Simulating a Queueing System in Python

Learn how to create a simulation in python from scratch

Bhavya Jain Jun 6, 2020 · 6 min read ★

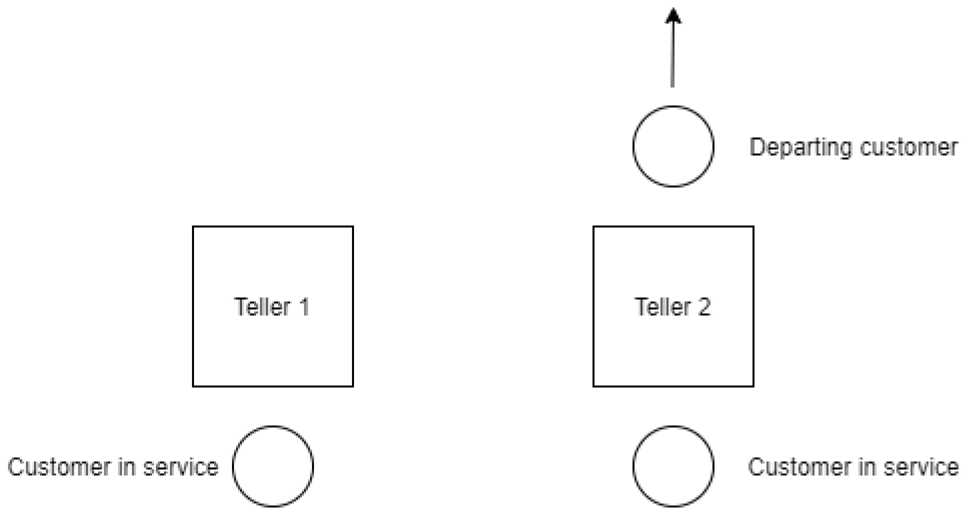
Image by Museums Victoria on Unsplash

We all have visited a bank at some point in our life, and we are familiar with how banks operate. Customers enter, wait in a queue for their number to be called out, get service from the teller, and finally leave. This is a queueing system, and we encounter many queueing systems in our day to day lives, from grocery stores to amusement parks they're everywhere. And that's why we must try and make them as efficient as possible. There is a lot of randomness involved in these systems, which can cause huge delays, result in long queues, reduce efficiency, and even monetary loss. The randomness can be addressed by developing a discrete event simulation model, this can be extremely helpful in improving the operational efficiency, by analyzing key performance measures.

In this project, I am going to be simulating a queueing system for a bank.

> *Let's consider a bank that has two tellers. Customers arrive at the bank about every 3 minutes on average according to a Poisson process. This rate of arrival is assumed in this case but should be modeled from actual data to get accurate results. They wait in a single line for an idle teller. This type of system is referred to as a M/M/2 queueing system. The average time it takes to serve a customer is 1.2 minutes by the first teller and 1.5 minutes by the second teller. The service times are assumed to be exponential here. When a customer enters the bank and both tellers are idle, they choose either one with equal probabilities. If a customer enters the bank and there are four people waiting in the line, they will leave the bank with probability 50%. If a customer enters the bank and there are five or more people waiting in the line, they will leave the bank with probability 60%.*

Lets first try and visualize the system



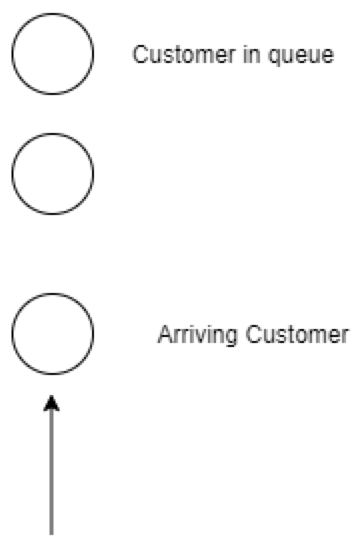


Photo by the author

Great! now let's start building the model

Generally, there are two widely used approaches to advance time in discrete event simulation, **next event time advance**(clock advances hopping from event to event)& **fixed-increment time advance**(clock advances in fixed time intervals). I will be using the next event time advance mechanism for this model.

