CS231: Project 5

Checkout Lines

Parth Parth | CS231L-A | Dr. Alan Harper

Abstract

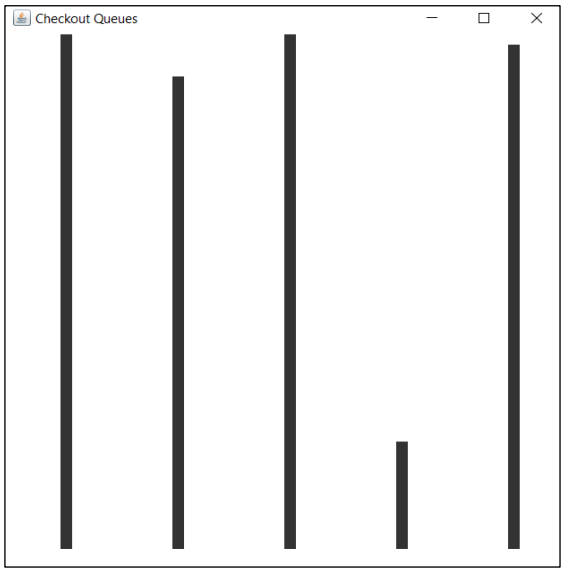
In the project, a simulation of checkout lines has been made. Different strategies have been employed by the simulated customers to compare them and not only get an optimal strategy, but also get an optimal number of items for checking out using each strategy.

The data structure used to implement this is a queue. This is not different from a daily life queue in that the first person to enter the queue (first in) is the first person to leave the queue (first out). This is called a FIFO data structure. In this project, the queue is a simple linear queue that is one ended (i.e. one entry point and one exit point only).

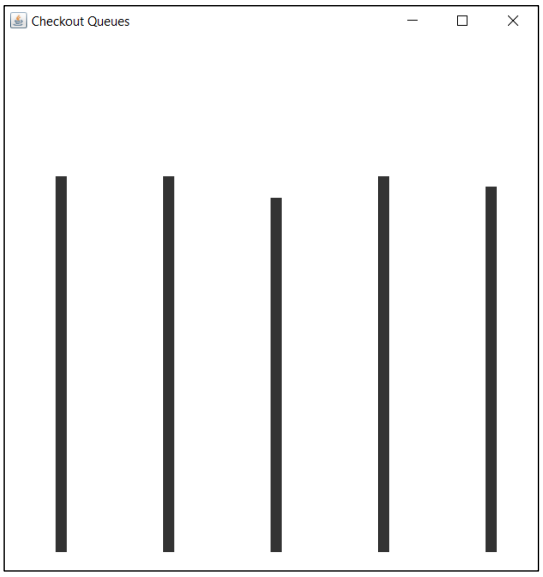
Experimentation and Results

In the following pictures, the PickyCustomer, RandomCustomer, and Pick2Customer classes have been shown. The purpose is to exhibit how there is an optimal number of items for each strategy, exceeding which, the queues grow exceptionally long and keep growing.

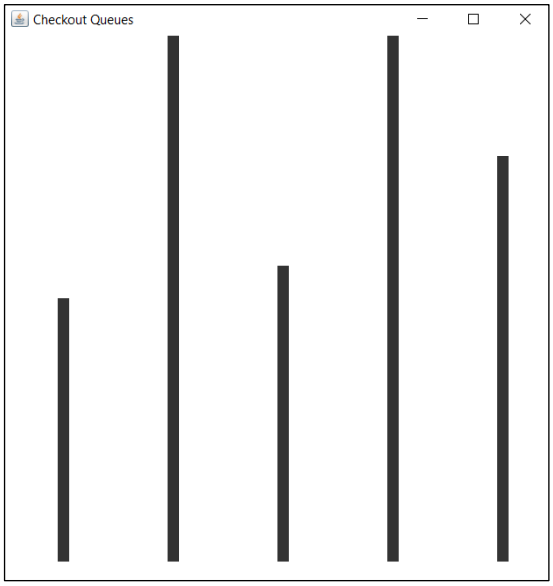
Queues growing too long:



RandomCustomer with 5 checkout queues and between 1-10 items after 2000 time steps

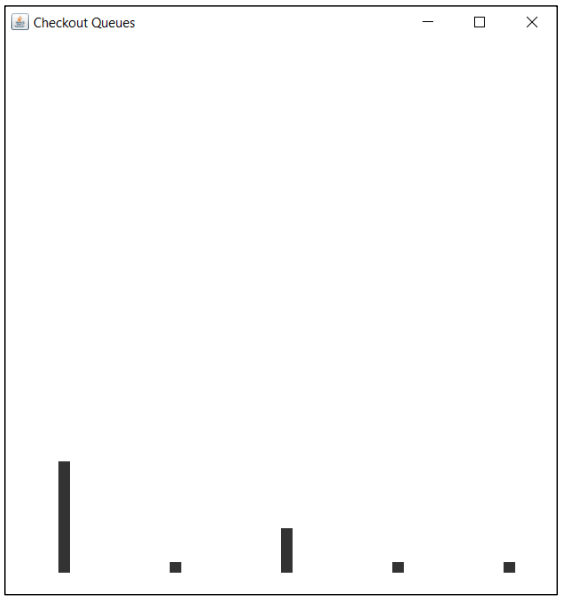


PickyCustomer with 5 checkout queues and between 1-10 items after 2000 time steps

