Project Report

Junzhu Xiang, Haomin Lin

Introduction:

This project is to conduct a discrete time simulation of a food court. In this simulation, this food court is divided to four types of stations: tray station, food station, beverage station and check station. The tray station is connecting to the generator of the simulation system, which means the tray station can receive new customers entering this food court. The food station and beverage station just like the queueing station, so customers have to wait in these two stations and receive service then go to next station. The check station connects to the exit of this system, so after the check station, customers can exit or go back to the food station. The inter-arrival time for each customer and the service time of each station follow an exponential distribution. So it is possible that some customers have to wait in line before they receive the service of this station. Thus, the simulation system is constructed by the structure of discrete time events and a simulation engine. The goal of this study is to obtain the time data about waiting or staying in the system of customers and then analyze these data to recommend suggestions to make the system go smoothly.

Implementation:

To implement this discrete time event simulation system, firstly we create a linked list which is a FIFO queue(Future Event List, FEL) to simulate the system. The priority of this FIFO queue is the timestamp of each event. For each event, it has an event type (Arrival or Departure) on it. To start with this program, we should know the number of generators of this system according to the input file. Let's assume there are two generators, so we should add two new customers to the two stations which connects to these two generators to start this simulation. This adding process is that we add these two customers to the FEL of the system according to their timestamp. In the FEL, an arrival event is going to be processed, if currently this station is empty, we should immediately arrange its departure event with a timestamp and add this event to the FEL, but if this station is not empty, we should add this customer to the waiting queue of this station. After the customer next to him is processed, we can arrange his departure event. If the current station connects to the generator, the tray station, we should arrange a new customer to enter the system if this current event type is arrival. After processing a customer event, this customer event has a probability to go to the next station, but it depends on his current station type. For example, if he is in food station, he has to go the beverage station. But if he is in check station, he can go to the food station as well as exit the system.

In our program, the engine.c file contains functions that construct the engine of FEL, which can process both arrival and departure event and schedule new event to FEL. The initialize.c file contains functions that create and initialize all types of linked lists, arrays and variables which will be used in the application. The application.c file contains functions that construct and simulate the discrete time event system by using the engine. To process an event, there are some important functions that we will use. 1) *void Arrival(CusInfo *customer)*: this function is used to process an arrival type event, if the station is not empty, we should call *void addTail(CusInfo *customer, CusInfo *pumpHead)* to add it to the waiting queue. 2) *void Departure(CusInfo *customer):*this function is used to process a departure type event. After this departure event is processed, we should call *CusInfo *removePump(CusInfo *pumpHead)* to extract the next customer's info and arrange its departure event. Then we should call *void Inject(CusInfo *customer)* and *int selectStation(int num, double *prob)* to inject the last customer to the next specific station. If the customer has to exit according to the probability, we should call void *InjectToExit(CusInfo **customer*)* to let the customer exit the system. Meanwhile, there are some helper functions like helping read and write data, generating the random inter-arrival and service time, and computing the waiting time of customers in

this program. **Important notes**: Before running the simulation, we set the arrival time for the first customer entering the system is 10. You can change it to what you like in the main() function of application.c file.

Testing

First, we design an input files which contains the info for each station. It looks like this:

```
11
0 G 1 1
1 Q 0.1 4 0.2 0.4 0.2 0.2 2 3 4 5
2 Q 3.2 2 0.5 0.5 6 7
3 Q 1.7 2 0.5 0.5 6 7
4 Q 3.3 2 0.5 0.5 6 7
5 Q 2.8 2 0.5 0.5 6 7
6 Q 0.5 2 0.5 0.5 8 9
7 Q 0.5 2 0.5 0.5 8 9
8 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
9 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
10 E
```

So as you can see, there are 11 stations in this system. They are 1 generator, 9 queueing stations and 1 exit. So after we run this program, we can get:

```
Number of queueing stations: 9
Number of generators: 1
Number of exits: 1
```

In our program, we set the timestamp of the first customer of this system is 10. If there are two generators, we should insert two customers to the system. So, after we insert one ,we can get the initial FEL, and it's no need for him to wait in the Station 1(QID = 1), so it should be processed immediately:

```
Initial event list:
Event List: 10.000000(EventType: ARRIVAL, QID:1)
Processing Arrival Event at time: At 10.000000, Station: 1, 0 customers waiting
```