The main components of a simulation model are:

1. **State Variables:** describe the system at a particular time
2. **Simulation Clock:** Keeps track of time
3. **Statistical Counters:** Variables for storing statistical info about performance parameters
4. **Initialization Routine:** A subprogram or class that initializes the model at time 0
5. **Timing Routine:** A subprogram or a class that determines the next event
6. **Event Routine:** A subprogram or a class that updates the system when a particular event occurs

Starting with importing the required libraries, only NumPy and Pandas are needed in this case (python has a simulation library called SimPy, a process-based simulation framework which can also be used)

```python
import numpy as np
import pandas as pd
```

Bank_sim01.py hosted with ♡ by GitHub view raw

Source https://gist.github.com/BhavyaJain08

Next, we start by defining variables and initializing them in the __*init*__ function, inside the main class. The variable defined below are state variables as well as statistical counters.

The key variables that tell us about the performance of the system are average wait time, utilization of servers, number of people waiting in line, and lost customers, some of which are directly calculated and others derived.

```python
class Bank_Simulation:
    def __init__(self):
        self.clock=0.0                      #simulation clock
        self.num_arrivals=0                 #total number of arrivals
        self.t_arrival=self.gen_int_arr()   #time of next arrival
        self.t_departure1=float('inf')      #departure time from server 1
        self.t_departure2=float('inf')      #departure time from server 2
        self.dep_sum1=0                     #Sum of service times by teller 1
        self.dep_sum2=0                     #Sum of service times by teller 2
        self.state_T1=0                     #current state of server1 (binary)
        self.state_T2=0                     #current state of server2 (binary)
        self.total_wait_time=0.0            #total wait time
        self.num_in_q=0                     #current number in queue
        self.number_in_queue=0              #customers who had to wait in line(counter)
        self.num_of_departures1=0           #number of customers served by teller 1
        self.num_of_departures2=0           #number of customers served by teller 2
        self.lost_customers=0               #customers who left without service
```

Bank_sim02.py hosted with ♡ by GitHub view raw

Source https://gist.github.com/BhavyaJain08

Timing Routine

The timing routine decides which event occurs next by comparing the scheduled time of events and advances the simulation clock to the respective event. Initially, the departure events are scheduled to occur at time infinity(since there are no customers), which guarantees that the first event will be an arrival event.

```python
    def time_adv(self):
        t_next_event=min(self.t_arrival,self.t_departure1,self.t_departure2)
        self.total_wait_time += (self.num_in_q*(t_next_event-self.clock))
        self.clock=t_next_event

        if self.t_arrival<self.t_departure1 and self.t_arrival<self.t_departure2:
            self.arrival()
        elif self.t_departure1<self.t_arrival and self.t_departure1<self.t_departure2:
            self.teller1()
        else:
            self.teller2()
```

