

# Project 5: Checkout Lines

[Abstract](#)

[Core Data Structure](#)

[Results](#)

[Acknowledgement](#)

## Abstract

In this week's project, I tried to simulate shoppers picking a check-out line at a large store. I modeled three types of customers by implementing three line-choosing strategies, 1. choosing randomly, 2. choosing the shortest, and 3. randomly choosing 2 and then choosing the shortest. I used a simulation to compare the performance of these three strategies. To store the customers at each checkout agent, I implemented a data structure called `Queue`, a First-In-First-Out data structure that resembles to the real queue at counters of a shopping mall. It turns out that the third strategy is the most time-efficient for customers to queue, when running the simulation with five checkout agents.

## Core Data Structure

The `Queue` data structure I used in my project are implemented by Node-based Double Linked List, where every item in the `Queue` is a Node that points to its previous and next node. There are three core methods of `Queue`,

1. `offer(T item)` This method adds the item to the end of the queue.
2. `poll()` This method removes the first item and return the first item.
3. `peek()` This method returns the first item without removing it from the queue.

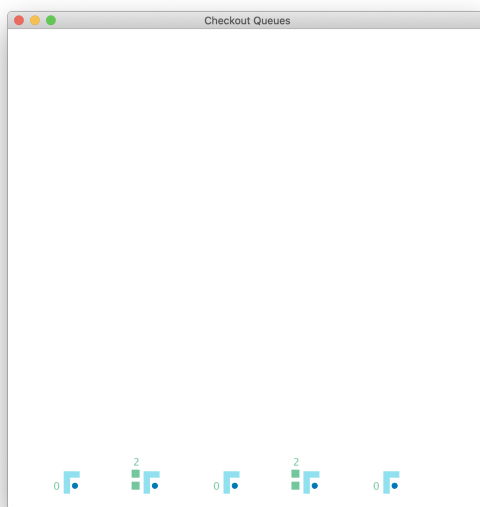
In my project, I used `Queue` instances to represent the queuing line for each `CheckoutAgent`, modeling the real life case where customers queue behind an checkout count in a mall. Every new customer is added to the queue by `offer()` and every customer who finishes checkout will be removed to the queue by `poll()`. The remaining `peek()` can call the `giveUpItem()` method on the customer that is checking out, then to examine whether one customer has finished checkout by inspecting the number of items that customer has.

# Results

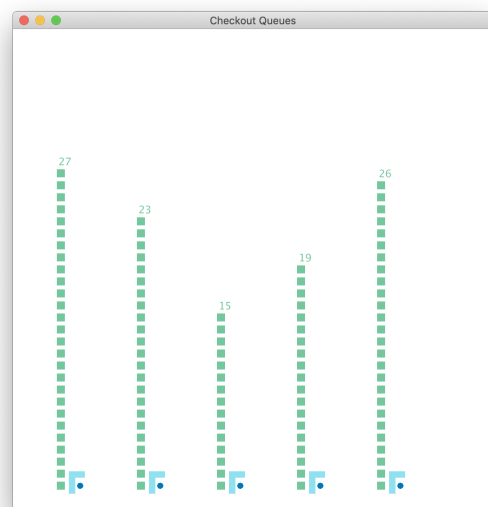
I tried with different values of maximum items when instantiating the customer, and believed that when the value is 6, lines under all three strategies are more manageable and not likely to grow too long. Images below.

## RandomCustomerSimulation:

We can see the queue length when number of item ranges from 1 to 10 is not evenly distributed and also long, while that when number of item goes from 1 to 6 is much shorter.



