

# CS 150 - Project 2 Report

Yazdan Basir

April 20th, 2020

## 1 Introduction

The purpose of this project was to find the ideal number of cashiers  $s$  needed in a certain shop in order to maximize the net profit made. This project involved the use of a simulation and randomized input data (based on bounds from the input file) to get the metrics that would be useful in concluding a definitive answer. The scope of this project is an extremely useful one given the potential real-world applications and decision-making conclusions of a such a simulation for businesses and services around the world.

The underlying assumption made in project was that the given bounds and data in the input file accurately reflect the range of profit, serving time, and arrival time of possible customers in real-world shops like ours.

## 2 Approach

The project required the use of several different data structures due to their different characteristics and requirements of the project. These included Linked Lists, Array Lists, Arrays, and most importantly, a Priority Queue.

An array of Cashier objects was used in the Cashiers class to track the cashiers available and manage their operations whilst serving a customer. An array was used here because of its fixed size. Using a dynamic data structure

would've meant the indexes of cashiers would've been moved around every time a customer was assigned to them or removed. Hence, an array seemed to be the best data structure to avoid unnecessary complications. An Array List was used to temporarily hold all the newly created Customer objects in the CustomerQueue class after reading the data from the file. A Linked List was then used to dynamically store all the upcoming Cashier free up times (the time when they would finish serving their current customer) in order to manage the customers waiting/waiting events. The Priority Queue was used to hold all the Events (Serve, Waiting, Arrival, Overflow) and each Event was handled one at a time by looking at the top of the Priority Queue.