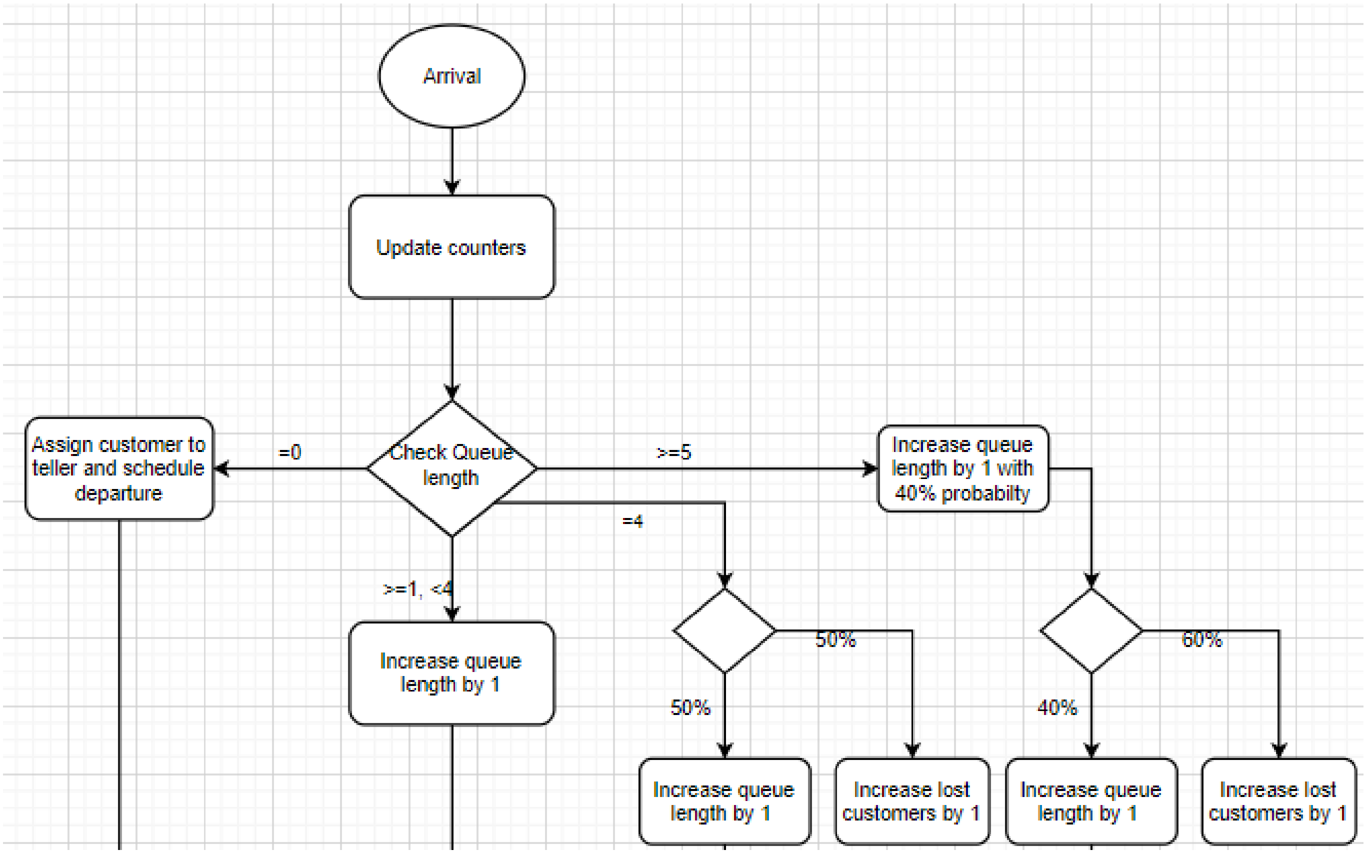
Bank_sim03.py hosted with ♡ by GitHub view raw

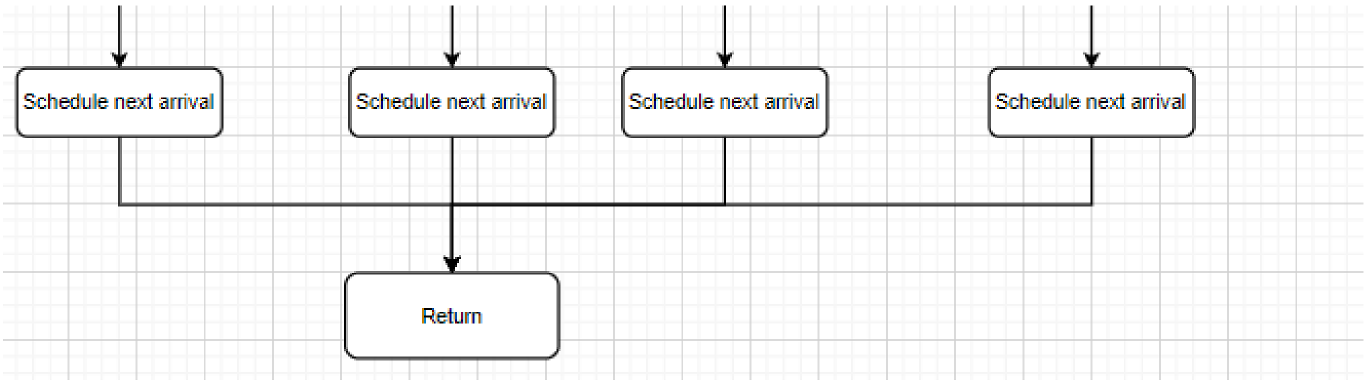
Source https://gist.github.com/BhavyaJain08

I have also computed the total wait time of customers in this function. Which makes sense because we would want to evaluate the waiting time of customers after the end of an event, which is also the beginning of the timing routine.

Arrival Event

As per the problem stated above, an arrival event can have multiple outcomes, which have been highlighted in the chart below.