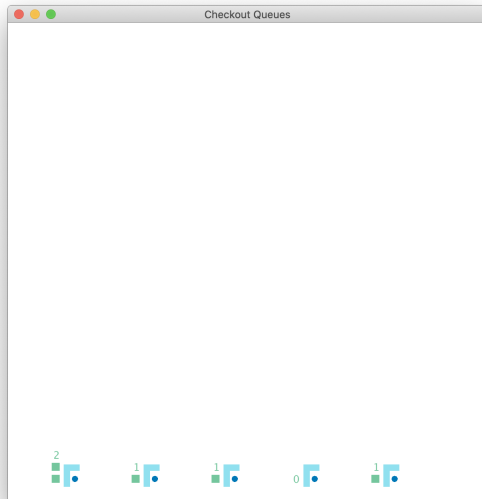
RandomCustomerSimulation (items 1-6)



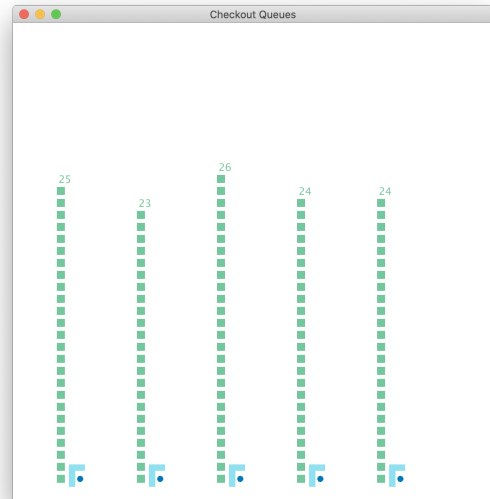
RandomCustomerSimulation (items 1-10)

## Pick2CustomerSimulation:

We can see the queue length is also pretty long when the number of item ranges from 1 to 10; however, it is more evenly distributed compared to the RandomCustomerSimulation. Holding the same trend with the previous simulation, when the number of item ranges from 1 to 6, the queue becomes much shorter.



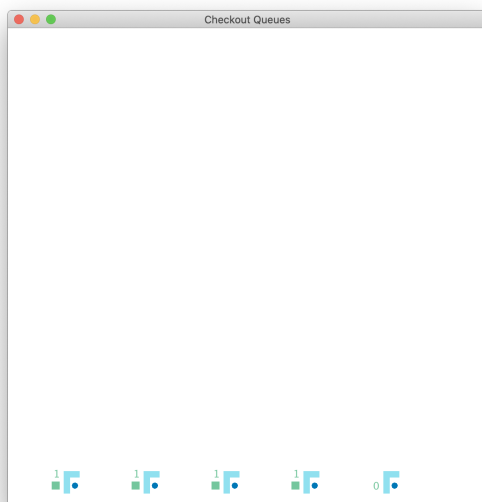
Pick2CustomerSimulation (items 1-6)



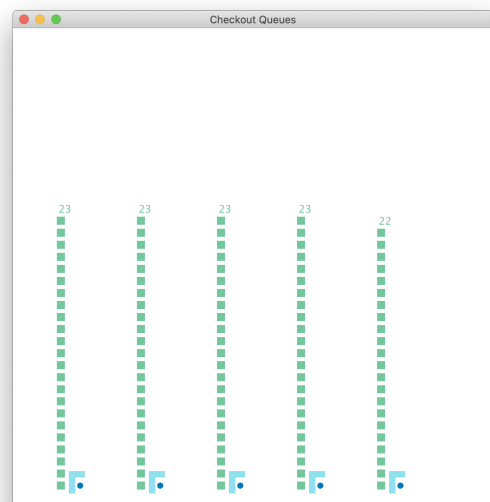
Pick2CustomerSimulation (items 1-10)

### PickyCustomerSimulation:

We can see the queue length in this simulation is the most evenly distributed among all three. Similar to the previous three, when number of items goes from 1 to 10, the queue length is very long and congested, while it is much shorter when smaller number of items, i.e. from 1 to 6.



