Claire Liu

November 6, 2020

CS 150-01

Project II: Event-Driven Simulation of a Coffee Shop

1. Introduction

The purpose of this project is to decide how many cashiers (baristas) to hire at a certain coffee shop in order to optimize the profit through event-driven simulation. An event-driven simulation is where the program processes events as they come, advancing the current time only when there is an event (Weiss). The event-driven simulation is implemented through the use of data structures like ArrayList, LinkedList, ArrayDeque, Queue, and Priority Queue (Oracle, "ArrayList"; Oracle, "LinkedList," ; Oracle, "ArrayDeque"; Oracle, "Queue"; Oracle, "Priority Queue"). The LinkedList and the ArrayDeque (a circular array) are implemented through the Queue interface and their performances are compared. The data from the coffee shop to be analyzed are taken and read from a given input file.

A few assumptions about the coffee shop and the customers were made. First, it was given that the shop opens at 6am and closes at 9pm (Xia). At 9pm, the customers cannot enter the store, but will be served if they are already in the store. Also, the profit from serving a customer is randomly generated numbers given the bounds in the input file, and is not a few set amounts that would be typical on a real menu. Additionally, when a customer arrives at the shop, if the length of the queue is 8 x number of baristas or more, then they leave. Lastly, it was assumed that when called, it took zero seconds for the customer to go from the line to the counter. Though this time could technically be factored into the serving time, it was not explicitly stated that the serving time would include the time it takes for the customer to come from the queue to the counter to be served.

2. Approach

This simulation is implemented through three classes. First, the Event class creates events that were either labelled as ARRIVAL or DEPARTURE. The event object also stores the customer number and the time of the event (either arrival or departure) The event objects are comparable by the time of the event so that the events can be used in a priority queue.

Second, the Customer class created customer objects that stored the customer number, the arrival time of the customer, the departure time of the customer, how long the customer waited to be served (in the queue and at the counter), and the profit generated from the customer.

The main class of the program was the Simulation class. This is where the event-driven simulation occurred. The user specifies the number of baristas as one of the instance variables of this class and also specifies whether a LinkedList or an ArrayDeque is to be used as the customerLine Queue, which is the queue where the customers wait in the store. A queue was chosen because the customers are to be served on a first-in-first-out basis. When the program runs, the first thing that happens is the input file, "input.txt," is read using the Scanner and File classes (Oracle, "Scanner"; Oracle, "File"). The parameters of profit limits, time limits, and barista cost are filled from the input file Also, each arrival time read from the input file and converted to an instance of the LocalTimeClass (Oracle, "LocalTime"). For each arrival time, a new Event is created with the arrival time and inserted to the eventSet PriorityQueue. A PriorityQueue was chosen because then the events would be processed in order of time. Additionally, for each arrival time, a new Customer is created and inserted into the customerRecords ArrayList. A Customer's instance variables are updated as the customer is served. Therefore, at the end of the program's run, the customerRecords is an accurate record of each customer's visit.

After the input file is read, the Events in the PriorityQueue are processed. For every arrival event, if there is room in the queue then they join the customerLine queue, otherwise, they immediately depart, the corresponding Customer in customerRecords is updated accordingly, and the totalNumberOverflowed counter increments by one. If they were able to join the queue, then the Customer was given a randomly generated profit and counter serving time. The profits are randomly generated using ThreadLocalRandom (Oracle, "ThreadLocalRandom"). People are served in a first-in-first-out fashion from the customerLine. If there was an available barista, then they were served immediately, so their departure time was simply their arrival time plus their counter serving time, and their total wait time was their arrival time minus their departure time, as calculated using Duration (Oracle, "Duration"). However, if they had to wait in the customerLine Queue, then they were served once a departure event was processed and a barista became available. Thus, their departure time was the departure time of

the last Customer plus their counter serving time, and their total wait time was their arrival time minus their departure time.

