STORE QUEUE OPTIMIZATION

**Team Members:**

**Daksh Khandelwal, Dhruv Khanna, Dhruvi Shah, Duc Long Ngoc, Pratiksha Sharma, Purav Tekwani, Rithwik Chhugani & Swraj Palekar**

**Background & Industry Perspective:**

Nowadays, supermarkets are the main way people get groceries. With a total retail sales of 18800 B (USD) in 2011 and is predicted to reach 31300 in the year of 2021. On average, a person would spend 50-100 AUD/ week on groceries in Australia. But nowadays, with the development of smart phone, people have become less and less patent.

**Introduction:**

People generally get groceries from stores and supermarkets. While stores only have a limited type of item, a supermarket is normally big and have a wider selection of items. But browsing, searching and checking out in the supermarket is time-consuming, especially in rush hour. Many times, we have witnessed few active counters in the store during peak hours leading to hour long queues and fights due to the reducing tolerance level of individuals. This in turn, also reduces customer satisfaction.

One way to solve this is to add more counters to speed up check-out process. But with the fluctuation of number of customers at a particular time in a particular day, it is difficult to find the number of counters which keep the customer satisfaction above a safe threshold while remaining supermarkets’ profit. To solve this problem, we thought to run simulations in controlled setting to understand the number of active queues the stores require during peak hours to avoid massive delays and low customer satisfaction while maximizing store savings.

**Problem Statement**

Our simulation problem is based on optimizing the satisfaction and wait-times of a consumer by minimizing the number of active cash-counters which in turn reduces the operational cost incurred by the store.

1. Function or objective to be optimized: Multi-Channel Queue Optimization

2. Parameters or variables to be used:

Customer Population – Arrival Time, Tolerance, Cart Size, Picking Rate, Status etcetera.

Checkout/Queue Operator – Efficiency / Packing Time.

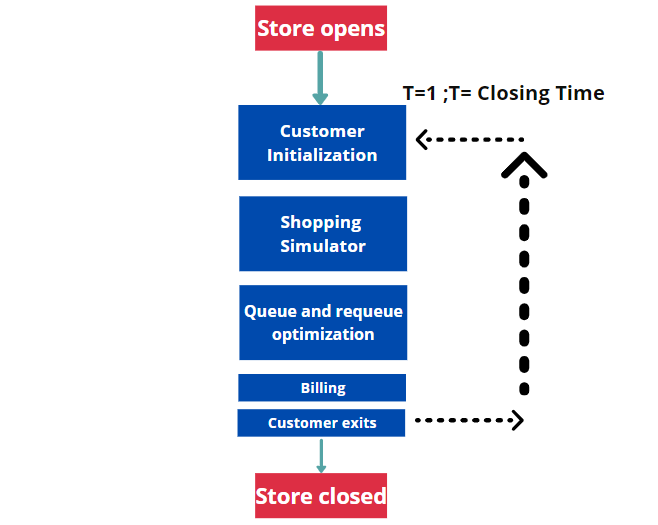
Customer Satisfaction – Calculated using a custom equation using the existing variables.

**Methodology**

**Simulation process idea:**

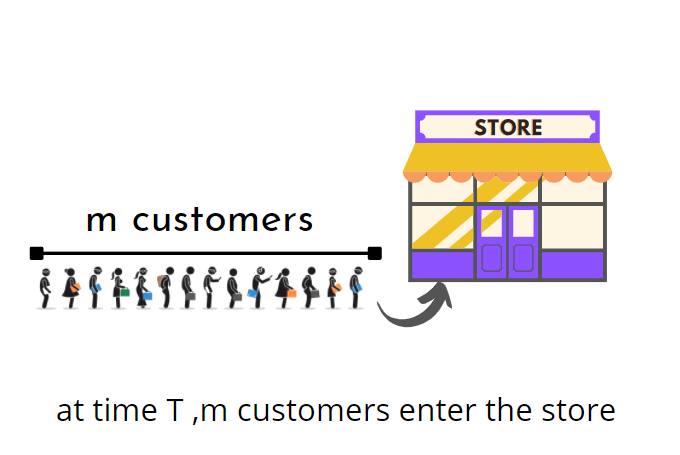
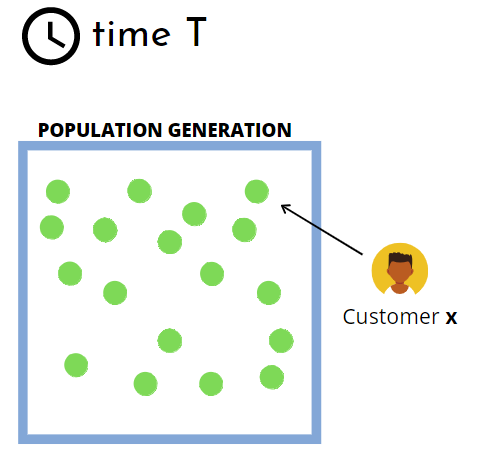
The Time Loop:

The idea of a time loop is a simple iteration through each second of the store opening times. Within the time loop as it iterates further, we observe all the processes mentioned ahead. New customers walk in, they start buying a set of items. While buying the items the tolerance of the customers also keeps decreasing (basically the customers get tired as they spend time buying the items). As soon as they’re done with shopping, they are channeled to the queue where the billing takes place. The queue channel does 2 kinds of processes:

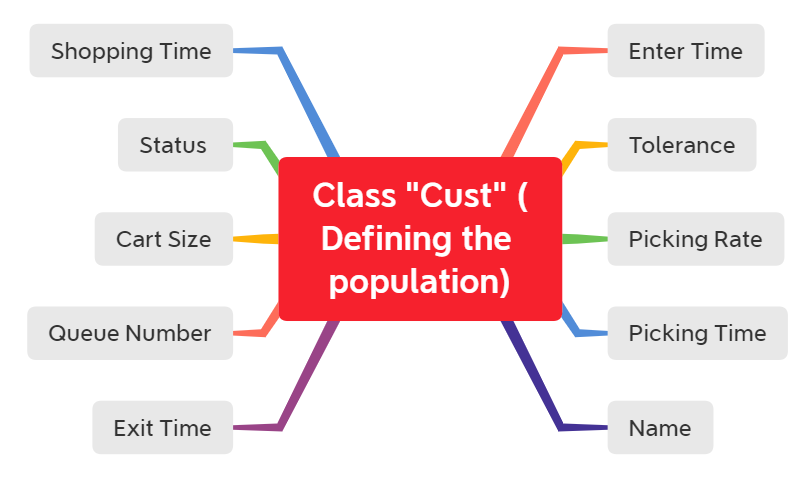
* Popping customers out of the queue: As soon as the customers are done with the billing process they’re popped out of the queue and they leave the store.
* Pushing Customers in the queue: This operation is adding the customers done with adding items to their carts to a cash counter queue where their billing takes place

A simulation of the queue processing happens in the time loop where new people are added to the queue with time as they’re done with adding items to the cart. This way the queue proceeds with time and the orders are processed. While customers stand in the queue their wait time is noted and as this wait increases, the customer satisfaction gets penalized. This penalization and final customer satisfaction levels after they’re done with shopping is the problem that we’ll be solving in the project. Building optimization techniques for this multichannel queue problem will be discussed further in the report. It is also important to note that although the simulation runs for 15 hours a day, entry of new customers is halted slightly before the store closing time. This ensures that all checkouts have been successfully processed before shut down.