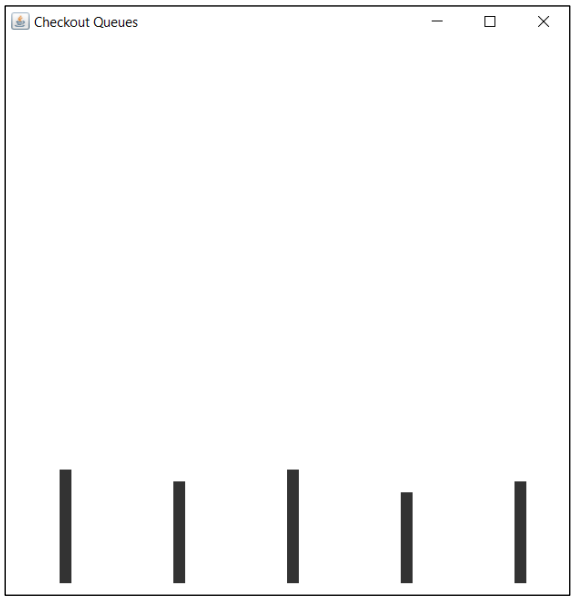


Pick2Customer with 5 checkout queues and between 1-10 items after 2000 time steps

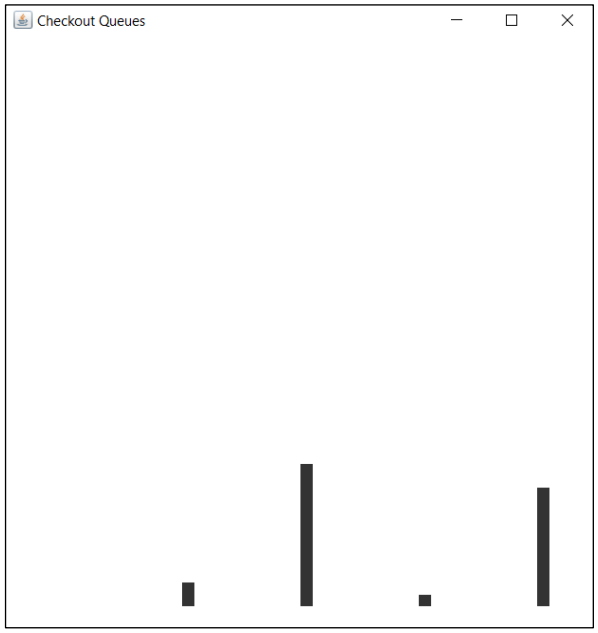
More manageable queues:



RandomCustomer with 5 checkout queues and between 1-8 items after 2000 time steps



PickyCustomer with 5 checkout queues and between 1-8 items after 2000 time steps



Pick2Customer with 5 checkout queues and between 1-8 items after 2000 time steps

Therefore, the optimal number for *RandomCustomer* is 8, *PickyCustomer* is 8, and *Pick2Customer* is 8. *PickyCustomer* can also use 9, but the queues oscillate between becoming so big to go off the canvas and medium, and the time jumps by 12x, so the optimal number is 8.

All three kinds of customers were simulated for 1000 time steps with average time and standard deviation per customer being calculated after every 100 time steps. The results were as follows:

Time steps	Random Customer		Pick2 Customer		Picky Customer	
	Average	S.D.	Average	S.D.	Average	S.D.
100	10.9	7.2	9.2	3.4	10.2	20.8
200	15.0	12.5	8.9	3.4	11.4	3.3
300	18.7	14.9	9.0	3.6	11.1	3.3
400	19.3	14.6	8.5	3.6	11.0	3.1
500	19.6	13.7	9.1	4.0	10.7	3.0
600	19.3	12.9	9.4	4.2	10.5	3.0
700	18.6	12.4	9.5	4.1	10.5	3.0
800	18.8	13.5	9.6	4.2	10.4	3.0
900	19.1	13.5	9.7	4.1	10.3	2.9
1000	18.9	13.1	9.7	4.1	10.3	3.0
Average	17.82	12.83	9.26	3.87	10.64	4.84

S.D.=Standard Deviation

Thus, it is clear that Pick2Customer's strategy is the best since it has the least average time and also the least standard deviation.

Reflection

For this project, I learned how to implement a queues using generics which led me to discover how Java does not allow arrays of generics (Extension 1). I also learnt that the most efficient strategy over a large number of people checking out in a line is to pick two lines randomly and select the smaller line from that.

