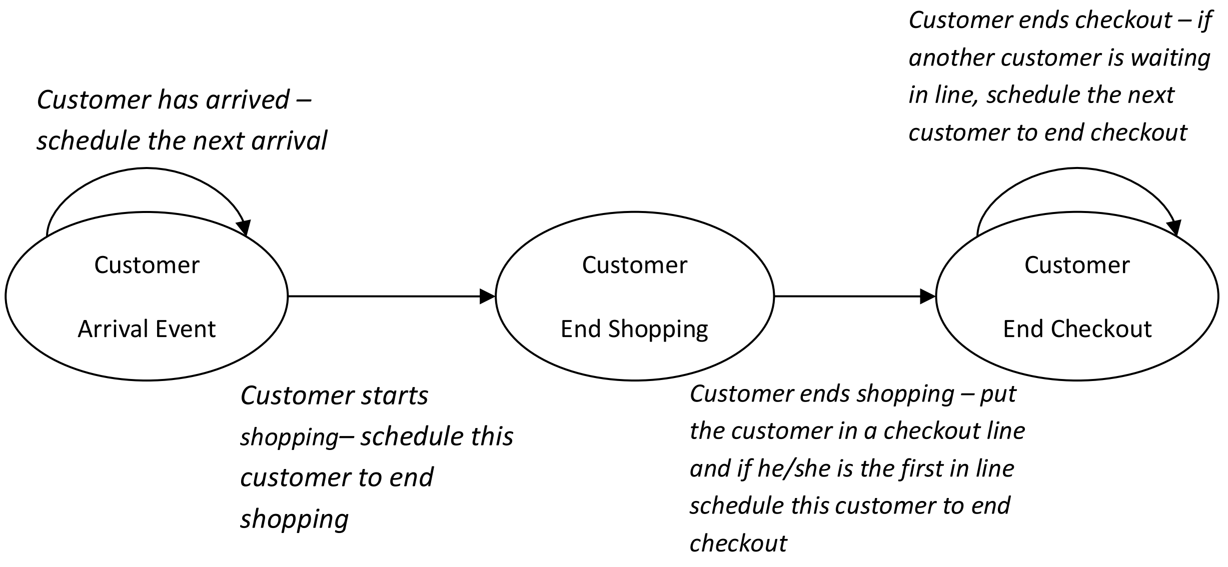
If there is no one in line for a checkout lane, then the customer can immediately begin checking out. If there is anyone in line, then the customer must wait for everyone in front of them in their line to check out before they can begin to check out.

The time to check out is determined by the characteristics of the type of lane (regular or express) and the size of the order. Regular lanes have an average checkout time of 0.05 minutes per item in the order plus 2.0 minutes to process payment and coupons. Express lanes have an average checkout time of 0.10 minutes per item in the order plus 1.0 minutes to process payment and coupons.

Continuing the example, our customer with 15 items moves to check out at 9.5 minutes after the store opens. Since they have 15 items, they must use a regular checkout lane. They select the checkout lane with the shortest line. Let us assume that there is a regular checkout lane with no line (0 wait time). They immediately begin to check out. The time to check out is 15 (items) * 0.05 (minutes per item) + 2.0 (minutes to process payment) = 2.75 minutes. The customer leaves the store at 12.25 minutes after the store opens (9.5 min arrival at checkout + 0 wait + 2.75 minute checkout). If all lanes had at least one customer already in the lane, then our example customer could not begin to check out until every customer in line in front of them had first completed _their_ checkout.

We will use a discrete event simulation to simulate the grocery store. This means time is managed by using a simulation clock that is advanced whenever an interesting event occurs. For example, if the simulation time is 0.0 minutes and a customer enters the store at a time 2.0 minutes, we mark the passage of time by advancing the simulation clock to 2.0 minutes (current time = 0.0 + time of arrival event = 2.0). If a customer reaches a checkout line 7.5 minutes later, we advance the simulation clock to 9.5 (2.0 + 7.5). Our assumption is no interesting activities occur between events (i.e. nothing significant happens in the intervals (0.0, 2.0) or (2.0, 9.5) so we can record the passage of time by moving the simulation clock forward to the time when the next important event occurs).

The diagram shown below outlines the flow of events in our simulation.


Customer has arrived – schedule the next arrival
Customer Arrival Event
Customer starts shopping– schedule this customer to end shopping
Customer End Shopping
Customer ends shopping – put the customer in a checkout line and if he/she is the first in line schedule this customer to end checkout
Customer End Checkout
Customer ends checkout – if another customer is waiting in line, schedule the next customer to end checkout

To begin, set the simulation clock to 0.0 and schedule the first customer arrival. We place each new event on a list (a priority queue might be a good choice) that contains the sequence of pending activities ordered by increasing simulation time.