**Initialization of customers and the beginning of the TIME LOOP**



The initial stages of the problem statement require the user to build a population. Defining a population is the first building block of the problem statement. The size of the population is defined by a variable to have the flexibility to simulate different flows of crowd.

We have opted for an object-oriented approach to generate the population where the number of instances created of the S4 class will signify the number of people part of the simulation. The class named “cust” initializes the variables given in the adjoining diagram.

All these variables will help us to set attributes for the existing population. Since we need the population to be of a decent size, a loop is the best option for instance generation.

Inside the loop we define n people who are the part of the population and set their variables as follows:

* Enter time -> manually initialized
* Tolerance -> comes from a normal distribution
* Picking Rate -> comes from a normal distribution
* Cart Size -> a discrete number sampled from 1 to the maximum cart size limit (to bound the problem statement)
* Status -> set to 1 [ 1 (going to store), 2(added to queue), and 3(Exiting)]
* Shopping Time -> Picking Rate \* Cart Size
* Exit Time -> Shopping time + Wait time (Wait time is the time taken to bill the orders)
* Picking Time -> Time taken to add the items to the cart (Picking Rate \* Cart Size)
* Name -> Id for each customer
* Queue Number -> Queue assigned to the person

In those variable, Tolerance, Picking Rate, and probability of going to the shop follow the normal distribution with bounds at 2 ends. The function named “norm\_dist” is created to serve this purpose. While the cart size follows a uniform distribution and the status is the random number without biases between 1 and 3.

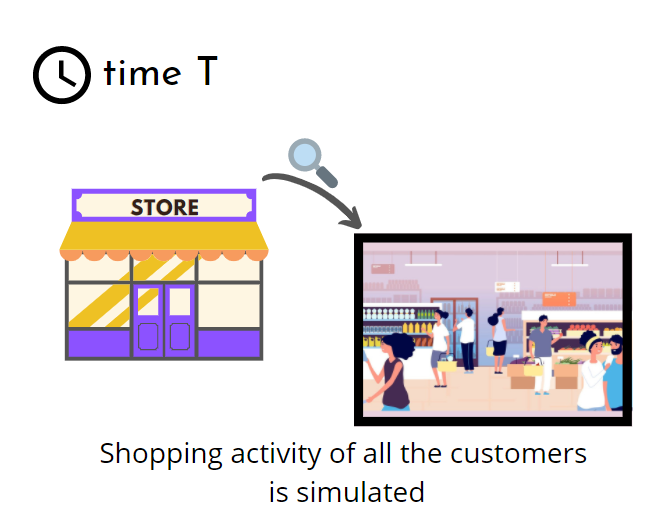
The next task was to declare a set of variables mentioned above these were found through trial and error while tolerance and picking rate are found through a normal distribution.

* m\_tol = mean of tolerance for customers
* Sd\_tol = standard deviation of tolerance for customers
* m\_picking = mean picking rate (picking the products) for customer
* sd\_picking = standard deviation picking rate (picking the products) for customer
* Max\_cart = maximum number of products in a cart
* Store\_close\_time = total hours in seconds
* Cust\_time\_m = mean number of customers entering the store every given time of the day
* Cust\_time\_sd = mean number of customers entering the store every given time of the day
* Go\_to\_queue = empty vector which will have people who are done with shopping and are entering the queue for checkout
* Cust\_in = empty vector which will have people entering the store

Further we have assumed the total hours that the store remains open for is 15 hours. We have a counter for time which increases with every iteration of the for loop that is being used ahead.

Another assumption is the mean number of customers present in the store during a particular hour of the day, cust\_time\_m and cust\_time\_sd give the number of customers in the store. Packing rates are randomized through a normal distribution

**Shopping Distribution**



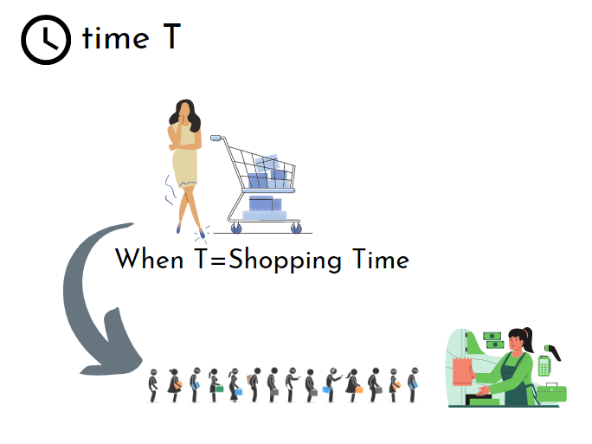
Assumption: all items are equidistant

The main task for this section was to create a loop that gives the number of people in the store along with their individual enter times and shopping times.

Each customer’s shopping time is determined with the help of their picking rate and cart size.

picking\_time = picking\_rate1 \* cart\_size

shopping\_time = time + picking\_time



Once a customer is done shopping, the next step is to calculate new tolerance for the customer by subtracting picking rate \* alpha(constant) which is set to 0.0085 from the old tolerance. This is calculated to maintain the decrease in the tolerance while the person shops.

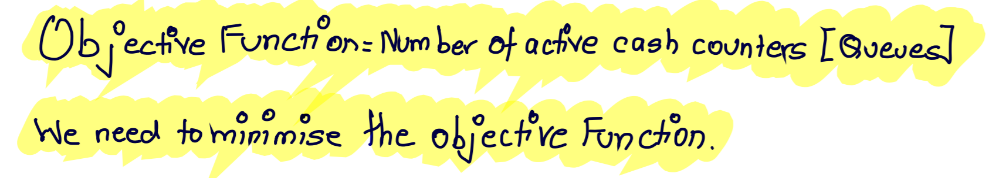
After all these calculations are made the person is then sent to the queue and the status is updated to 2.