### 3 Methods

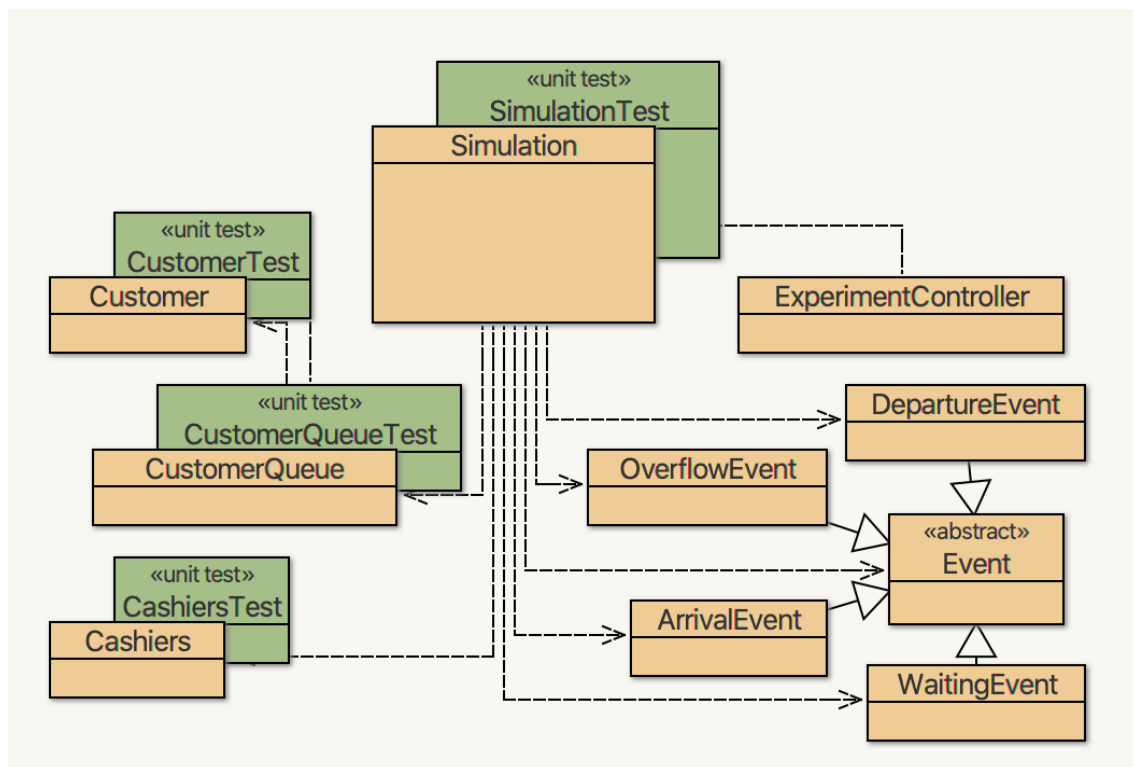


Figure 1: Structure and organization of the classes

The Cashiers class created an array called Tracker to keep track of the cashiers

and whether they were available or not to serve a customer. A method was added to this class to start serving a customer when called upon. This set the status of the first free cashier in the array to false (meaning they were not available anymore and were busy serving a customer. This method randomly generated the time the assigned customer needed to be served and returned the finishTime (the free up time) for that cashier in order to be added to the Linked List. Another method was present to finish serving a customer. This set the status of the cashier marked false back to true to indicate they were free again. The profit made from that customer was randomly generated here and added to a totalProfit variable. The CustomerQueue class created an Array List customersList to hold the Customer objects being created and added after data from the file was read. It contained the relevant methods to return the Customer object at the top of the queue and a method to remove that Customer object once its Event instance had been added to the Priority Queue. A Customer class was also present to create each individual instance of a Customer. This class took in the Customer object's arrivalTime, servingTime, and startOfServingTime.

The entire simulation was placed and operated inside of the Simulation class. A few prominent methods were present in here: the first was run() itself to run the simulation and return the data needed. The next two methods were adapter methods between Simulation and Cashiers to start serving and finish serving customers. A method was also present to return the earliest time any cashier would be free to serve again and 4 additional methods were present to add each type of Event (Serve, Waiting, Arrival, and Overflow) to the Priority Queue based on the simulation's needs.

The simulation in the run() method did not consider any customers arriving after the time 54000 seconds. This represented the 540000 seconds between 6am and 9pm and was used as the cutoff time indicating the shop had closed.

The Event instances still inside the Priority Queue were still served as customers left in a real store after closing would be served.

Important milestones of the day inside the simulation - such as a customer joining the queue, a customer being turned away, a customer heading to the desk to be served, and a customer finally getting served etc. - were all written into an output log file. This is the Simulation Log file in the project folder. This was included to give a sense of how the day was going at the shop to add a dynamic of realism to the project. This also made the debugging process easier by showing if the proper loops and conditional statements were being entered. The simulation has been designed as such to allow the Simulation Log file to get updated each time the run method is executed.

Furthermore, the Experiment Controller class held methods to create 10 Simulation instances with a different number of cashiers (from 1 to 10 inclusive). These methods executed the run() method from Simulation 100 times and averaged the data to get rid of the fluctuations associated with randomized input, even though bounds were present to restrict that fluctuation. This is where the data for the graphs was collected from.

## **4 Data Analysis**

The simulations were run and the data was averaged at the end to give an output set that could be used to derive a definitive answer to the cashier question. The data from running the simulation once per  $s$  number of cashiers was pretty much the same as running the simulation 100 times and averaging the data. While this meant that the upper and lower bounds for the randomly generated data was giving realistic/consistent numbers for the simulation to work with, the need for averaging the data was still there to get a final metric that could be used against other  $s$  number of cashiers.

This required three graphs for the most accurate conclusions:

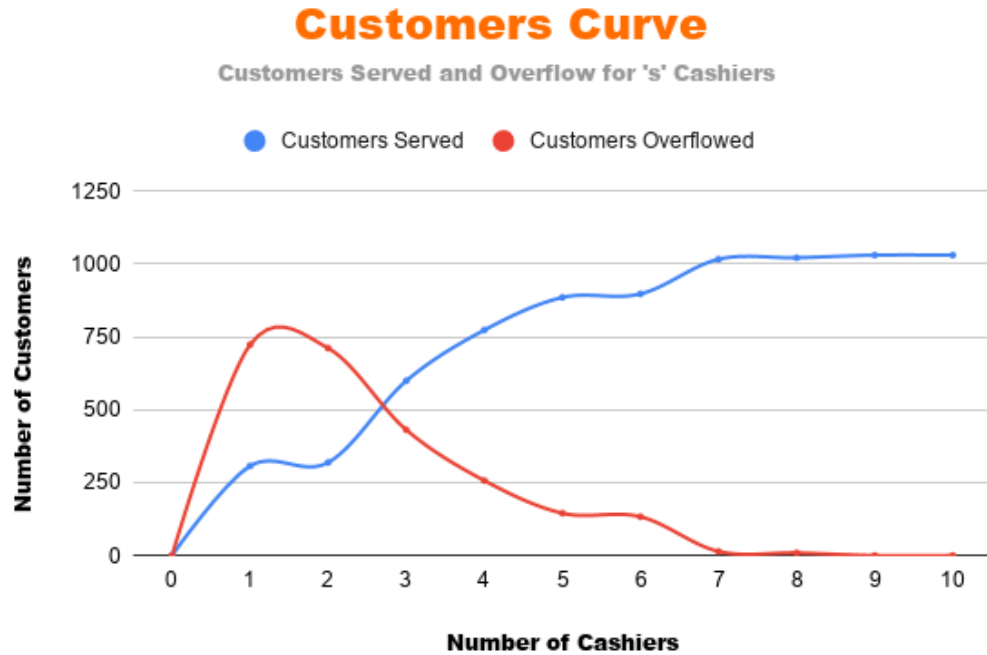


Figure 2: Graph for Customers Served and Overflowed as  $s$  increased

This graph shows that the number of customers served naturally increases as the number of cashiers available to serve them increases. This value essentially peaks at 7 cashiers and remains the same for additional cashiers as it only takes 7 cashiers to successfully serve almost every customer who enters the shop before closing time. The number of customers overflowed peaks at cashier 1, which follows logically, and decreases for every additional cashier. This value reaches near 0 at 7 cashiers because that is the value where almost all customers get served successfully.

Every additional cashier from this point onward resulted in all customers getting served successfully and none being lost due to overflow.

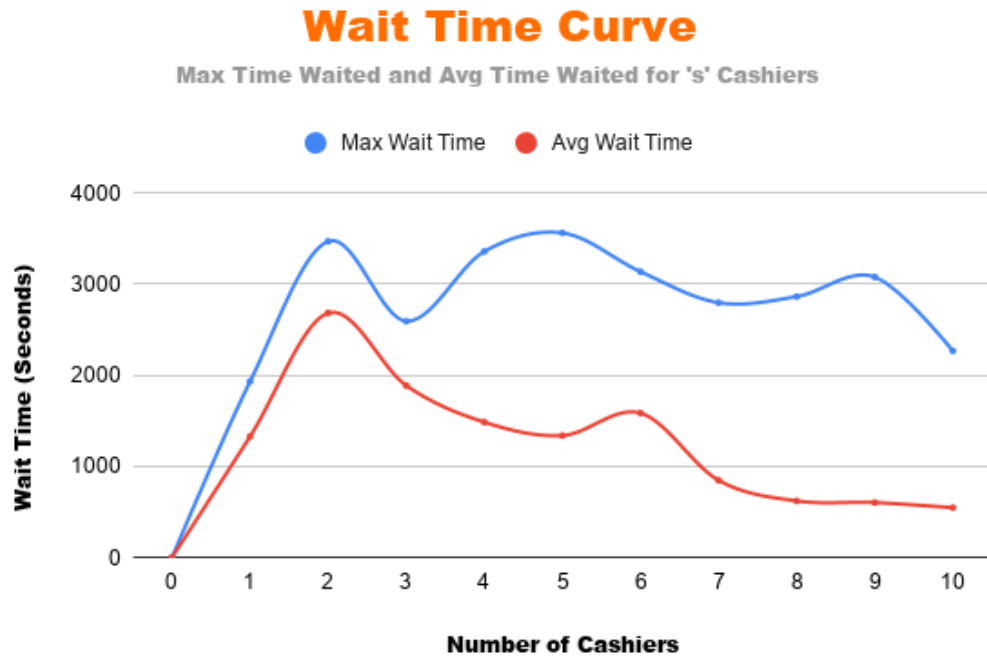


Figure 3: Graph for the Maximum and Average Wait Time as  $s$  increased

This graph shows the maximum and average waiting time for the customers as the number of cashiers  $s$  increased. The maximum wait time isn't very useful as it is the data of one customer from a set of 1054. However, we can see that the average peaks at 2 cashiers and sharply declines from that point onward. Both maximum and average wait time approach 0 as  $s$  increases. The value of 7 cashiers from the previous graph also stands out here. Since this is the first value of  $s$  where almost all customers are served successfully, all customers have to wait the minimum amount too. While the average wait time at 7 isn't absolutely 0, it's near the flat part/horizontal asymptote of the curve, which the curve approaches as  $s$  increases, just enough to make a case for it being the ideal value.