Finally, the specified statistics (the amount of daily net profit, the percentage of overflow, and the average and maximum waiting time of all customers served) are computed and printed to the terminal window. The daily profit was calculated by subtracting the cost of a barista multiplied by the number of baristas from the totalProfit (a running total of profits from each customer). The percentage of overflow was computed by dividing the totalNumberOverflowed by the customerNumber and multiplying by 100. The max time a customer had to wait was found by iterating through the customerRecords and the average time a customer had to wait was found by iterating through the customerRecords, keeping a running total of time waited, and dividing by the customerNumber.

All classes were unit tested (JUnit Team). The dependency map is shown in *Figure 1*.

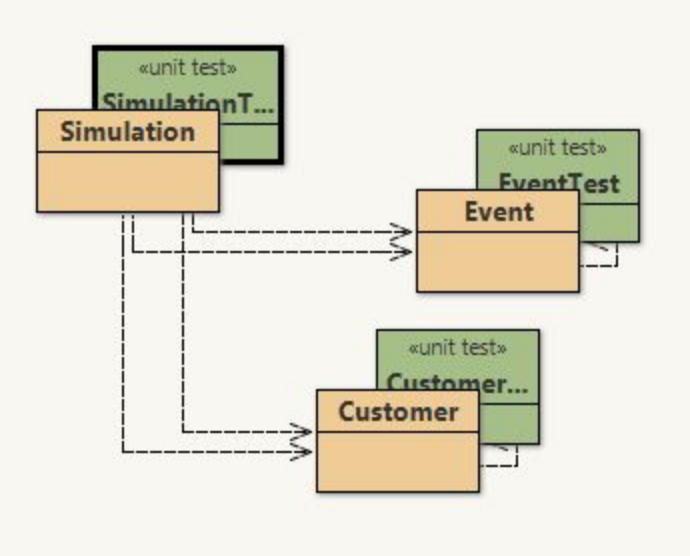

Figure 1: The design architecture of the program

3. Methods

The simulation is run by running the main method in the Simulation class. The specified results of the amount of daily net profit, the rate of overflow, the average waiting time, the maximum waiting time, and the time it took the program to run are outputted to the terminal window. Printing to the terminal window was chosen over writing to a file so that the user can run the program multiple times and not have to keep reopening a file.

Based on the project description, the only parameter that could be changed by the user was the number of baristas. Data was collected for one barista to nine baristas. This range was chosen because in this range, it was clear how many baristas produced the maximum amount of profit.

For each number of baristas (1, 2, 3, … 9), the experiment was run five times in order to ensure consistent results as the profit from each customer and the waiting time of each customer were randomly generated. The data from each of the five runs for each of the number of baristas was collected, averaged, and graphed

Additionally, half of the runs were run with LinkedList as the queue and half of the runs were run with ArrayDeque as the queue. Since the input size was fixed, it was fine to switch between the two. Also, switching between the two was of no consequence to the functionality of the program because they were both being implemented and used as Queues.

