Simulating a Post Office: A Report

SID: 690000912

Design

To emphasise modularity and compartmentalisation, I arranged each function defined outside the main function in 'simQ.c' to be in its own file, included via header files containing the function signatures and type definitions.

Originally, the Queue was implemented as a wrapper around a dynamically allocated array of Customer objects. Inside the Queue type definition, two pointers kept track of the head and tail of the Queue, and the array was circularly buffered to prevent having to shift the contents of the array on removal of an object from the front of the queue. This was chosen to emphasise ease of use, as direct access of elements in the array was simple, plus with constant time complexity. However, whilst implementing customers being able to time out and leave the main queue, using an array made this impractical; removing an item from the center of the array meant that items would have to be shuffled, regardless of the circular buffer, plus the shuffling had to be with regards to the head/tail pointers. Other implementations were considered, such as a tagging system to mark a customer as having timed out whilst still being physically present in the queue, to be purged at a later time- this, however, made it much harder to predict space requirements for the Queue's array.

With this then, the implementation of the Queue was switched to a doubly-linked list, to allow for backwards referencing with items, and for much easier removal of nodes mid-queue.

All type-defined instances are referenced via pointers, with functions returning and taking pointers to Queues/Customers. This helps maintain consistency for knowing what to free and when, whilst also keeping the actual data being manipulated in a single instance.

Customers' attributes are all generated randomly around a mean and variance that may be altered in the input file. This is to allow for customisation of customers past simple quantities and flat values- variance must be inherent to an object simulating a human or modelling human-like behaviour. This means that variance can be simulated not just across executions, but also across simulation passes, and even from customer to customer in a pass, whilst still maintaining a probabilistic space as defined by the user.

Experiment

To begin our experiment, it is important to establish a baseline for all manipulated simulations to be compared to.

Using the time step iterator variable to denote minutes, we can say that an 8.5-hour period (the traditional length of a Post Office opening time [1]) is 8.5 * 60 iterations, or 510 iterations.

Information for the average number of service points at a Post Office is hard to find, especially with the increase in self-service at agency branches (branches managed by a larger franchise, such as W.H. Smith). Hence, we shall use more familiar information; at my local Post Office, there are three service points.

As for the number of customers in a Post Office branch on average per day, numbers published by the Post Office in their 2015 Network Report states that, on average, 17 million people visit a branch per week. [2]. In March 2015, the Post Office had 11,634 branches. In this case, barring concrete statistics, we can derive a rough estimate for customers per branch per day; 17000000 / 11634 / 7, or 208.74 customers. Therefore, using a mean of 208 customers, and a variance of 1 (as we have no information for busiest branch vs quietest branch), we have two more values for our benchmark.

In a 2018 Consumer Trends Survey by Lightspeed, the average length a customer will wait in a queue is 6 minutes, 46 seconds. [3] We can use this as the average, or mean, waiting tolerance, for our customers: 6 time steps. We also have a rough variance of around a minute (the most tolerant customers waiting 7 minutes and 7 seconds, and the least tolerant waiting just over 6 minutes).

Service time was not a metric I could find a verifiable source for, so we will go off personal experience again, and say a rough serving time of 2 minutes, or time steps. Queue length, as well, was hardly verifiable, so we shall assume a maximum queue length of 15, since if I saw a queue for the post office that long, I would go and do something else in town first.

With this, our input file contains the following:

```
maxQueueLength = 15↓
numServicePoints = 3↓
closingTime = 510↓
↓
numCustomersMean = 208↓
numCustomersSD = 1↓
customerServiceTimeMean = 2↓
customerServiceTimeSD = 1↓
customerToleranceMean = 6↓
customerToleranceSD = 2↓

```

Running the simulation 1000 times, we get this output:

```
Sim count: 1000
Maximum queue length: 15
Number of service points: 3
Closing time: 510
Number of customers mean: 208
Number of customers standard deviation: 1
Customer service time mean: 2
Customer service time standard deviation: 1
Customer tolerance mean: 6
Customer tolerance standard deviation: 2

No. of total customers over 1000 simulations: 208724
Average no. of unfufilled customers: 0.00
Average no. of timed-out customers: 0.00
Average no. of fufilled customers: 208.72
Average no. of time steps after closing: 2.55
Average no. of time steps fufilled customers waited for: 0.01
```

What does this mean then? Well, every single customer in all 1000 simulations were served practically immediately, with only very minor wait times for any customer at any point. The post office closed roughly two minutes after the final customer entered the queue.

Let us simulate then a period around Christmas and say that twice the amount of customers enter to send presents and good will to their family and friends via the post. Therefore, our input reads as follows:

```
maxQueueLength = 15
numServicePoints = 3
closingTime = 510

numCustomersMean = 516
numCustomersSD = 1
customerServiceTimeMean = 2
customerServiceTimeSD = 1
customerToleranceMean = 6
customerToleranceSD = 2
```

And our output after 1000 simulations reads as:

```
Sim count: 1000
Maximum queue length: 15
Number of service points: 3
Closing time: 510
Number of customers mean: 516
Number of customers standard deviation: 1
Customer service time mean: 2
Customer service time standard deviation: 1
Customer tolerance mean: 6
Customer tolerance standard deviation: 2

No. of total customers over 1000 simulations: 516724
Average no. of unfufilled customers: 0.00
Average no. of timed-out customers: 3.83
Average no. of fufilled customers: 512.90
Average no. of time steps after closing: 5.20
Average no. of time steps fufilled customers waited for: 1.62
```

We can see that the average wait time was around 1 minute 30 seconds, impressive for a post office, but not good enough for almost 4 customers on average per

simulation, who left the queue. No-one however, turned away from the queue upon seeing its length.

Perhaps then, the customers are more irritable then usual when it comes to the festive period, and the average waiting tolerance drops from around 6 minutes to 4 minutes. The output then becomes:

```
Sim count: 1000
Maximum queue length: 15
Number of service points: 3
Closing time: 510
Number of customers mean: 516
Number of customers standard deviation: 1
Customer service time mean: 2
Customer service time standard deviation: 1
Customer tolerance mean: 4
Customer tolerance standard deviation: 2

No. of total customers over 1000 simulations: 516702
Average no. of unfufilled customers: 0.00
Average no. of timed-out customers: 9.47
Average no. of fufilled customers: 507.23
Average no. of time steps after closing: 4.91
Average no. of time steps fufilled customers waited for: 1.26
```

Interestingly, whilst bad moods have meant almost ten people per simulation on average have left the queue, the actual waiting time decreased for the average fulfilled customer, as less people were present in the queue to extend waiting times.

Issues

One issue with how the customers are dispersed throughout the day is the random number generator that creates an array of integers, depicting how many people join at what time, is naïve, meaning that if the ratio of closing time to mean of customers is too large, an appropriate array cannot be generated. This could be fixed by implementing a hard lower limit on the generation, or by using a ceiling rounding function instead of a round-to-nearest algorithm in the generation.

[1] Annabel Barnett, National Association of Citizens Advice Bureaux, Consumer Use of Post Offices, June 2017.

[2] Post Office Limited, Network Report 2014/2015, accessed 15/02/2021. https://www.postoffice.co.uk/dam/pdf/network-report-2014-2015.pdf

[3] Lightspeed, Lightspeed EPOS 2018 Consumer Trends Survey.

https://www.lightspeedhq.co.uk/blog/customers-wait-6-minutes-46-seconds-queue-can-create-efficient-service-uk-shoppers