The most important graph of the bunch is the profit curve, of course. As the goal of the project was to find the ideal number of cashiers at which the net profit per day was maximized, this graph offers the required insight into

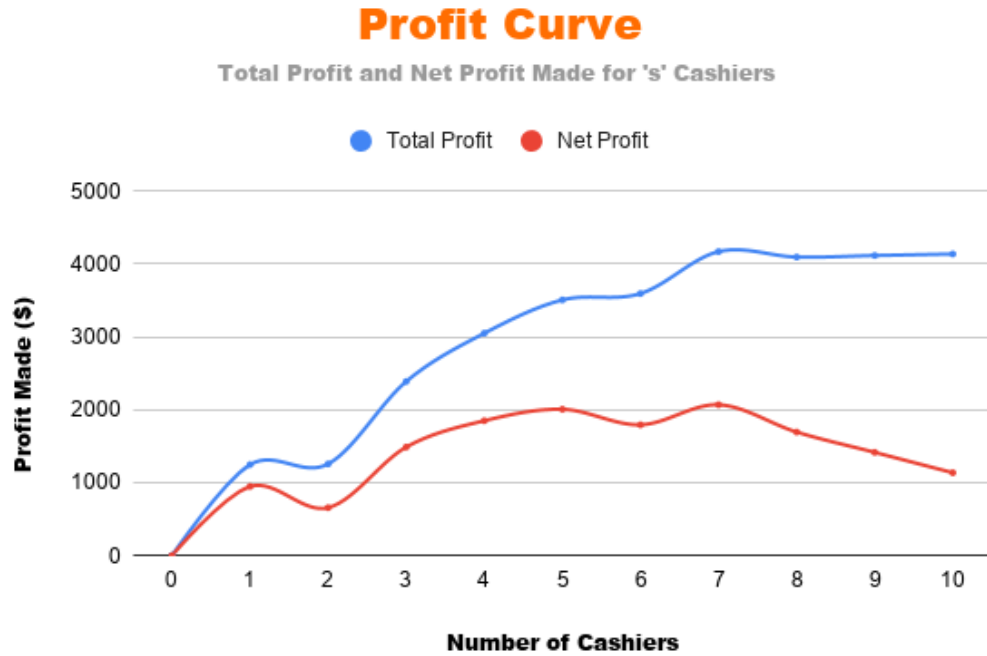


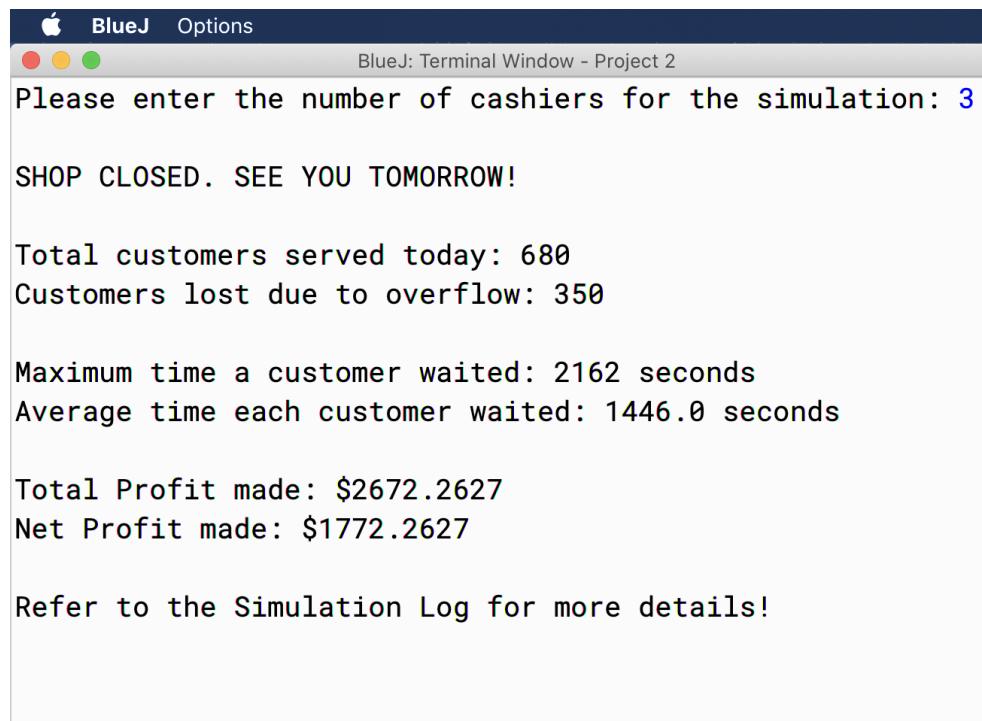
Figure 4: Graph for the Total and Net Profit made as s increased