4. Data and Analysis

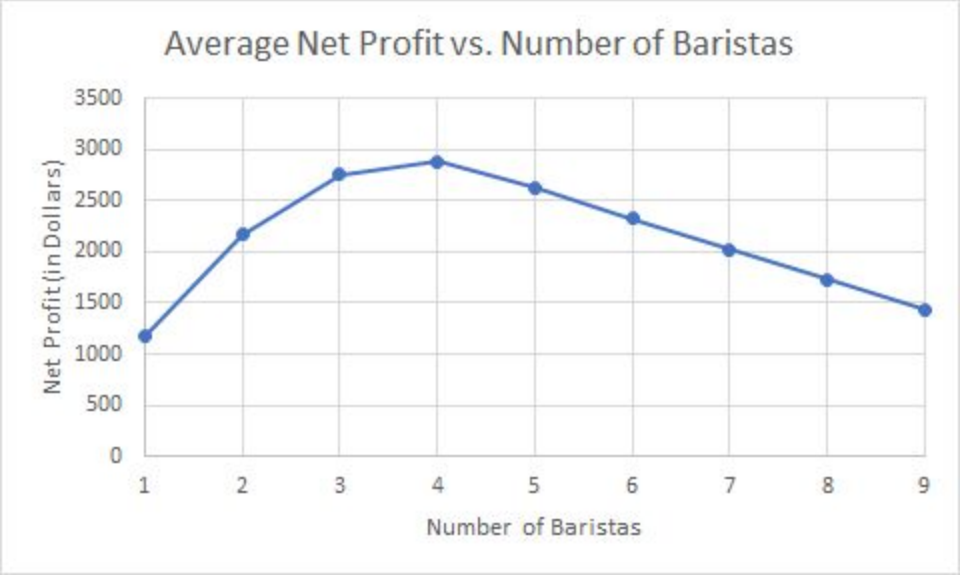

Figure 2: A graph showing the net profit given by staffing x number of baristas.

Based on *Figure 2*, staffing 4 baristas generates the most profit. From 1 to 4 baristas, the profit rises, but once the number of baristas exceeds 4, the coffee shop is overstaffed and there is not enough work for all the baristas. However, the baristas must still be paid, so when the coffee shop is overstaffed, the net profit starts to decrease.

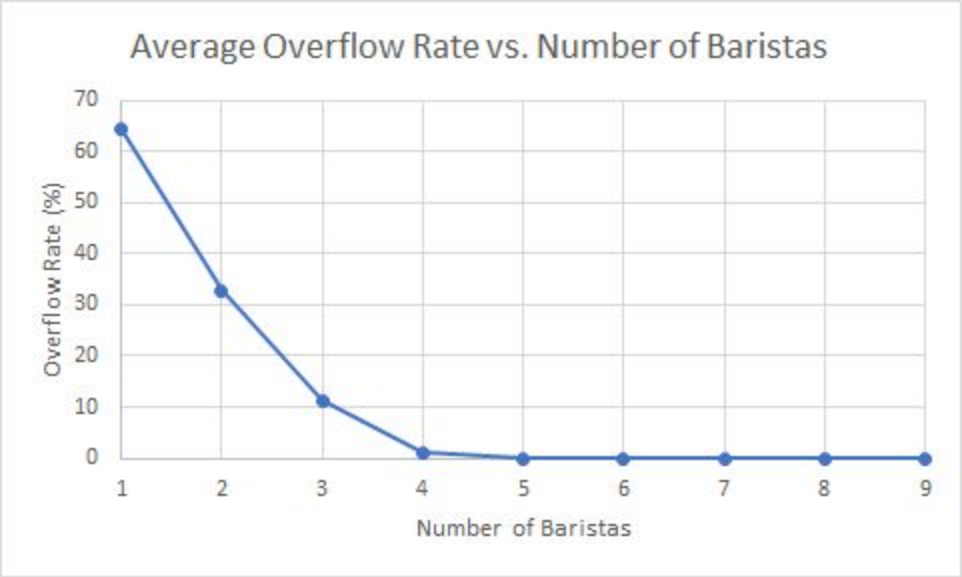

Figure 3: The percentage of customers overflowed/leave because the line is too long given x number of baristas

Figure 3 corroborates the conclusion that staffing 4 baristas maximizes profit. Having the fourth barista only increases the average net profit by about $100 dollars (Figure 2), so it may not necessarily seem worth it to staff the extra baristas. However, customers getting turned away may discourage them from coming to the coffee shop in the future, so it is critical that the number of customers getting turned away is minimized so that profit in the future can be maximized. When 4 baristas are staffed, on average, only 1% of the customers are overflowed, which is definitely an improvement from the 11% overflowed when there are only 3 baristas. Though the overflow rate decreases to 0% when five baristas are staffed, the coffee shop starts to see a decrease in profit from being overstaffed.