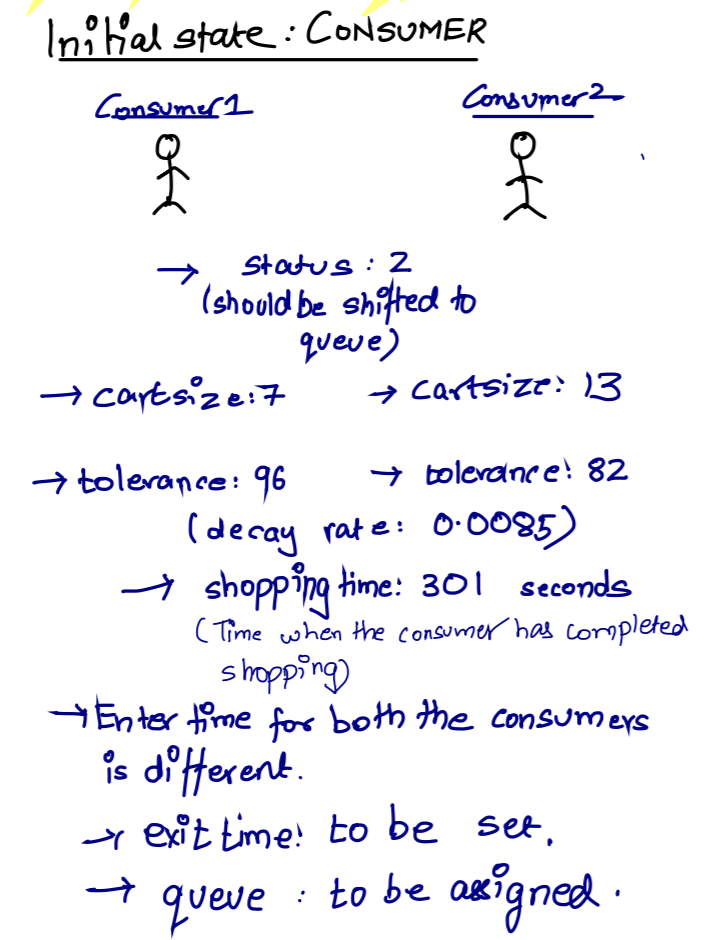
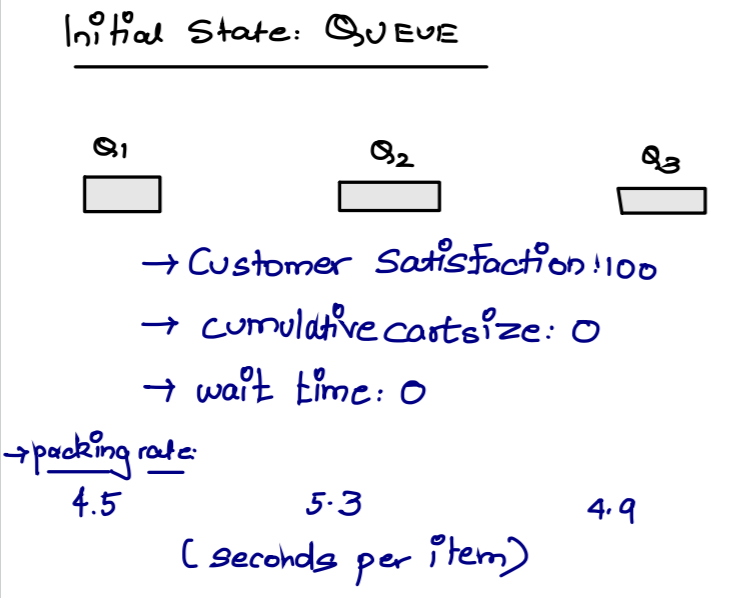
**Optimization Algorithm**

In this project, we are trying to minimize the number of active queues or cash counters in a store and have thus formulated a custom optimization algorithm for the same. All metrics (or equations) that are calculated and tuned in our process have also been formulated from scratch. The optimization of this number of queues takes place in various stages of the program such as pushing the customers in a queue.

**The Initial States and Equation Definitions**

Objective Function and Initial States:





We have incorporated several pictorial representations using examples to make the understanding of the process clear.

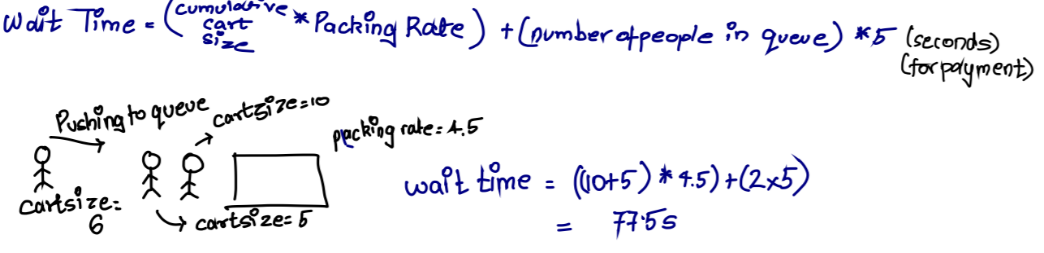
Equation Definitions

1. Wait Time

Wait time is the time a customer at the back of the queue will take to reach the front of the queue.

For its inputs, we require:

* Num\_queue à Number of people in front of the incoming customer.
* Cum\_cart à summation of number of products for all the customers in front of the incoming customer.
* Packing\_t à packing time for the counter.
* Constant ‘5’ à multiplying 5 seconds to each customer as ‘payment time’.

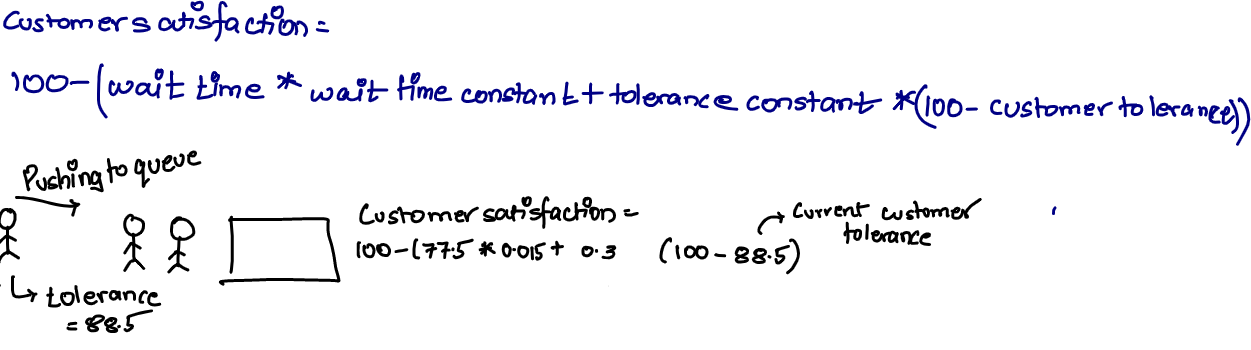


1. Customer Satisfaction

Customer Satisfaction is the satisfaction level of the last person in each queue

Required inputs:

* Wait\_time à wait time of the last person in the queue the customer is joining
* Cust\_tol à the tolerance level of each customer (each customer has different tolerance levels)
* Wait\_time\_c à a designated wait time constant for each customer by which the wait time increases
* Tol\_con à the rate at which the customer tolerance penalizes the customer satisfaction level of the queue.