Arrival
Update counters
Check Queue length
=0
Assign customer to teller and schedule departure
>=5
Increase queue length by 1 with 40% probabilty
=4
>=1, <4
Increase queue length by 1
50%
50%
Increase queue length by 1
Increase lost customers by 1
60%
40%
Increase queue length by 1
Increase lost customers by 1



Flow chart for arrival event (Photo by the author)

The outcome of the arrival event is decided by the number of customers in the queue and the state of the servers. For every outcome, statistical counters are updated, and the next event is scheduled.

```
    def arrival(self):
        self.num_arrivals += 1
        self.num_in_system += 1

        if self.num_in_q == 0:                          #schedule next departure or arriv
            if self.state_T1==1 and self.state_T2==1:
                self.num_in_q+=1
                self.number_in_queue+=1
                self.t_arrival=self.clock+self.gen_int_arr()


            elif self.state_T1==0 and self.state_T2==0:

                if np.random.choice([0,1])==1:
                    self.state_T1=1
                    self.dep1= self.gen_service_time_teller1()
                    self.dep_sum1 += self.dep1
                    self.t_departure1=self.clock + self.dep1
                    self.t_arrival=self.clock+self.gen_int_arr()

                else:
                    self.state_T2=1
                    self.dep2= self.gen_service_time_teller2()
                    self.dep_sum2 += self.dep2
                    self.t_departure2=self.clock + self.dep2
                    self.t_arrival=self.clock+self.gen_int_arr()


```

```python
            elif self.state_T1==0 and self.state_T2 ==1:       #if server 2 is busy customer goe
                self.dep1= self.gen_service_time_teller1()
                self.dep_sum1 += self.dep1
                self.t_departure1=self.clock + self.dep1
                self.t_arrival=self.clock+self.gen_int_arr()
                self.state_T1=1
            else:                                              #otherwise customer goes to serve
                self.dep2= self.gen_service_time_teller2()
                self.dep_sum2 += self.dep2
                self.t_departure2=self.clock + self.dep2
                self.t_arrival=self.clock+self.gen_int_arr()
                self.state_T2=1

        elif self.num_in_q < 4 and self.num_in_q >= 1:       #if queue length is less than 4 gen
            self.num_in_q+=1
            self.number_in_queue+=1
            self.t_arrival=self.clock + self.gen_int_arr()

        elif self.num_in_q == 4:                             #if queue length is 4 equal prob to
            if np.random.choice([0,1])==0:
                self.num_in_q+=1
                self.number_in_queue+=1
                self.t_arrival=self.clock + self.gen_int_arr()
            else:
                self.lost_customers+=1


        elif self.num_in_q >= 5:                            #if queue length is more than 5 60%
            if np.random.choice([0,1],p=[0.4,0.6])==0:
                self.t_arrival=self.clock+self.gen_int_arr()
                self.num_in_q+=1
                self.number_in_queue+=1
            else:
                self.lost_customers+=1
```