When the system starts to run, in the terminal, each time one customer is going to be processed, we print the FEL and related info like this:

```
Event List: 11.176854(EventType: ARRIVAL, QID:4) 11.229486(EventType: DEPARTURE,
 QID:1) 11.235250(EventType: ARRIVAL, QID:1) 11.976492(EventType: DEPARTURE, QID
:3) 15.202349(EventType: DEPARTURE, QID:4) 16.311532(EventType: DEPARTURE, QID:2
) 19.317598(EventType: DEPARTURE, QID:5)
Processing Arrival Event at time: At 11.176854, Station: 4, 6 customers waiting
```

As you can see, the event list is the FEL list, it contains all future events which will happen in the future and their event type and the station ID (QID). The command line below indicates that it's time now we should process this event with a specific timestamp, and in this station 4, there are 6 customers waiting.

If we set the end time to 100, we can see if the timestamp of first customer in the FEL exceeds to 100, the simulation should stop to run, and print out some useful information.

```
Event List: 100.002766(EventType: ARRIVAL, QID:1) 100.343755(Eve
TURE, QID:2) 101.353999(EventType: DEPARTURE, QID:8) 101.698304(
PARTURE, QID:4)

The number of customers entering the system: 903
The number of customers exiting the system: 120
```

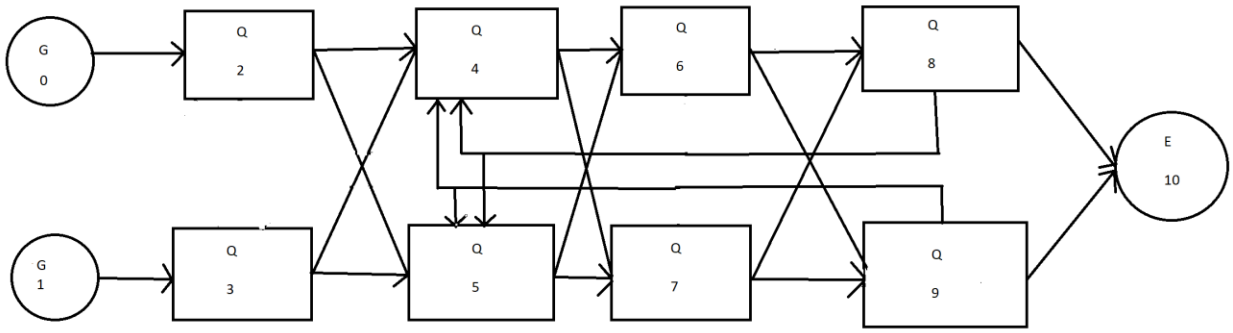
We can see during the time of 100, there are 903 customers entering the system, and 120 customers exiting the system.

```
The minimum/maximum/average time of one customer in the system:
0.906606
80.130704
37.715401
The minimum/maximum/average time of one customer waiting in queues:
0.006251
82.010430
37.378913

The average waiting time experienced by customers at each station:
Queueing Station ID: 1, Time: 1.032515
Queueing Station ID: 2, Time: 28.468616
Queueing Station ID: 3, Time: 33.776402
Queueing Station ID: 4, Time: 32.594467
Queueing Station ID: 5, Time: 32.841488
Queueing Station ID: 6, Time: 0.228841
Queueing Station ID: 7, Time: 0.175074
Queueing Station ID: 8, Time: 0.727874
Queueing Station ID: 9, Time: 0.531328
```

The picture above shows the time about customers in the system. Owing to the inter arrival time and service time is randomly, we cannot precisely test the correctness of our program. So what we do is to fix the inter arrival time and service time of each station:

```
11
0 G 0.5 2
1 G 0.5 3

2 Q 0.5 2 0.5 0.5 4 5
3 Q 0.5 2 0.5 0.5 4 5

4 Q 3 2 0.5 0.5 6 7
5 Q 3 2 0.5 0.5 6 7

6 Q 1 2 0.5 0.5 8 9
7 Q 1 2 0.5 0.5 8 9

8 Q 1.5 3 0.025 0.025 0.95 4 5 10
9 Q 1.5 3 0.025 0.025 0.95 4 5 10

10 E
```

As you can see the picture above, this is testing file we designed. There are 2 generators connecting queueing station 2 and 3. There are 6 queueing stations. And Station 8 and 9 connect to Exit station. So in our program, we get rid of the *randExp(double P)* function and let the inter arrival time for each customer is 0.5, as well as the service time is fixed.