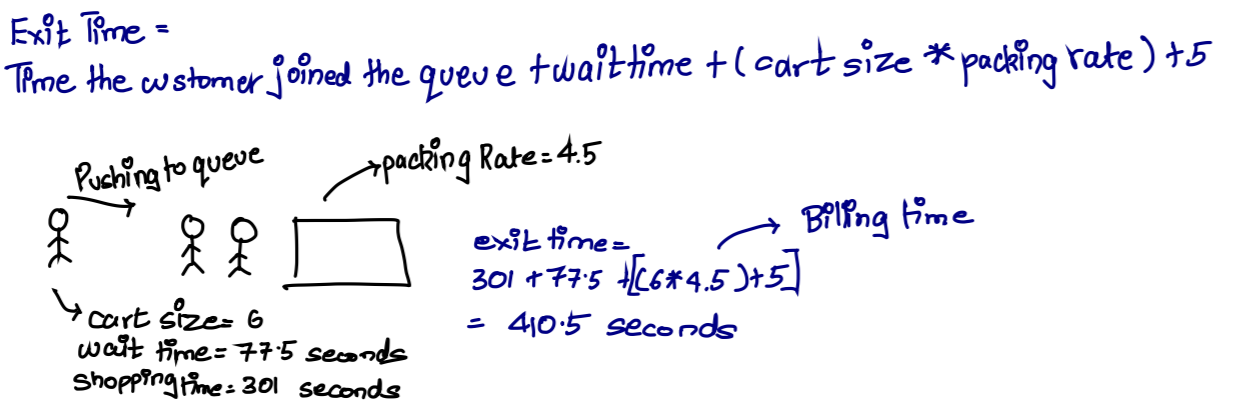
The constants 0.015 and 0.3 have been chosen after substantial trial and errors.

1. Exit Time

After we have defined the time at which a person enters a queue for shopping, we will immediately assign him an exit time (in seconds) at which he exists the queue and done with his billing.

Required Inputs:

* Join\_queue\_time à the time the customer joins the queue
* Wait\_time à wait time for the customer.
* cartSize à cart size for the incoming customer.
* Packing\_t à packing time for each product.



**Defining Queue properties**

* Packing\_m à mean packing time.
* Packing\_sd à Standard Deviation for packing time.
* q\_packing\_rates à initialise 6 packing rates for 6 defined queues.
* q\_data àqueue data for each of the 6 queues.
* q\_s à defining satisfaction level for each queue.
* q\_cum\_carts à vector for loop number of products in each queue.

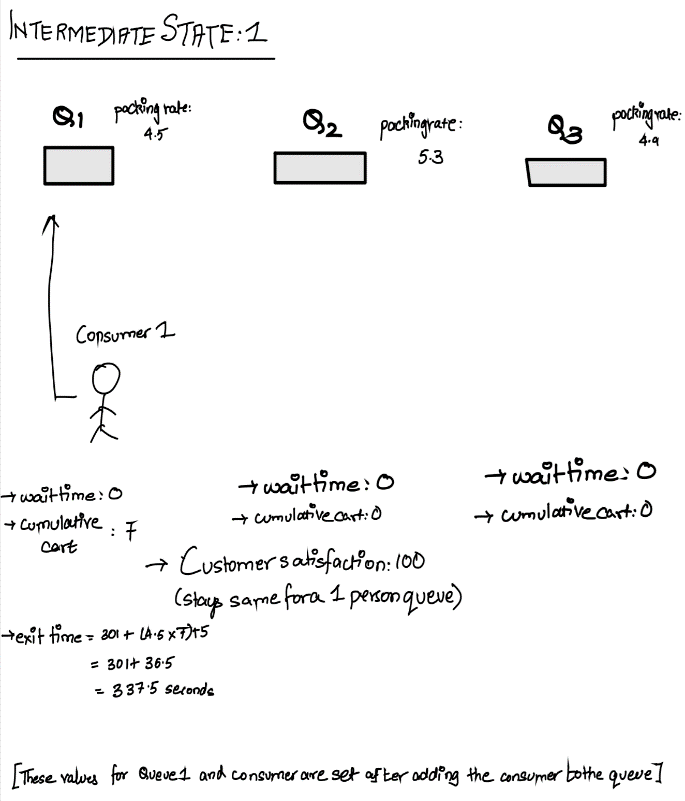
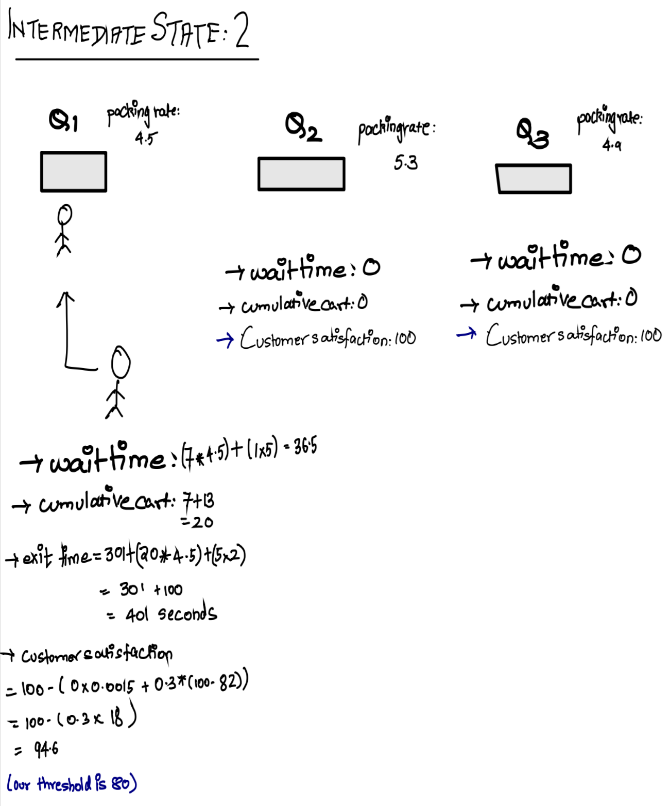
**Pushing a customer to a queue**