Bank_sim04.py hosted with by GitHub view raw

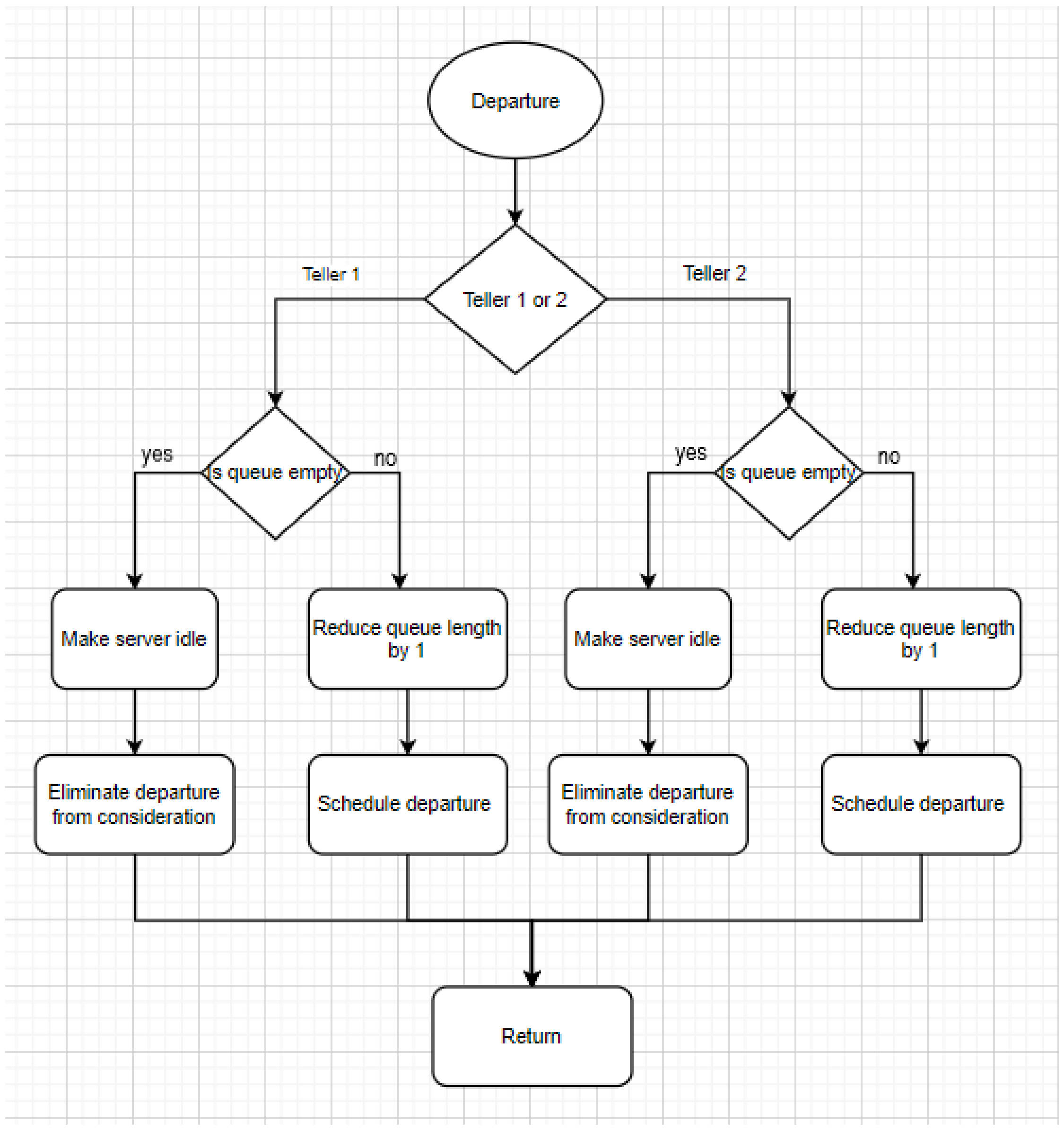
Source https://gist.github.com/BhavyaJain08

In the case of queue length being zero, the state of each server is checked, and the customer is assigned to the free server, a departure is scheduled along with an arrival. If both servers are busy and the queue length is zero, the queue length gets increased by one, and the next arrival is scheduled.

In all other cases, the queue length is increased by one, and the next arrival is scheduled, or a customer is lost, and the counter is updated.

Departure Event

A departure event occurs when the timing routine identifies any of the two departure events to be scheduled next. The flow of departure events is shown in the chart below



Flow chart for departure event (Photo by the author)

Source https://gist.github.com/BhavyaJain08

When there are customers waiting in the queue, the next customer is taken into service, the queue length is reduced by one, and the server stays busy.

When there are no customers in the queue, the server is made idle, and no departure events are scheduled for the future.

Generating arrival and service times

The arrival and service times are generated using inverse transform sampling. These random number generator functions are called in the arrival and departure functions to generate random arrivals and service times.

Source https://gist.github.com/BhavyaJain08

Running the Simulation

The next step is to run the simulation. I decided to run it from 9 am to 1 pm(4 hours) for 100 replications, and collect and store the data in an excel workbook for further analysis. I have used a for loop which updates the random number seed every time, runs for 100 times, and appends the results to a pandas data frame. Later this data frame is exported as an excel workbook

Source https://gist.github.com/BhavyaJain08

Results

Here is a snippet of the results obtained from 100 replications

Photo by the author

Finally, we can derive performance measures from these results to then analyze them and use them in improving efficiency, reducing costs, allocating resources, etc.

Photo by the author

Simulation can be used as a crucial decision-making tool in many industries, from manufacturing to service and even biology. This project is just a small example of the endless possibilities of simulation. There are many Softwares and resources out there to model complex systems and simulate them with ease. However, the goal of this project was to get a fundamental understanding of how a discrete event simulation works and its use as a decision-making tool.

References:

https://www.youtube.com/channel/UCkQ5dusAivKJ66hb3EDeBZg

Simulation Python Decision Making Discrete Event Simulation Queuing Theory