The actions associated with our example customer are (1) create customer object containing the customer's data and (2) schedule the customer's arrival. The setup after scheduling the first customerArrival event might appears as follows:

simClock = 0.0
event list = (
{2.0, customer 1 arrives}
// other customers/events not shown – the event list should start with all customer arrivals!
)

To process an event, simply remove the first event from the list, update the simulation clock to the time of the event and perform the action needed to simulate the event. After removing the event the list appears as follows:

simClock = 2.0 Note simulation time has advanced to the time of the event
event list = (
// other customer/events not shown
)

Since the first event is an arrival, perform the actions associated with a new arrival. This customer is selecting 15 items and will take an average of 0.5 minutes to select each item so you will need to schedule an end shopping event for this customer in 15 * 0.5 = 7.5 minutes. Now the event list looks like this:

simClock = 2.0
event list = (

{9.5, customer 1 done shopping},
// other customer/events not shown
)

Events will be processed in scheduled 'time' order… there may be other customer arrivals or other events before our example customer 1 is done shopping. When their 'event' is next, then we will remove them from event list and assign them to a lane.

simClock = 9.5
event list = (
// other customer/events not shown… for the example we will assume that one of them is
// using our assigned lane until 10.0
)

If that lane is full, then we place that customer/event in a queue for that lane. Eventually they will get to the front of their checkout line at which point we can schedule their checkout. For this example, we will assume that they were assigned to checkout lane #3 and that the person 'in front' of them finishes _their_ checkout at time 10.0 minutes after the store opens. Now that they are at the head of the checkout line, we start their checkout by scheduling its completion.

simClock = 10.0
event list = (
{12.75, customer 1 done with checkout}
// other customer/events not shown
)

simClock = 12.75
event list = (
// other customer/events not shown
)

Once the customer completes checkout then we have finished processing that customer. We remove them from the event queue and check to see if there is anyone behind them in their checkout lane. If there is, then we'll schedule that next customer for checkout. We might want to save the customer to provide data to calculate statistics at the end of the simulation.

To summarize, the actions to perform for each event are:

customerArrival -- When an arrival occurs, create a customer, schedule the customer to end shopping in an amount of time determined by the size of the customer's order and the time to pick up each item. If there is another arrival in the arrival file, then schedule the next arrival.

customerEndShoppingPicksLane -- Pick the shortest line and place this customer in the checkout line. If the customer has 12 or fewer items, they will pick the lane (express or regular) with the shortest queue. If there is a tie for queue lengths, then we will assume that customers with 12 items or fewer items will select an express lane. Customers with more than 12 items cannot use the express lanes. If the checkout lane is not already busy then schedule the customer to end checkout at a future time determined by the size of the order and the time to check out each item. If the checkout lane is busy, put the customer in the appropriate queue for the lane.

customerEndsCheckout -- Remove the customer from the system and save any information necessary to calculate/display statistics. If there are more customers in the checkout lane waiting for that customer to finish, then schedule the next customer to end checkout at the appropriate time.

ANALYSIS

As a customer moves through the store, display the following information in a log file: the customer number, arrival time, end shopping time, end waiting time, and end checkout time. At the end of the simulation, display statistics you deem valuable to your analysis. You may want to include the total number of customers served, average queue lengths (or the average queue length as perceived by the customers), the average waiting time per line/customer, total customers passing through each queue, the maximum length of each queue, etc.

You will be given one or more "typical" arrival.txt files that characterize the store's customers. Use these files together with statistics generated by your program to recommend a "best" mix of regular and express, and 'closed' lanes. For the purpose of our analysis, assume that the goal is to decrease the time that customers spend 'waiting' in line. You have a maximum of twelve lanes – each lane can be designated as 'regular', 'express', or 'closed'. Having more lanes open costs more money to operate. How much will it cost (in terms of additional customer wait time) if I close one or more lanes?

Write a one page paper (typed) supporting your recommendation on how the lanes should be used/assigned. Make sure to cite your simulation results to support your argument! This is the essence of what computer scientists do professionally! Remember, a store owner would hire you to conduct such a study and their concern is twofold: (1) to increase customer throughput and (2) to decrease the number of unused checkout lines. Save your paper as a PDF file in the same directory that contains the arrival.txt file BEFORE you ZIP/archive your project and upload it to Pilot.

DESIGN REQUIREMENTS: For this assignment, you are encouraged to use any standard API data structures. The entire standard Java library is open to you. Good luck!

GRADING

(70 pts) **Base Functionality**

[10] There is at least one class (e.g. Customer or Event) that represents a customer's activity within the store
[10] There is a class to represent a checkout lane, and some distinction is made between regular and express lanes
[10] arrival time correctly computed for each customer
[10] finished shopping time correctly computed for each customer
[10] finished checkout time correctly computed for each customer
[10] wait time correctly computed for each customer
[10] average wait time correctly computed

(10 pts) **Simulation Analysis/Result**

[10] Position paper that argues, based on the results of your simulation, your recommendation for the 'best' number of normal and express checkout lanes.

(20 pts) **Style**

[10] Object-oriented principles such as encapsulation, inheritance, and polymorphism, are used appropriately
[5] Standard coding conventions are followed, including variable names and indentation.
[5] The coding is ***meaningfully*** commented, and there is not commented out junk code

NOTE: There is no valid excuse for turning in a program which does not compile. A program will receive no credit if it does not compile and execute.