concluding that answer. From the data above, we can see the total profit made peaks at 7 cashiers and remains the same for all cashiers from that point onward. This data is consistent with the Customers Curve and Wait Time Curve because it was at 7 cashiers where almost all customers were successfully being served and 0 were lost due to overflow. It makes sense to have the maximum profit at this point.

However, the total profit metric does not factor in the cost of hiring the cashiers in the first place. In this experiment, the cost of hiring each cashier for the day was 300 dollars. Subtracting the total cost from the total profit made gave us the real metric - the net profit made per day. In this case, the net profit made peaks at the same value of 7 cashiers. Since almost all customers are successfully served when there are 7 cashiers present in the shop and the numbers are near 0, the profit made is just enough to not decline after factoring the cost of hiring.

The fact that both the net profit curve and total profit peaks at 7 cashiers can be attributed to the fact that, despite randomized input data, average calculations were made and they proved useful in concluding a result from the data. This is exactly why running the simulations 100 times for each and averaging the data takes away all the fluctuations and inconsistencies to provide us with one final, definitive, mathematical solution to this project. Hence, from the data above, it seems that hiring 7 cashiers per day is the ideal choice for this shop.

## 5 Output



```
BlueJ Options
BlueJ: Terminal Window - Project 2
Please enter the number of cashiers for the simulation: 3

SHOP CLOSED. SEE YOU TOMORROW!

Total customers served today: 680
Customers lost due to overflow: 350

Maximum time a customer waited: 2162 seconds
Average time each customer waited: 1446.0 seconds

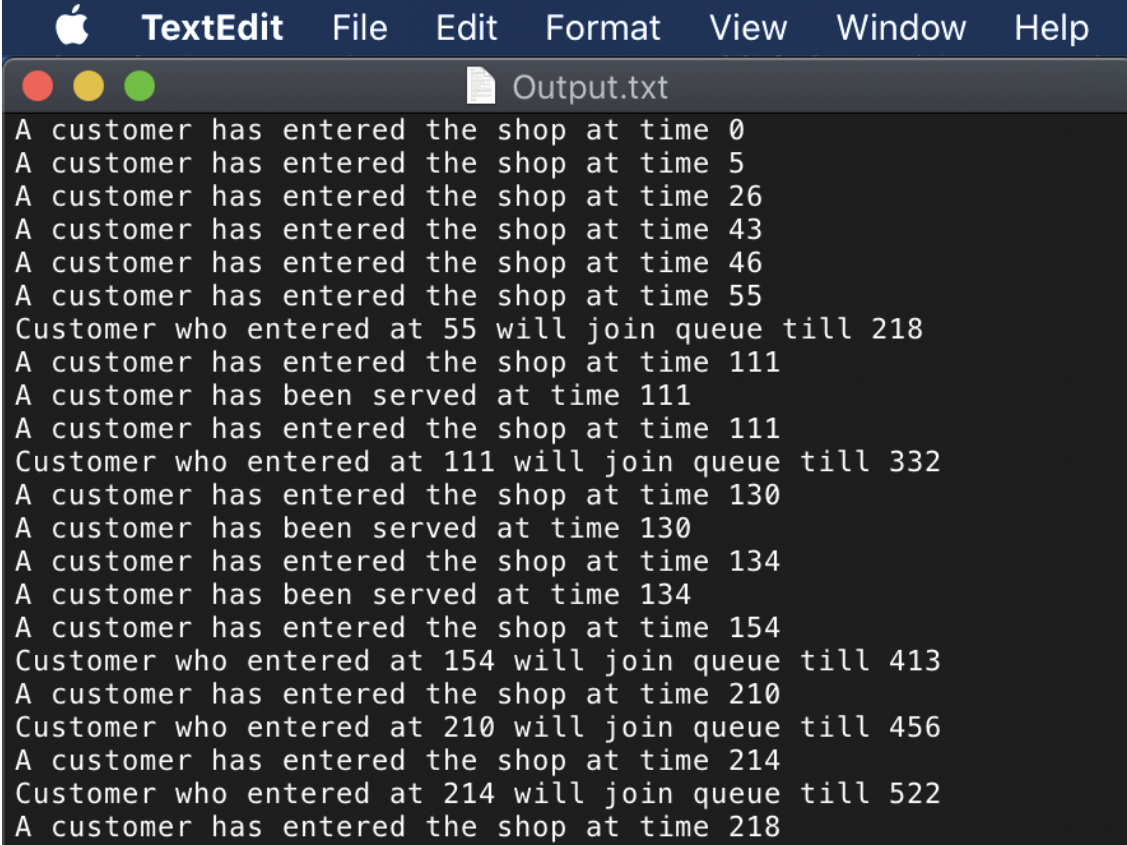
Total Profit made: $2672.2627
Net Profit made: $1772.2627

Refer to the Simulation Log for more details!
```