So the whole system we designed should be like this. The fixed service time for Q2 and Q3 is 0.5, for Q4 and Q5 is 3, for Q6 and Q7 is 1, and for Q8 and Q9 is 1.5. Therefore, the average service receiving time for each customer who exit the system should be 6 (0.5+3+1+1.5 = 6), and during the time of 100, the system should roughly generate 360 + 2 customers if the first two customers both arrive at 10. And in this test procedure, to calculate the waiting time for one customer, we consider only customers exiting the system. For the time of one customer remaining in the system, we consider customers exiting the system as well.

```
The number of customers entering the system: 362
The number of customers exiting the system: 55

The minimum/maximum/average time of one customer in the system:
6.000000
74.500000
40.754545
The minimum/maximum/average time of one customer waiting in queues:
0.000000
68.500000
34.754545

The average waiting time experienced by customers at each station:
Queueing Station ID: 2, Time: 0.000812
Queueing Station ID: 3, Time: 0.007537
Queueing Station ID: 4, Time: 35.717410
Queueing Station ID: 5, Time: 35.781343
Queueing Station ID: 6, Time: 0.197635
Queueing Station ID: 7, Time: 0.018859
Queueing Station ID: 8, Time: 0.088866
Queueing Station ID: 9, Time: 0.411108
```

According to the results, we can see the number of customers entering the system is 362, the min time of one customer staying in the system is 6, which the sum of service time of all type stations. The min time of one customer waiting in queues is 0 (maybe the first customer of this system). If we use the average time of staying in system (40.754545) minus the average time of waiting in queues (34.754545), we can get the average service time for each customer who exits the system is 6. And the difference between the two max times is also 6. Meanwhile, the service time of Station 2 and 3 is 0.5, which equals to the inter arrival time. So in theory, the waiting time for these two stations should be 0. Here, in practice, the waiting time of both approaches zero (0.000812, 0.007537). Therefore, these evidences are enough to verify the correctness of our program.

Error Detection

To maintain the functionality of our program and avoid errors caused by illegal configuration files, we set the following function to detect the errors and terminal our program in advance:

1. Total station number conflicts : When we use an illegal configuration file which has conflicting total number like this:

```
20
0 G 2 1
1 Q 0.1 4 0.2 0.4 0.2 0.2 2 3 4 5
2 Q 3.2 2 0.5 0.5 6 7
3 Q 1.7 2 0.5 0.5 6 7
4 Q 3.3 2 0.5 0.5 6 7
5 Q 2.8 2 0.5 0.5 6 7
6 Q 0.5 2 0.5 0.5 8 9
7 Q 0.5 2 0.5 0.5 8 9
8 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
9 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
10 E
```

We will get error message:

```
Error 1: there should be 20 components, but only 1 Generators, 9 Queues,
 1 Exits, which sums up to 11
```

2. Total possibility conflicts: When we use an illegal configuration file with probability not totaling 1 like this:

```
2 Q 3.2 2 0.5 0.6 6 7
3 Q 1.7 2 0.5 0.5 6 7
```

We will get error message:

```
Error 2: #2 Station's total connecting probability is wrong, it should a
dds up to 1
```

3. Incomplete information of a station: When we use an illegal configuration file with missing information like this:

```
0 G 2
1 Q 0.1 4 0.2 0.4 0.2 0.2 2 3 4 5
2 Q 3.2 2 0.5 0.5 6 7
```

We will get error message:

```
Error 3: Incomplete information for components
```

4. Illegal possibility values: When we use an illegal configuration file with illegal possibility like this:

```
5 Q 2.8 2 0.5 0 6 7
6 Q 0.5 2 0.5 0.5 8 9
```

We will get error message:

```
Error 4: #5 Station's #1 connection probability is illegal, it should be
 between (0, 1], but it's now 0.000000
```

5. Missing one kind of vital stations: When we use an illegal configuration file with illegal possibility like this (there's no exits):

```
8 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
9 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
```

We will get error message:

```
Error 5: Missing Exits!
```

6. Illegal information of stations (same station number): When we use an illegal configuration file with duplicate station number like this:

```
2 Q 3.2 2 0.5 0.5 6 7
2 Q 1.7 2 0.5 0.5 6 7
```

We will get error message:

```
Error 6: #2 Queue's number has duplicate!
```

7. Illegal information of stations (connect to itself): When we use an illegal configuration file with station connecting to itself like this:

```
6 Q 0.5 2 0.5 0.5 8 6
7 Q 0.5 2 0.5 0.5 8 9
```