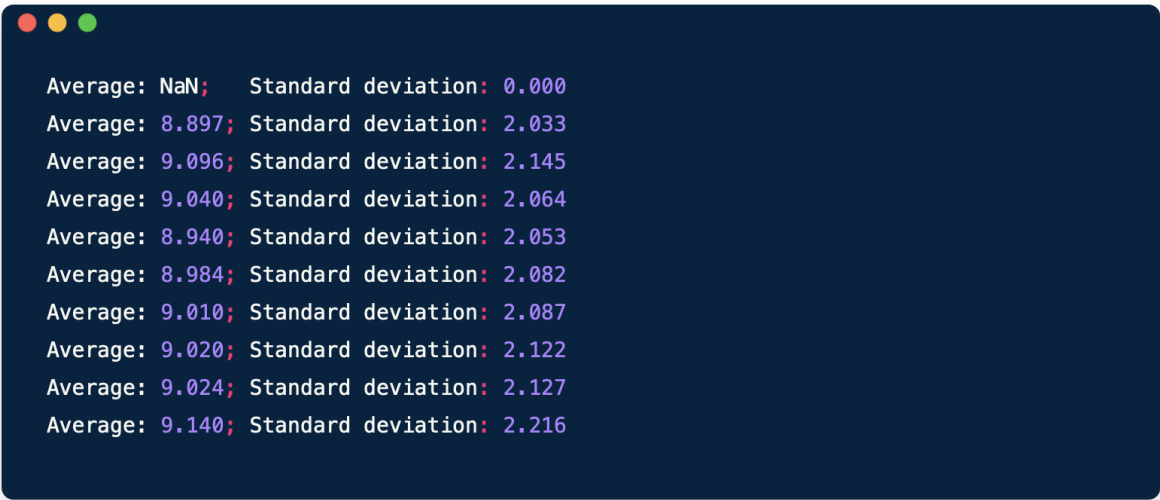
PickyCustomerSimulation (items 1-6)



PickyCustomerSimulation (items 1-10)

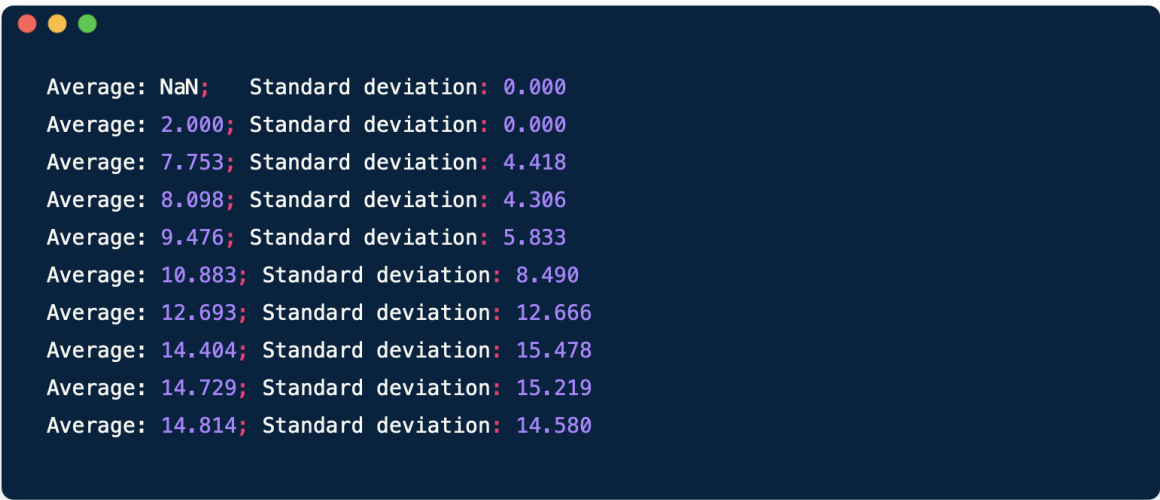
I also printed out the required statistics, periodically as running all three simulations, of the average and standard deviation of the time-to-leave for all of the Customers in the finished customer list.