Figure 5: Graph for what the run() method returns on each individual run

This is the data that can be expected once the main() method in ExperimentController is called and the number of cashiers has been entered.





```
TextEdit  File  Edit  Format  View  Window  Help
Output.txt
A customer has entered the shop at time 0
A customer has entered the shop at time 5
A customer has entered the shop at time 26
A customer has entered the shop at time 43
A customer has entered the shop at time 46
A customer has entered the shop at time 55
Customer who entered at 55 will join queue till 218
A customer has entered the shop at time 111
A customer has been served at time 111
A customer has entered the shop at time 111
Customer who entered at 111 will join queue till 332
A customer has entered the shop at time 130
A customer has been served at time 130
A customer has entered the shop at time 134
A customer has been served at time 134
A customer has entered the shop at time 154
Customer who entered at 154 will join queue till 413
A customer has entered the shop at time 210
Customer who entered at 210 will join queue till 456
A customer has entered the shop at time 214
Customer who entered at 214 will join queue till 522
A customer has entered the shop at time 218
```

Figure 6: Screenshot of the Simulation Log created from the run() method

```
BlueJ: Terminal Window - Project 2

Average Data for 1 Cashier:
Average number of customers served per day: 307
Average maximum time a customer waited per day: 1932.0
Average time each customer waited per day: 1328.0
Average number of customers lost due to overflow per day: 723
Average total profit made per day: 1247.166
Average net profit made per day: 947.1666

Average Data for 2 Cashiers:
Average number of customers served per day: 319
Average maximum time a customer waited per day: 3470.0
Average time each customer waited per day: 2683.0
Average number of customers lost due to overflow per day: 711
Average total profit made per day: 1256.1151
Average net profit made per day: 656.11334

Average Data for 3 Cashiers:
Average number of customers served per day: 599
Average maximum time a customer waited per day: 2593.0
Average time each customer waited per day: 1887.0
Average number of customers lost due to overflow per day: 431
Average total profit made per day: 2387.551
Average net profit made per day: 1487.5537

Average Data for 4 Cashiers:
Average number of customers served per day: 773
Average maximum time a customer waited per day: 3358.0
Average time each customer waited per day: 1485.0
```

Figure 7: Output from running the simulations 100 times

Average Data for 4 Cashiers:

Average number of customers served per day: 773

Average maximum time a customer waited per day: 3358.0

Average time each customer waited per day: 1485.0

Average number of customers lost due to overflow per day: 257

Average total profit made per day: 3048.7822

Average net profit made per day: 1848.7827

Average Data for 5 Cashiers:

Average number of customers served per day: 885

Average maximum time a customer waited per day: 3562.0

Average time each customer waited per day: 1338.0

Average number of customers lost due to overflow per day: 145

Average total profit made per day: 3505.9382

Average net profit made per day: 2005.9385

Average Data for 6 Cashiers:

Average number of customers served per day: 897

Average maximum time a customer waited per day: 3136.0

Average time each customer waited per day: 1585.0

Average number of customers lost due to overflow per day: 133

Average total profit made per day: 3592.9148

Average net profit made per day: 1792.9163

Average Data for 7 Cashiers:

Average number of customers served per day: 1016

Average maximum time a customer waited per day: 2796.0

Average time each customer waited per day: 847.0

Average number of customers lost due to overflow per day: 14

Average total profit made per day: 4168.5938

Average net profit made per day: 2068.5938

Figure 8: Output from running the simulations 100 times



Average Data for 8 Cashiers:

Average number of customers served per day: 1021

Average maximum time a customer waited per day: 2863.0

Average time each customer waited per day: 621.0

Average number of customers lost due to overflow per day: 9

Average total profit made per day: 4093.4663

Average net profit made per day: 1693.4631

Average Data for 9 Cashiers:

Average number of customers served per day: 1030

Average maximum time a customer waited per day: 3078.0

Average time each customer waited per day: 663.0

Average number of customers lost due to overflow per day: 0

Average total profit made per day: 4114.9067

Average net profit made per day: 1414.9064

Average Data for 10 Cashiers:

Average number of customers served per day: 1030

Average maximum time a customer waited per day: 2667.0

Average time each customer waited per day: 547.0

Average number of customers lost due to overflow per day: 0

Average total profit made per day: 4135.51

Average net profit made per day: 1135.5156

Figure 9: Output from running the simulations 100 times

## 6 Unit Testing

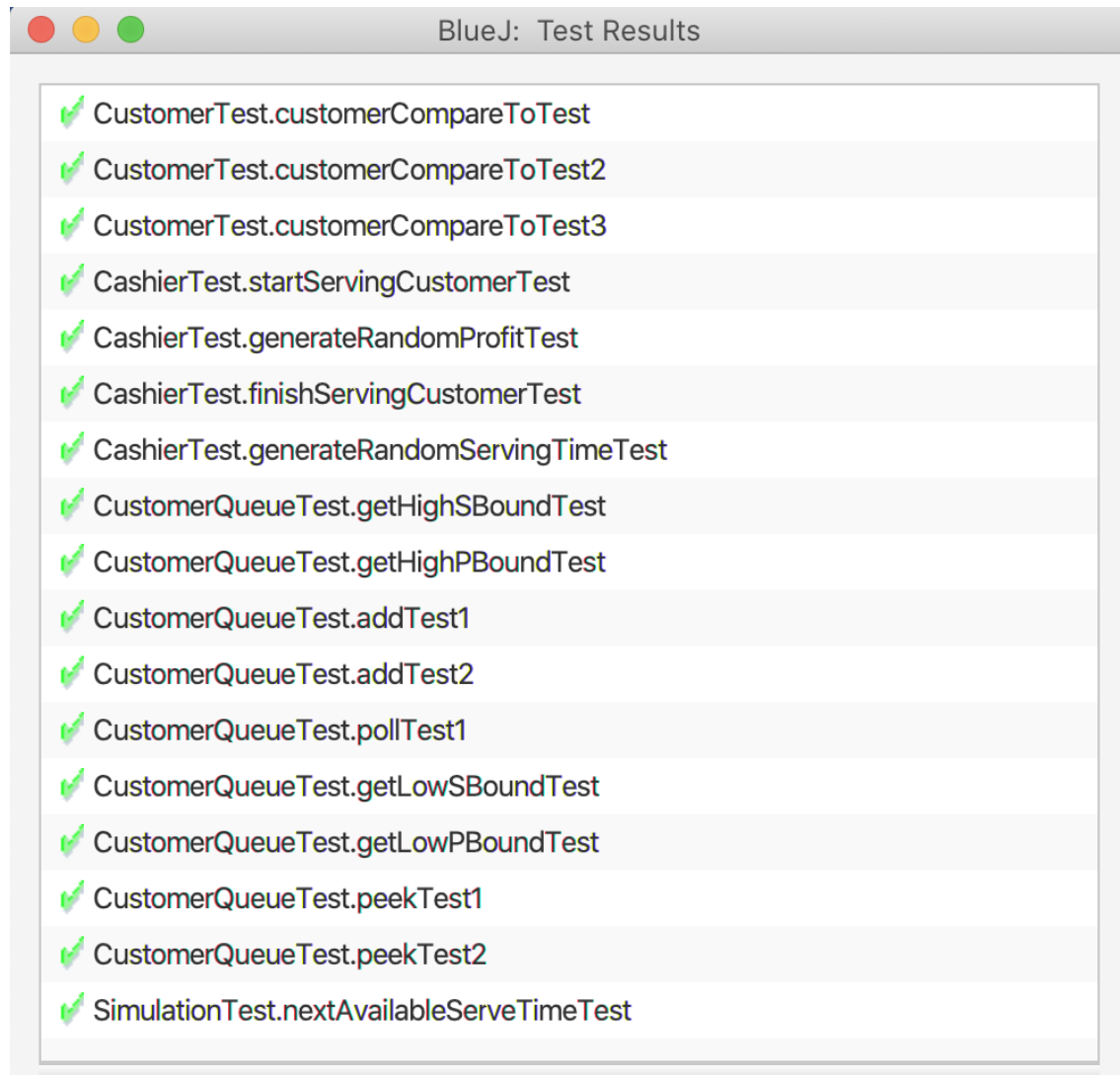


Figure 10: Output of the Unit Tests

Unit Tests were conducted on the random profit generation and serving time generation methods as well. These tests were conducted keeping their bounds from the given input file in mind. If the input file is changed and the program is tested on another set of customers, those Unit Tests will inevitably fail.

## 7 Troubleshooting

Initially there were some problems when dealing with the customers left being served when the shop closed. In the initial design, these customers were simply ignored until the main loop had been created. At the end of the simulation, the sum of the customers served and customers lost to overflow didn't match the possible customers that could be served - 1030 in case. Although there were 1054 possible customers in the file, 24 of them had arrival times after the closing time of the shop so they could not be considered.

Once the Event-Driven Simulation model had been implemented successfully, it became extremely simple to just ignore the customers who were arriving at after 9 pm/54000 seconds and treat them as customers turned away but not overflowed - customers who never entered the shop essentially.

Another problem that came up was when two Events in the Priority Queue had the same eventTime and caused issues when a customer arriving was prioritized before a customer who was finished being served at the same time. This caused an issue akin to a real-life customer cutting the line, ignoring the queue, and getting served. To fix this issue, the customers who had been served and needed to leave were assigned an eventType of 0, customers at the top of the waiting line were assigned an eventType of 1, and customers who had just entered the store were assigned an eventType of 2. This established a hierarchy for the management of customers. The compareTo() method was updated to break ties, if arrivalTime was the same, by prioritizing Departure Events first, Waiting Events second, and Arrival Events last.

## 8 Conclusion

The project, in interesting ways, tested various aspects of data structures in real-world scenarios and offered a taste into designing algorithms and simulations to answer real-world economic/business dilemmas.

The data that was received from the simulations and the conclusions reached from the graphs showed us that 7 cashiers were needed (at the minimum) to serve almost all customers successfully (9 were needed for the numbers to reach 0 but at an extra cost of 600 dollars) while having them wait the least and 7 cashiers were actually needed to maximize the profit being made every day. This can be attributed to the fact that hiring 7 cashiers provides the perfect balance between the costs and total profit made each day to give the maximum net profit. Even though not all customers were being served with 7 cashiers in the shop (around 10 were lost due to overflow), it can be said - from the output data in Experiment Controller - that almost all customers were being served.

All in all, it does seem that hiring 7 cashiers ( $s = 7$ ) in the shop for any given day is the best way to take the highest possible amount of profit home.

## 9 References

1 - Chapter 13.32: Data Structures and Problem Solving Using Java (4th Ed)

2 - Priority Queue: Java API

<https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html>