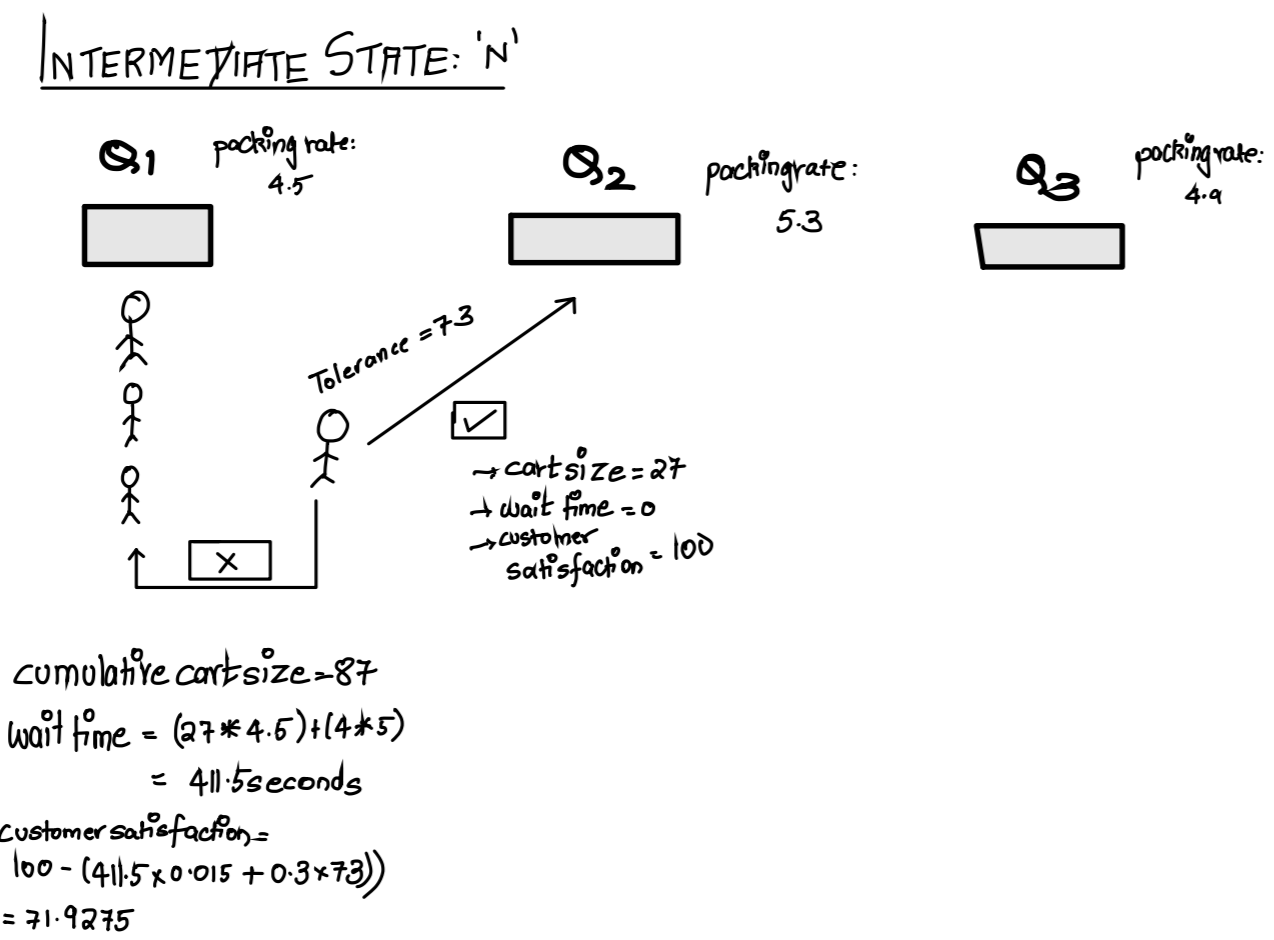
This algorithm is for pushing a customer to a queue. We push a customer to the queue after we have popped customers who might have completed their billing (popping will be discussed later on as it is skipped for the very first iteration where all queues are empty). This is only done when there is an incoming stream of customers, that is, current\_cust (it is a list of customers wanting to join a queue at time ‘t’) is not null.

The outer loop traverses through the current\_cust list while the inner loop facilitates addition of a customer to a respective queue. The algorithm works in a way that the initial queues will always be preferred rather than starting another counter. If the current customer satisfaction of a queue is greater than a certain threshold (in our case, we decided it to be 80), the customer will be added to the end of that queue. The exit time of the customer will be estimated here which is used for popping at a later stage.

In the case of a 1-person queue, the customer satisfaction is reset to the initial value (in our case, 100), signifying that a second person can freely join.

Subsequently, the queue metrics would be re-calculated using the data provided by the newly added customer. The sketches below would help elaborate on the same.



**Popping a customer from a queue**

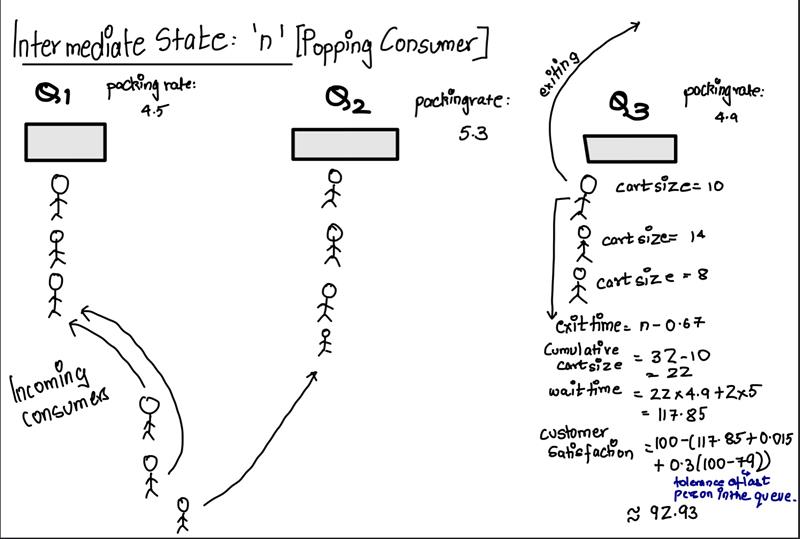
This algorithm is for popping a customer from a queue. (This is sequentially carried out before pushing.) We will first remove a customer from the queue before adding another person to the same queue. This is done to make sure that all the metrics for a particular queue to add a new customer are calculated accurately.

We start with looping through 10 predefined queues. In case current queue is empty, the control is shifted to the next queue.

If the queue is not empty, the exit time of the first customer is checked against the current time and deductions are made if the customer has completed his/her billing. After deductions, in case the queue is empty, which indirectly states that the counter is free, all the initial properties of the queue are reset.

In the case of a 1-person queue, the satisfaction level is reset to its initial value.

In all other cases, after making the deductions, the wait time of the last person in the queue is re-calculated which in-turn also re-calculates the customer satisfaction level. The sketch below would help elaborate on the same.