Figure 4: The average wait times (in blue) and the longest time a customer waits (in orange) given x number of baristas

According to *Figure 4,* the average time a customer waits in line and at the counter is highest when there are two baristas, starts to decrease after that, and plateaus around 5 baristas. This makes sense because once there are five baristas, the coffee shop starts to be overstaffed, so it would make sense that the customers would never really have to wait in line and all their wait time is at the counter. *Figure 4* also supports the conclusion that four baristas is the best because customers would on average only wait 400 seconds (about 6.5 minutes) which is reasonable. Though no wait time in line is optimal for the customer, it is not optimal for the coffee shop owner because the coffee shop would be overstaffed, resulting in a loss of profit.

Additionally *Figure 4* reveals the maximum time a customer had to wait. The maximum time slowly decreased from 1 to 4, sharply decreased from 4 to 5 and continued to decrease steadily after that. When there were 4 baristas, the longest wait time was around 1433 seconds (about 24 minutes). Though this is longer than the longest wait time at 5 baristas, 24 minutes at peak rush time is not unreasonable and considering the other variables of profit, overflow and average wait time, employing 4 baristas still seems to be the best option.

	LinkedList	ArrayDeque
Average Program Runtime (in ms)	4.84	2.69

Table 1: The program's average runtimes when LinkedList is used and when ArrayDeque is used

According to *Table 1*, in practice the ArrayDeque implementation of Queue is slightly faster than the LinkedList implementation of Queue. This is likely because in theory, the add and remove method is more efficient in an ArrayDeque than a LinkedList since there is no need for assigning the previous and next node addresses in an ArrayDeque. Additionally, the LinkedList takes up more memory in the program because in the java implementation it is a doubly linked list, so each node needs two additional spaces to store the addresses of the previous and next nodes where as an ArrayDeque is a circular array and does not need extra space to store the previous and next indices.

5. Conclusion

Overall, employing four baristas will be the best for the coffee shop because that is when the profit is maximized, which is the top priority for the client (the coffee shop owner). The overflow rate is very low at four baristas as well, which is good for retaining customers to make future profit. Despite the fact that the average wait time and the maximum wait time is not the lowest for four baristas and it might seem that five baristas or more may be better from the customer's point of view, from the client's point of view, it is best if the coffee shop is not overstaffed and profit is maximized, so four baristas is best.

Additionally, it is important to note that the ArrayDeque had better performance than LinkedList when implemented and used as a Queue.

Despite being a relatively simple event-driven simulation, the simulation still produced useful information that could help the client make an informed decision on how many baristas to employ. This model could be made better by limiting the profit from a customer to correspond to exact menu prices and collecting data on what are the most popular menu items then adjusting the random generator to favor the profits from those menu items. Additionally, certain menu items may take more time to prepare than other menu items, and if the serving time was chosen based on the item the customer orders, then the model would be more accurate. Finally, the period from 6 am to 9 pm constitutes about 3 shifts, and another model could consider staffing

different numbers of baristas at different times of the day. For example, modelling how staffing five baristas in the morning for the morning rush and four in the afternoon when business slows a little affects the profit. Incorporating this element would further improve the simulation to match reality.

Overall, event-driven simulations show how computer science principles can be applied to modelling the real world, helping people to optimize business models and more.

6. References

JUnit Team. "Packages." *Junit4 Javadoc 4.8,* 2020. https://junit.org/junit4/javadoc/4.8/

Oracle. "Class ArrayDeque<E>." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html

Oracle. "Class ArrayList<E>." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Oracle. "Class Duration." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html

Oracle. "Class File." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/io/File.html

Oracle. "Class LinkedList<E>." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

Oracle. "Class LocalTime." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html

Oracle. "Class PriorityQueue<E>." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

Oracle. "Class Scanner." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Oracle. "Class ThreadLocalRandom." *Java Platform, Standard Edition 8 API Specification*, 2020. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadLocalRandom.html

Weiss, Mark Allen. *Data Structures & Problem Solving Using Java*. Pearson Education, Inc, 2010.

Xia, Ge. *CS 150: Project II.* Lafayette College CS Dept., 2020.