We will get error message:

```
Error 7: #6 Queue connects to itself!
```

8. Illegal information of stations (number not in order)

According to the assignment, we should have “component ID (0, 1, ... N-1)”, so when we use an illegal configuration file with station numbers not in order like this:

```
6 Q 2.8 2 0.5 0.5 6 7
5 Q 0.5 2 0.5 0.5 8 9
```

We will get error message:

```
Error 8: Queue at line#5 has number out of order
```

9. Illegal information of stations (exceeding limit): As the biggest ID number should be N-1, when we use an illegal file with ID exceeding the limit like this:

```
8 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
12 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
10 E
```

We will get error message:

```
Error 10: ID at line #9 exceeds the limit
```

10. Missing total number: When we use an illegal configuration file without total number like this:

```
0 G 2 1
1 Q 0.1 4 0.2 0.4 0.2 0.2 2 3 4 5
2 Q 3.2 2 0.5 0.5 6 7
3 Q 1.7 2 0.5 0.5 6 7
4 Q 3.3 2 0.5 0.5 6 7
```

We will get error message:

```
Error 9: Missing total number!
```

Experiment Report

After proving the functionality of our program and resolve memory leak issues, we run the program to simulate a food court according to the assignment. In our configuration file, we define 1 generator, 4 food

stations, 2 beverage stations and several checkout stations. The details are presented as follows, in which there are 2 checkout stations:

```
12
0 G 1 1
1 Q 0.1 4 0.2 0.4 0.2 0.2 2 3 4 5
2 Q 3.2 2 0.5 0.5 6 7
3 Q 1.7 2 0.5 0.5 6 7
4 Q 3.3 2 0.5 0.5 6 7
5 Q 2.8 2 0.5 0.5 6 7
6 Q 0.5 2 0.5 0.5 8 9
7 Q 0.5 2 0.5 0.5 8 9
8 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
9 Q 1.0 5 0.01 0.02 0.01 0.01 0.95 2 3 4 5 10
10 E
```

To test the influence of changing the number of checkout stations, we run the experiments with checkout station number set as 1, 2, 3, 4, 5 (to maintain the property that all probabilities sum up to 1, we set the probabilities as 0.333, 0.333, 0.334), with interarrival time varying as 0.5min, 1min, 2min, 3min, 5min. And we run the simulation for 240 min according to the requirement.

The results are shown as follows:

Notes:

All_Average: The average amount of time a customer remains in the system among those customers who exited during the simulation run.

All_Min: The minimum amount of time a customer remains in the system among those customers who exited during the simulation run.

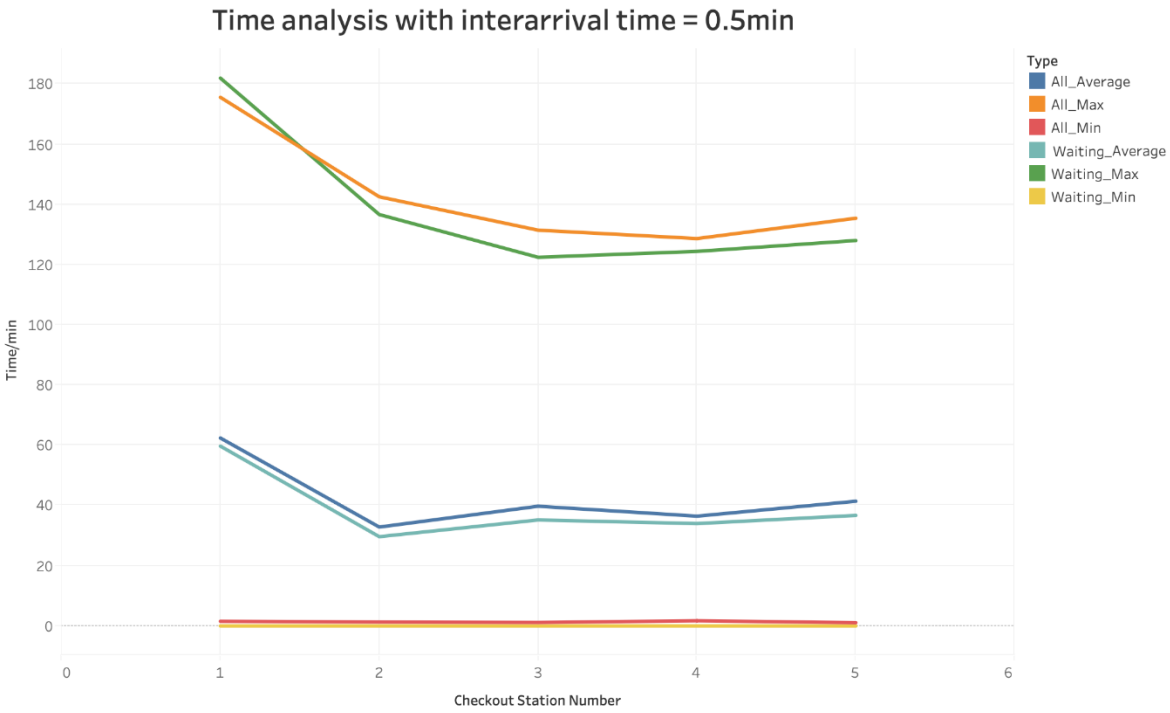
All_Max: The maximum amount of time a customer remains in the system among those customers who exited during the simulation run.