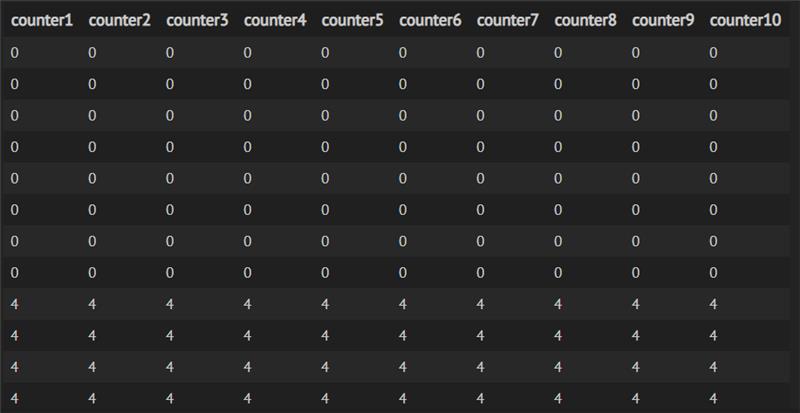
**Limitations of our study**

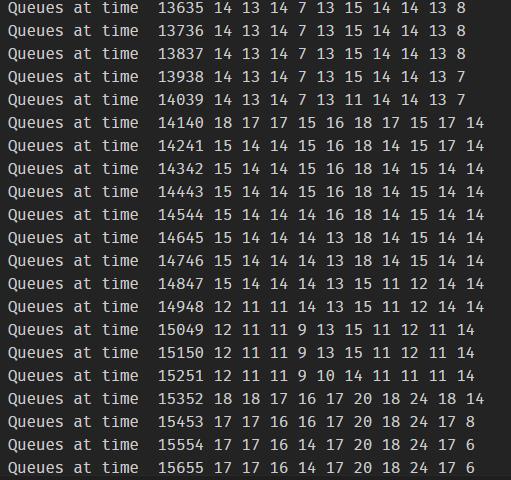
One of the key limitations of our study is that queue metrics are not dynamically adjusted owing to the added complexity of the process and edge cases. These metrics are only calculated while pushing or popping a consumer, so a lag may exist in some cases but is always on a higher end (making it simply more safer) rather than the risky lower end. Another aspect, which is more of an assumption that we needed to make was the equidistant placement of items in the store. But it is also important to note that our constants have been adjusted in a way to dampen any side-effects of these lag periods.

**Results**

Text

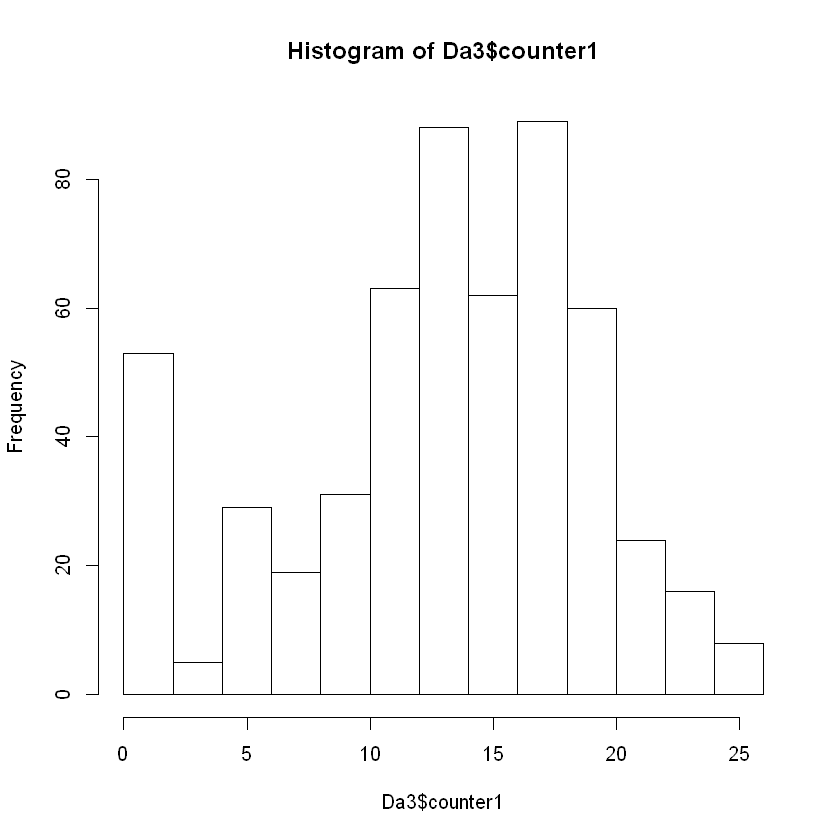
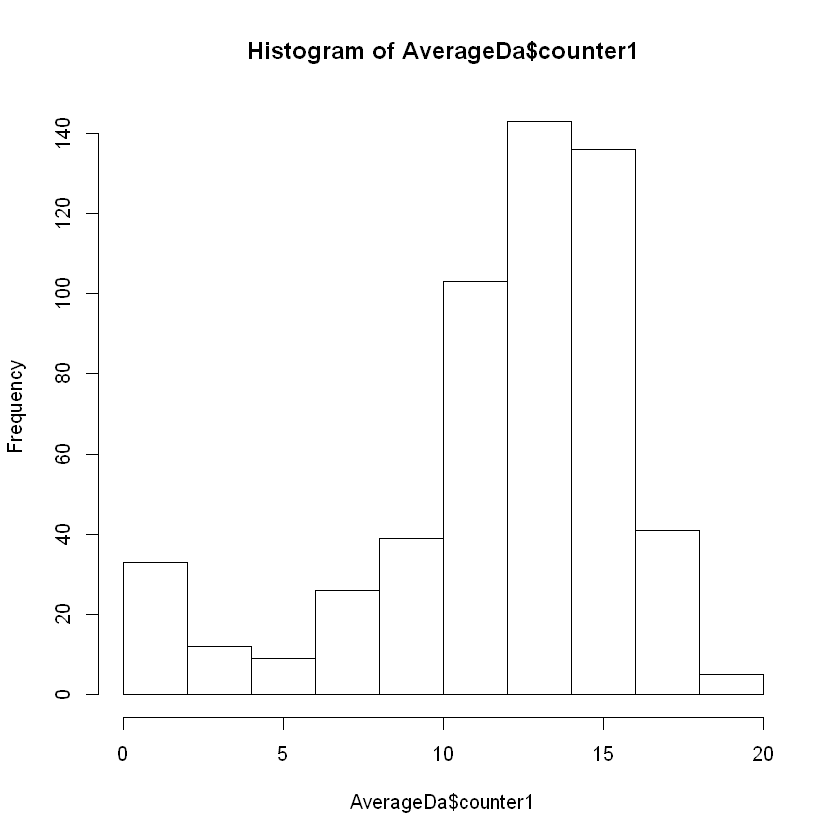
Description automatically generated



The first few minutes of the store rarely ever have any customers at counter 1, this is due to the early opening hours. Gradually the number of customers increase at the counter. The maximum number of people at this particular counter through the day are 18. We can also see that towards the end of the day the number of people at the counter begin decrease again. The middle of the day sees the most number of people

We can see times when the counter has no one joining the queue or if the number of people is too low then the store can decide to shut the counter at those times of the day.