Extension 1

In this extension, I have implemented the queue using an array instead of using a Node based queue implementation.

An interesting exploration tangent this extension taught me was that Java does not allow creating arrays of generic types, so I had to create an array of type *Object*. This works as all classes defined by the user would automatically be subclasses of the *Object* class.

Extension 2

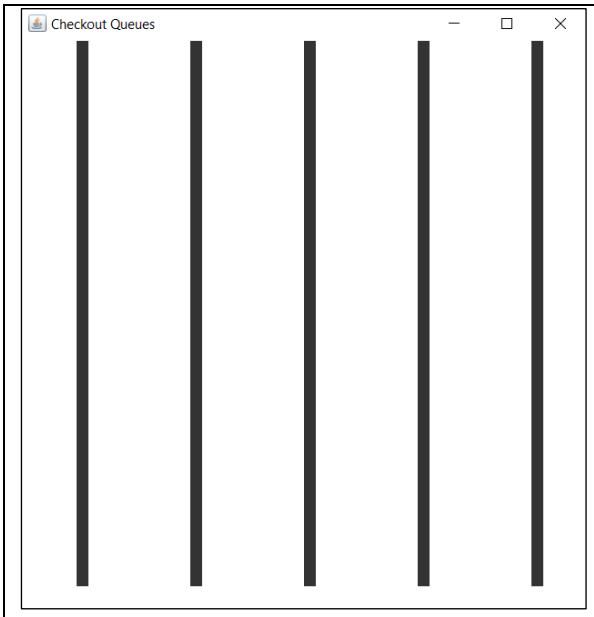
In this extension, I have implemented a Customer that chooses the line based on the number of items in each line. The initial time steps are counted by basically going to each customer and having them count all the items which takes one time step each. The customer was called PickItemCustomer and the results are as follows:

Time steps	Random Customer		Pick2 Customer		Picky Customer		PickItem Customer	
	Average	S.D.	Average	S.D.	Average	S.D.	Average	S.D.
100	10.9	7.2	9.2	3.4	10.2	20.8	19.3	6.3
200	15.0	12.5	8.9	3.4	11.4	3.3	17.6	5.8
300	18.7	14.9	9.0	3.6	11.1	3.3	17.1	5.5
400	19.3	14.6	8.5	3.6	11.0	3.1	17.6	5.7
500	19.6	13.7	9.1	4.0	10.7	3.0	18.4	6.1
600	19.3	12.9	9.4	4.2	10.5	3.0	18.8	6.5
700	18.6	12.4	9.5	4.1	10.5	3.0	18.8	6.5
800	18.8	13.5	9.6	4.2	10.4	3.0	18.2	6.4
900	19.1	13.5	9.7	4.1	10.3	2.9	18.9	7.2
1000	18.9	13.1	9.7	4.1	10.3	3.0	21.0	9.9
Average	17.82	12.83	9.26	3.87	10.64	4.84	18.57	6.59

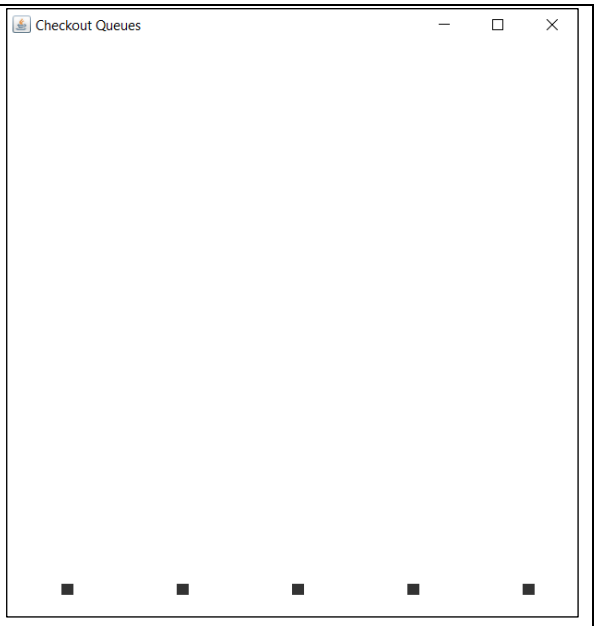
S.D.=Standard Deviation

Clearly, *PickItemCustomer*, with the highest average time per customer, is the least efficient of all three strategies. (continued on next page)

Also, the optimal number of items is 8. Even though the queues did not go over the canvas in 1,000,000 iterations, the average time kept increasing constantly, so the optimal items has been chosen to be 8.



PickItemCustomer with 5 checkout queues and between 1-10 items after 2000 time steps



PickItemCustomer with 5 checkout queues and between 1-8 items after 2000 time steps

References/Acknowledgements

The *LandscapeDisplay* class was retrieved from https://cs.colby.edu/aharper/courses/cs231/f21/labs/lab05/LandscapeDisplay.java .