Waiting_Average: The average amount of the total time each customer must wait in queues in traveling through the system.

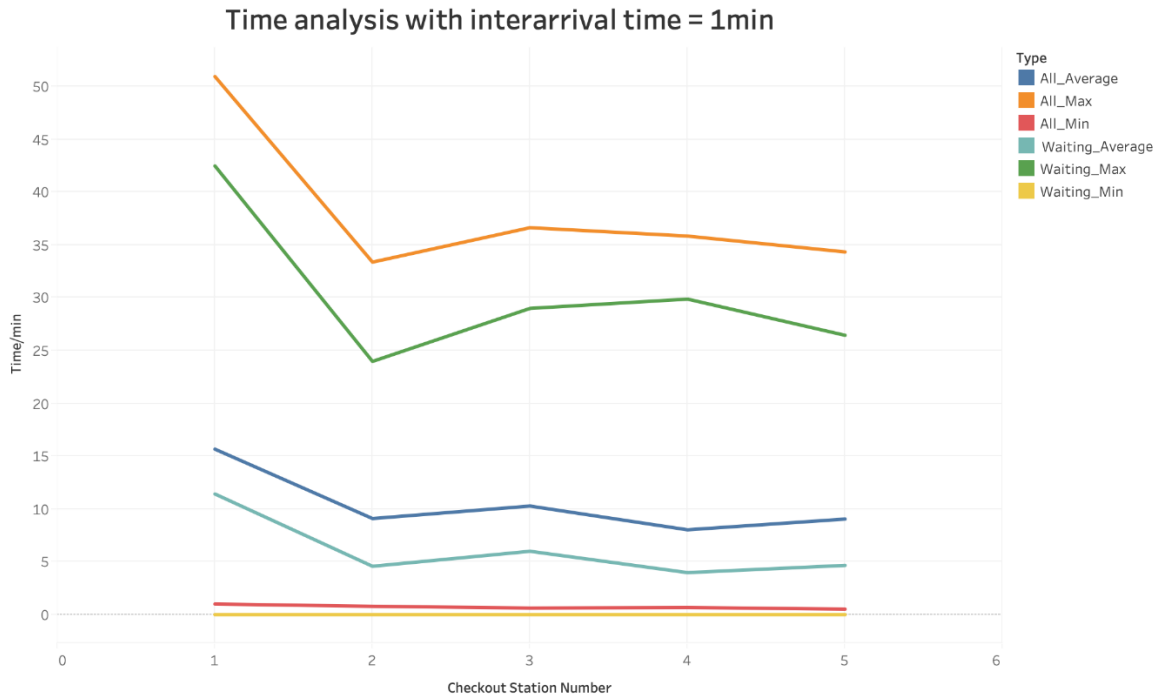
Waiting_Min: The minimum amount of the total time each customer must wait in queues in traveling through the system.

Waiting_Max: The maximum amount of the total time each customer must wait in queues in traveling through the system.