As this is an average over a few days we can get an estimate of the queue density at various times of the day.

**Conclusion**

Using the store simulator ,we can run the store hours for various days and with various distributions and store size by tuning the parameter of the simulation. After simulating the shopping and billing processes for several days we can try finding patterns in the mean number of people at a particular counter at various times of the day and then decide based on the operation costs and density of the queue to then control the number of operatable counters. This way we can save cost for the store and maintain the customer satisfaction levels of the store as well.

**Contribution Statement:**

Equal contributions by all members except for Devansh.

**Appendix**

A run of this code essentially highlights a store’s shopping behavior. Just like any real grocery store, the crowd at the queues follow a normal distribution.

**Dataframe creation for storing store activity for 5 different days.**

Da1 <- data.frame('counter1'= numeric(3),'counter2'= numeric(3)

,'counter3'= numeric(3),'counter4'= numeric(3)

,'counter5'= numeric(3),'counter6'= numeric(3)

,'counter7'= numeric(3),'counter8'= numeric(3)

,'counter9'= numeric(3),'counter10'= numeric(3))

Da2 <- data.frame('counter1'= numeric(3),'counter2'= numeric(3)

,'counter3'= numeric(3),'counter4'= numeric(3)

,'counter5'= numeric(3),'counter6'= numeric(3)

,'counter7'= numeric(3),'counter8'= numeric(3)

,'counter9'= numeric(3),'counter10'= numeric(3))

Da3 <- data.frame('counter1'= numeric(3),'counter2'= numeric(3)

,'counter3'= numeric(3),'counter4'= numeric(3)

,'counter5'= numeric(3),'counter6'= numeric(3)

,'counter7'= numeric(3),'counter8'= numeric(3)

,'counter9'= numeric(3),'counter10'= numeric(3))

Da4 <- data.frame('counter1'= numeric(3),'counter2'= numeric(3)

,'counter3'= numeric(3),'counter4'= numeric(3)

,'counter5'= numeric(3),'counter6'= numeric(3)

,'counter7'= numeric(3),'counter8'= numeric(3)

,'counter9'= numeric(3),'counter10'= numeric(3))

Da5 <- data.frame('counter1'= numeric(3),'counter2'= numeric(3)

,'counter3'= numeric(3),'counter4'= numeric(3)

,'counter5'= numeric(3),'counter6'= numeric(3)

,'counter7'= numeric(3),'counter8'= numeric(3)

,'counter9'= numeric(3),'counter10'= numeric(3))

#insert Row in the Dataframe

insertRow <- function(existingDF, newrow, r) {

existingDF[seq(r+1,nrow(existingDF)+1),] <- existingDF[seq(r,nrow(existingDF)),]

existingDF[r,] <- newrow

existingDF

}

## Function Definitions

a=1

norm\_dist <- function(n, m, s, lwr, upr, nnorm) {

samp <- rnorm(nnorm, m, s)

samp <- samp[samp >= lwr & samp <= upr]

if (length(samp) >= n) {

return(sample(samp, n))

}

stop(simpleError("Not enough values to sample from. Try increasing nnorm."))

}

#tolerance

tolerance<-function(m,sd,x=1,low=0,up=1000){

return (norm\_dist(x,m,sd,low,up,x\*5))

}

#picking\_rate

picking\_rate <- function(m,sd,x=1,low=0,up=1000){

return (norm\_dist(x,m,sd,low,up,x\*5))

}

# function to calculate wait time

waitTime = function(num\_queue, cum\_cart, packing\_t){

y = (cum\_cart \* packing\_t) + (num\_queue \* 5) # wait time equation

return(y)

}

# function to calculate customer satisfaction of last person in each queue

customerSatisfaction = function(wait\_time, cust\_tol){

wait\_time\_c = 0.015

tol\_constant = 0.3

y = 100 - (wait\_time\*wait\_time\_c + tol\_constant\*(100 - cust\_tol)) # customer satisfaction till the person reaches the counter

return(y)

}

# function to calculate the exit time for a customer once he joins the queue

exitTime = function(join\_queue\_time, wait\_time = 0, cartSize, packing\_t){

y = join\_queue\_time + wait\_time + (cartSize \* packing\_t) + 5

return(y)

}

##Customer Initialization

c=0

setClass('cust',representation(name='numeric',enter\_time='numeric',picking\_time='numeric',tolerance='numeric',picking\_rate='numeric',cart\_size='numeric',status='numeric',shopping\_time='numeric',exit\_time='numeric',queue\_number='numeric'))

initialization<- function(n,m\_tol,sd\_tol,m\_picking,sd\_picking,max\_cart,time, exit\_time1 = -1, queue\_number1 = -1,name1=-1){

vec\_people<-vector()

for (i in 1:n){

tolerance1<- tolerance(m\_tol,sd\_tol)

picking\_rate1<-picking\_rate(m\_picking,sd\_picking)

cart\_size<-sample(1:max\_cart,1)

picking\_time=picking\_rate1\*cart\_size

shopping\_time=time+picking\_time

person1<-new('cust',enter\_time=time,tolerance=tolerance1,picking\_rate=picking\_rate1,

cart\_size=cart\_size,status=1,

shopping\_time=shopping\_time,

exit\_time=exit\_time1,

queue\_number = queue\_number1,

picking\_time=picking\_time,

name=name1

)

#picking\_time is time taken to pick the items

#shopping\_time is enter\_time + picking\_time

vec\_people<- append(vec\_people,person1)

}

return (vec\_people)