### **PickyCustomer:**



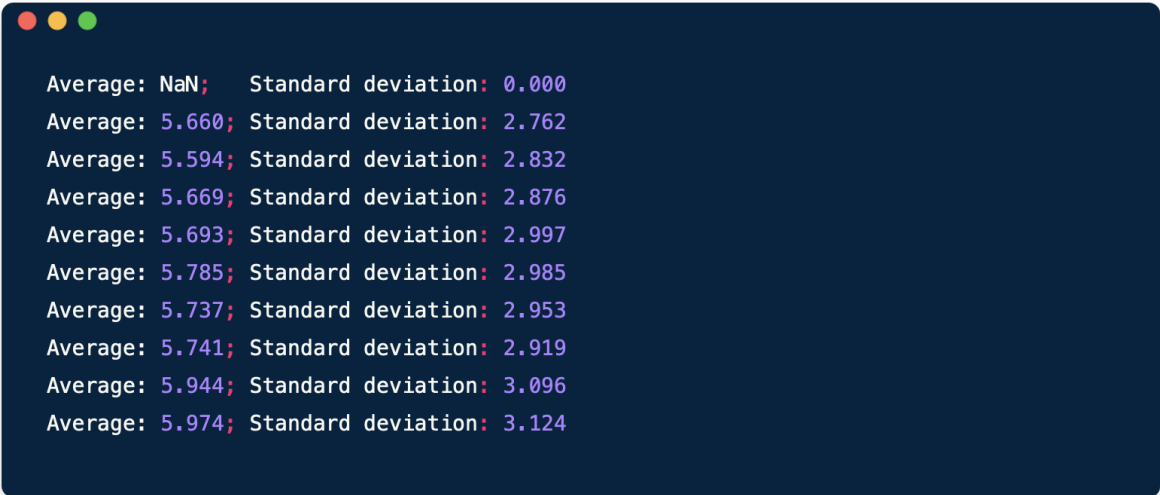
```
Average: NaN; Standard deviation: 0.000
Average: 8.897; Standard deviation: 2.033
Average: 9.096; Standard deviation: 2.145
Average: 9.040; Standard deviation: 2.064
Average: 8.940; Standard deviation: 2.053
Average: 8.984; Standard deviation: 2.082
Average: 9.010; Standard deviation: 2.087
Average: 9.020; Standard deviation: 2.122
Average: 9.024; Standard deviation: 2.127
Average: 9.140; Standard deviation: 2.216
```

### **RandomCustomer:**



```
Average: NaN; Standard deviation: 0.000
Average: 2.000; Standard deviation: 0.000
Average: 7.753; Standard deviation: 4.418
Average: 8.098; Standard deviation: 4.306
Average: 9.476; Standard deviation: 5.833
Average: 10.883; Standard deviation: 8.490
Average: 12.693; Standard deviation: 12.666
Average: 14.404; Standard deviation: 15.478
Average: 14.729; Standard deviation: 15.219
Average: 14.814; Standard deviation: 14.580
```

### **Pick2Customer:**



```
Average: NaN; Standard deviation: 0.000
Average: 5.660; Standard deviation: 2.762
Average: 5.594; Standard deviation: 2.832
Average: 5.669; Standard deviation: 2.876
Average: 5.693; Standard deviation: 2.997
Average: 5.785; Standard deviation: 2.985
Average: 5.737; Standard deviation: 2.953
Average: 5.741; Standard deviation: 2.919
Average: 5.944; Standard deviation: 3.096
Average: 5.974; Standard deviation: 3.124
```

As the previous images show, the Pick2Customer has the lowest average time-to-leave value of 5.7 while the RandomCustomer has the largest, around 12. The RandomCustomer also has the largest value of standard deviation of around 12, while the other two tend to remain theirs around 2 to 2.85, with the big difference of 3 resulting from the randomness when choosing a line.

A conclusion can be safely drawn that with five checkout lines open in the simulation, the Pick2Customer has the optimal line choosing strategy compared to the other two.

## Acknowledgement

I did the project independently; however, I didn't manage my time well and missed the deadline, so I decided not to finish extension this time.