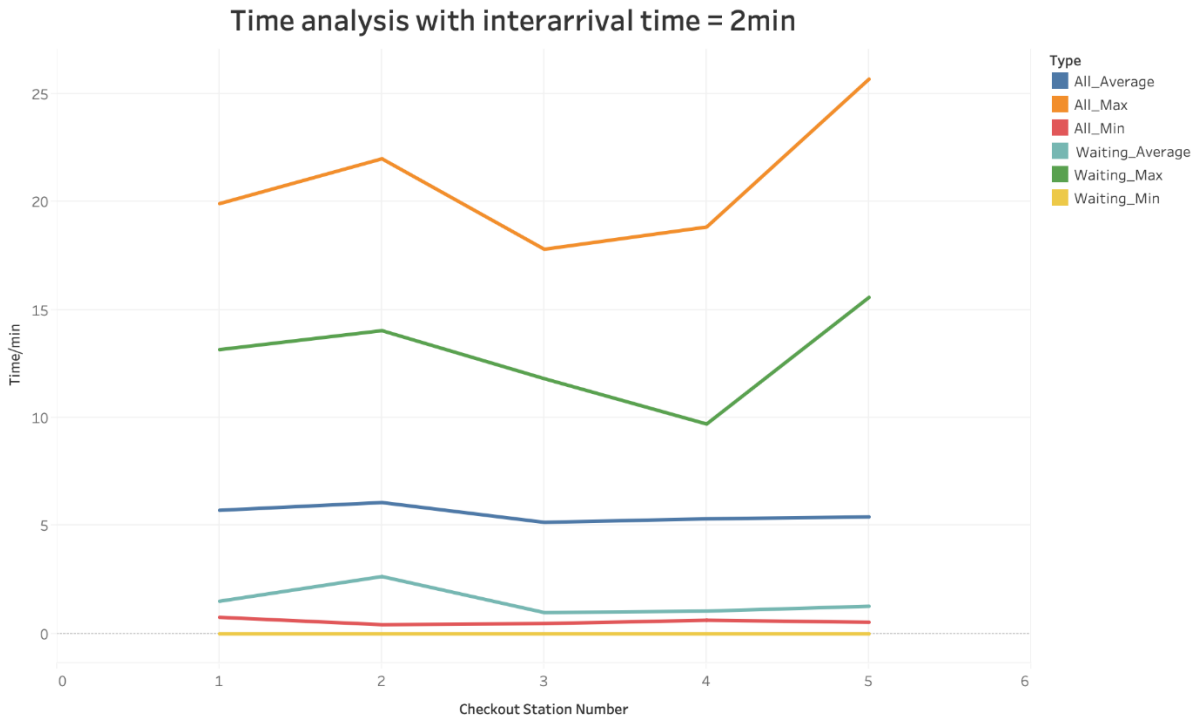
1. The interarrival time is 0.5min



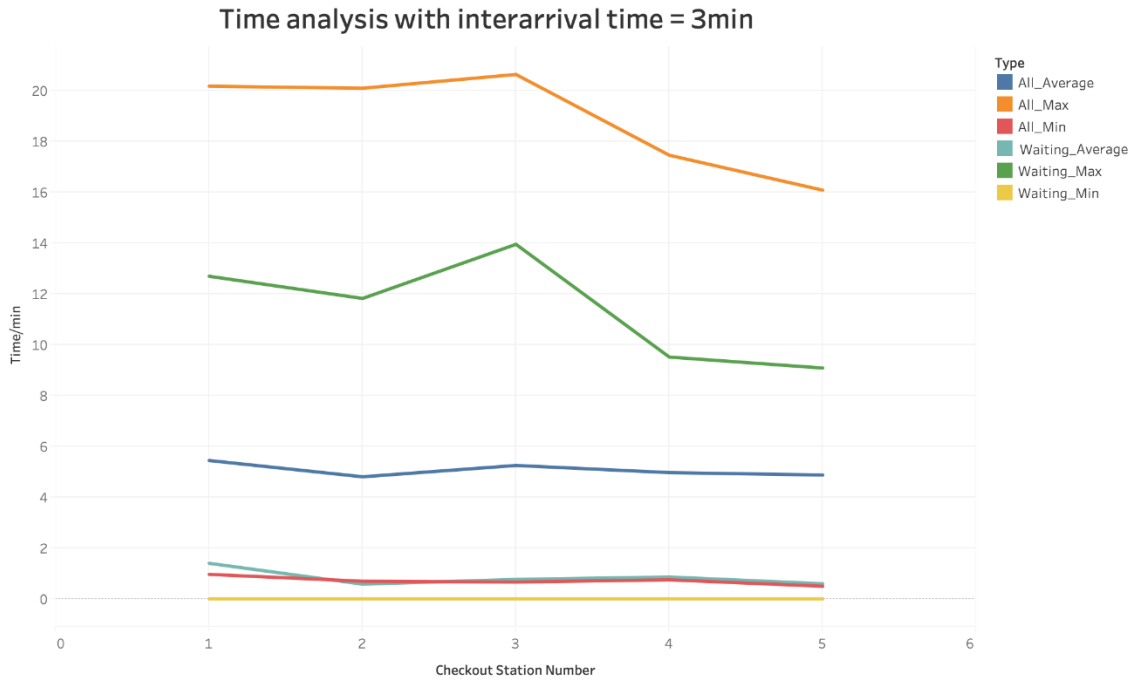
2. The interarrival time is 1min



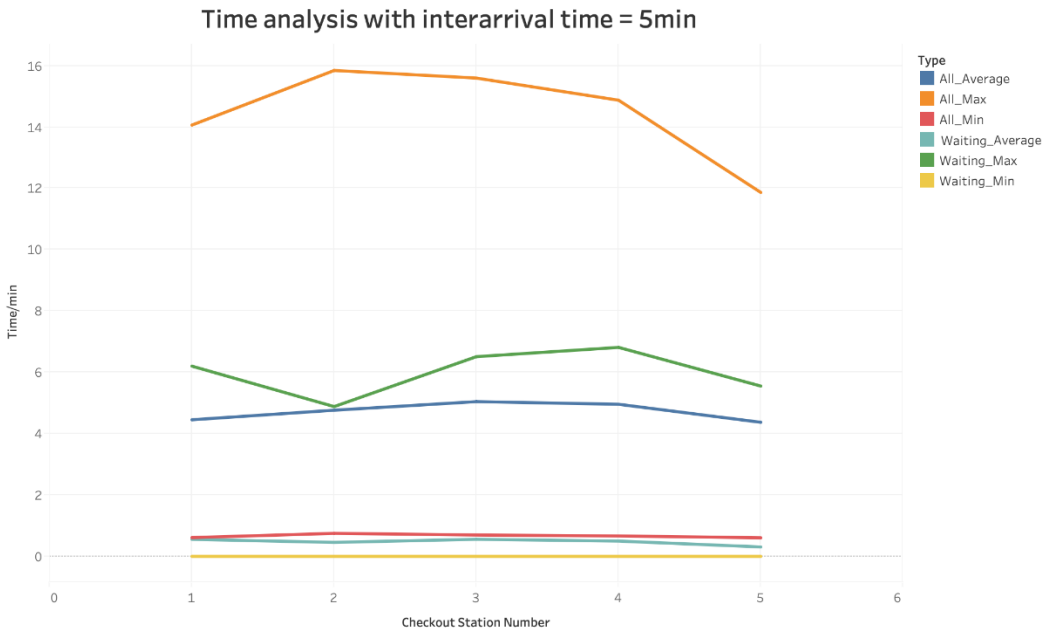
3. The interarrival time is 2min



4. The interarrival time is 3min



5. The interarrival time is 5min



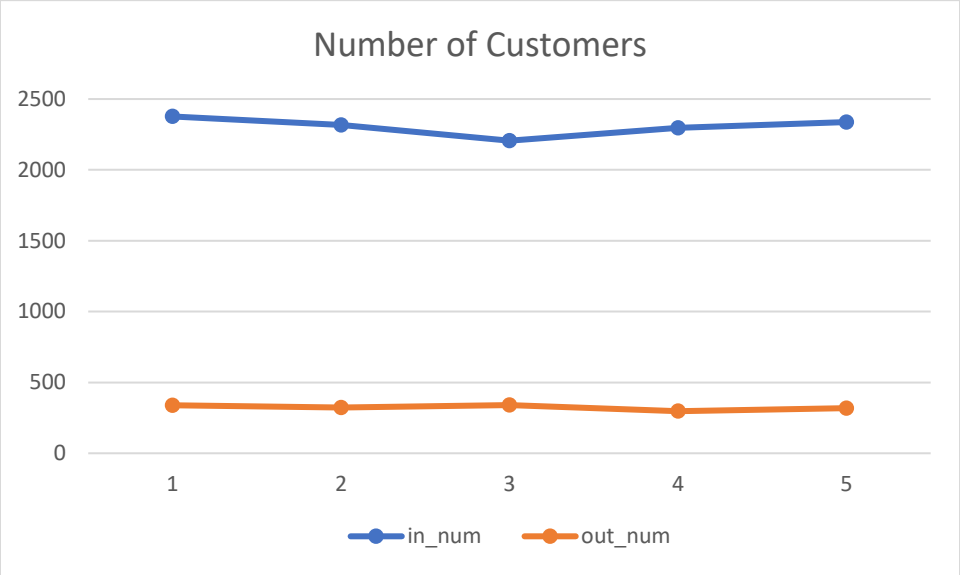
From the results, we can see that the maximum of waiting time or the staying time in the system for those who exit the system seems to be random, especially when the interarrival time exceeds 2min. Even when there are more checkout stations to serve the customers, it is still possible for the maximum time to raise prominently.