}

## Variable Declaration

m\_tol=100

sd\_tol=0.5

m\_picking=40

sd\_picking=4

max\_cart=40

store\_close\_time=54000 #15 hours

time\_counter=299 #for group 2

cust\_time\_m<-c(5,7,9,11,13,15,17,20,17,15,13,11,9,7,5)

cust\_time\_sd<-c(2,2,3,3,3,4,4,5,5,3,3,3,2,2,2,2)

names(cust\_time\_m)<-c(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14)

names(cust\_time\_sd)<-c(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14)

go\_to\_queue=c()

go\_to\_queue2=c()

cust\_in=c()

cust\_in2=c()

counterT = 0

# Optimization initializations

packing\_m = 5

packing\_sd = 0.5

q\_packing\_rates=abs(rnorm(10,packing\_m,packing\_sd))

q\_data = list(c(),c(),c(),c(),c(),c(),c(),c(),c(),c())

q\_s = c(100,100,100,100,100,100,100,100,100,100)

q\_num\_ppl = c(0,0,0,0,0,0,0,0,0,0)

q\_cum\_carts = c(0,0,0,0,0,0,0,0,0,0)

for(Time in 1:55000)#54000seconds

{

if(Time<=51000){

time\_counter=time\_counter+1

if (time\_counter==1200){

time\_counter<-0

num\_of\_cust<-abs(as.integer(rnorm(1,cust\_time\_m[toString(Time%/%3600)],

cust\_time\_sd[toString(Time%/%3600)])))

cust\_in=initialization(num\_of\_cust,m\_tol=m\_tol,sd\_tol=sd\_tol,m\_picking=m\_picking,

sd\_picking=sd\_picking,max\_cart=max\_cart,time=Time,exit\_time1=-1,queue\_number1=-1)

#Vector of Customers entering the store

#New customers coming in every 5 mins

cust\_in2<-append(cust\_in2,cust\_in)

}

}

remove=c()

for (customer in 1:length(cust\_in2)){

if(length(cust\_in2)>1){

if(cust\_in2[[customer]]@shopping\_time>=Time){

#shopping\_time = enter\_time+picking\_rate\*cart\_size

cust\_in2[[customer]]@status=2

cust\_in2[[customer]]@tolerance=cust\_in2[[customer]]@tolerance-0.0085\*cust\_in2[[customer]]@picking\_time

go\_to\_queue=append(go\_to\_queue,cust\_in2[[customer]])#go\_to\_queue is for group 3

remove=append(remove,customer)

go\_to\_queue2 = append(go\_to\_queue2, go\_to\_queue)

}

}

}

if( length(remove)>1){

if(length(cust\_in2)>1){

cust\_in2=cust\_in2[-(remove)]

}

}

# This is inside the main time loop

# Popping customers who are finished billing before adding incoming customers

for(i in 1:10){

if(q\_num\_ppl[i]==0){

next

} else {

x = q\_data[[i]][[1]]

if(x@exit\_time <= Time){

q\_data[[i]] = q\_data[[i]][2:q\_num\_ppl[[i]]]# Popping the first customer from the queue

#cat("person removed from queue ",x@queue\_number," at time ",Time)

#cat("\n")

q\_num\_ppl[i] = q\_num\_ppl[i]-1

q\_cum\_carts[i] = q\_cum\_carts[i] - x@cart\_size

if(q\_num\_ppl[i]<=1){

q\_s[i] = 100

}else{

wt = waitTime(num\_queue = q\_num\_ppl[i],

cum\_cart = q\_cum\_carts[i],

packing\_t = q\_packing\_rates[i])

q\_s[i] = customerSatisfaction(wt,q\_data[[i]][[q\_num\_ppl[i]]]@tolerance)

}

}

}

}

#cat(length(go\_to\_queue),' ')

current\_cust = go\_to\_queue

# Assuming current\_cust contains incoming customers at that particular time 't'

if(class(current\_cust)!="NULL"){

for(i in 1:length(current\_cust)){

for(j in 1:10){

if(q\_s[j]>=70){

if (class(current\_cust[[i]])=='NULL'){

next

}

q\_data[[j]] = c(q\_data[[j]],current\_cust[[i]]) # Pushing a customer at the end of the queue

#cat("person added to queue ",j," at time ",Time)

#cat("\n")

q\_num\_ppl[j] = q\_num\_ppl[j] + 1

q\_cum\_carts[j] = q\_cum\_carts[j] + current\_cust[[i]]@cart\_size

current\_cust[[i]]@queue\_number = j

q\_data[[j]][[q\_num\_ppl[j]]]@queue\_number = j

if(q\_num\_ppl[j]==1){

q\_s[j] = 100

current\_cust[[i]]@exit\_time = exitTime(join\_queue\_time = current\_cust[[i]]@shopping\_time,

cartSize = current\_cust[[i]]@cart\_size,

packing\_t = q\_packing\_rates[j])

q\_data[[j]][[q\_num\_ppl[j]]]@exit\_time = exitTime(join\_queue\_time = current\_cust[[i]]@shopping\_time,

cartSize = current\_cust[[i]]@cart\_size,

packing\_t = q\_packing\_rates[j])

}else{

wt = waitTime(num\_queue = q\_num\_ppl[j],

cum\_cart = q\_cum\_carts[j],

packing\_t = q\_packing\_rates[j])

q\_s[j] = customerSatisfaction(wt,current\_cust[[i]]@tolerance)

current\_cust[[i]]@exit\_time =exitTime(join\_queue\_time = current\_cust[[i]]@shopping\_time,

wait\_time = wt,

cartSize = current\_cust[[i]]@cart\_size,

packing\_t = q\_packing\_rates[j])

q\_data[[j]][[q\_num\_ppl[j]]]@exit\_time = exitTime(join\_queue\_time = current\_cust[[i]]@shopping\_time,

wait\_time = wt,

cartSize = current\_cust[[i]]@cart\_size,

packing\_t = q\_packing\_rates[j])

}

}

}

}

}

temp2=c()

for( n in 1:10){

for (m in 1:length(q\_data[[n]])){

if(class(q\_data[[n]][[m]])=='NULL'){

temp2=c(temp2,m)

}

q\_data[[n]][-m]

}}

for (hnm in length(current\_cust)){

if( class(current\_cust[[hnm]])=='NULL'){

current\_cust[-hnm]

}

}

if(counterT<100){

counterT = counterT+1

} else {

cat("Queues at time ", Time, c(q\_num\_ppl[1],q\_num\_ppl[2],q\_num\_ppl[3]

,q\_num\_ppl[4],q\_num\_ppl[5],q\_num\_ppl[6],q\_num\_ppl[7]

,q\_num\_ppl[8],q\_num\_ppl[9],q\_num\_ppl[10]))

new\_v<-c(q\_num\_ppl[1],q\_num\_ppl[2],q\_num\_ppl[3]

,q\_num\_ppl[4],q\_num\_ppl[5],q\_num\_ppl[6],q\_num\_ppl[7]

,q\_num\_ppl[8],q\_num\_ppl[9],q\_num\_ppl[10])

Da5 <- insertRow(Da5,new\_v, a)

a= a+1

cat("\n")

counterT = 0

# show the number of people at any counter at time 't' / checking counter status

}

# Updations

go\_to\_queue = c()

}

#-------------------------------------------------------------------------------------------------------------

library(data.table)

rbindlist(list(Da1,Da2,Da3,Da4,Da5))[,lapply(.SD,mean),list()]

ALL <- Da1+Da2+Da3+Da4+Da5

AverageDa <- ALL/5