However, according to the average waiting time, when the interarrival time is small, the increasing of the number of checkout stations still makes a considerable difference. As the checkout station number increase from 1 to 5, the average waiting time reduces generally. While in the cases that the interarrival time is relatively large (for example, 2), the increasing of checkout stations has little impact. The average waiting time barely change as more stations are set. And obviously, the waiting time will drop as the interarrival time increases.

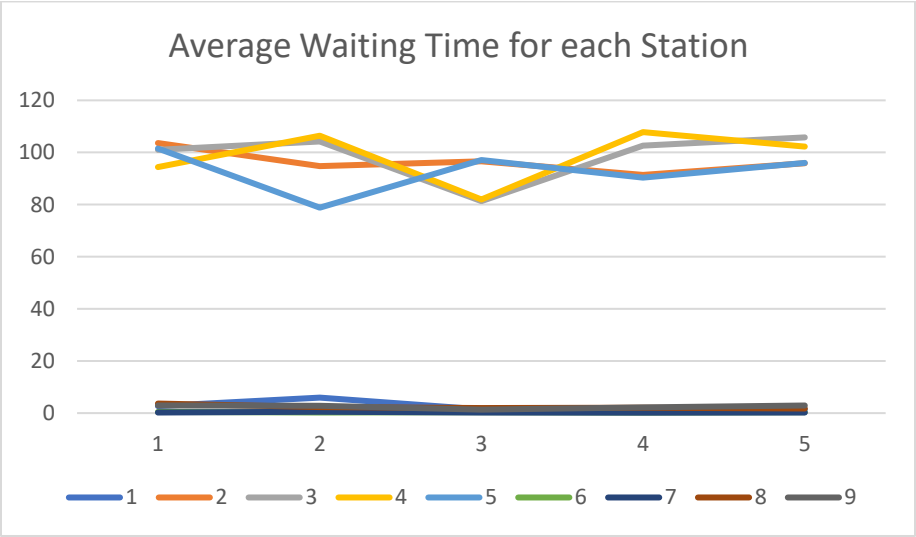
Also, as you can see in those figures, the max time for both waiting and staying is randomly distributed, no matter how the number of check stations changes. But the value of max time is still in a reasonable range. This is because the number of check stations does not influence the possibility of customers exiting the system or going back to the food station. And you can see the minimum waiting time is almost 0 in these 5 figures. To explain this, we can assume that it may result from the first customer entering the system. It is possible that the first customer doesn't have to wait in each station he goes through, because there is no customer in front of him!

From what we've observed, we can see that as the interarrival time increases, it means that in a real system, there are fewer customers. So it's reasonable to see that even there are more checkout stations, customers still can't expect to leave the system sooner. Since the number of food and beverage stations are fixed, if the load from customers can't be distributed by more food stations and more beverage stations, more checkout stations still can't help customers leave the system sooner. And for situation where interarrival time is less than 2min, 2 checkout stations can help the customers leave the system most efficiently.

Meanwhile, we use the original food court file to collect data after running the program 5 times. The end time is 240, and the start time is 10. After 5-time running, we get the number of customers entering the system and exiting the system. We plot the chart below and compute the average number of customers entering the system is 2307, which roughly equals to 2300. Theoretically, in_num should be 2300, because the inter-arrival time is 0.1, and the duration is 230. But it follows the exponential distribution. So the result is correct. The out_num is pretty small, because there are many customers still remaining in the system after the simulation.



And we collect the data about average waiting time for each station and plot the chart below. From station 1 to station 9, the time is: 2.6404948, 96.4470118, 99.008776, 98.5626978, 92.774675, 0.3346448, 0.2921218, 2.3442318, 2.4425638. As you can see from the data or the plot, the average waiting time for food station is very large compared to other stations. This is because the service time of food stations is much larger than those of other stations and customers who have sufficient money will go back to food stations. So if we want to reduce the congestion for stations, we should reduce the service of stations as more as possible or increase the number of stations that have more customers.



Collaboration

To complete this project, both of us has write a significant part of this program, and then we combine each part to test and run. Junzhu Xiang wrote parts such as handling the arrival event, injecting event and read data from file, Haomin Lin wrote parts such as handling the departure event and injecting events after check-out stations and write date to file. We discussed and wrote functions together such as how to compute time, how to create arrays and initialize them, how to free memories, etc.