1. Abstract

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics, behaviours and functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.

Simulation is used in many contexts, such as simulation of technology for performance optimization, safety engineering, testing, training, education, and video games. Often, computer experiments are used to study simulation models. Simulation is also used with scientific modelling of natural systems or human systems to gain insight into their functioning, as in economics. Simulation can be used to show the eventual real effects of alternative conditions and courses of action. Simulation is also used when the real system cannot be engaged, because it may not be accessible, or it may be dangerous or unacceptable to engage, or it is being designed but not yet built, or it may simply not exist.

Table of Contents

CHAPTER 1 : INTRODUCTION

A simulation is the imitation of the operation of a real-world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system. The behaviour of a system as it evolves over time is studied by developing a simulation model. This model usually takes the form of a set of assumptions concerning the operation of the system. These assumptions are expressed in mathematical, logical, and symbolic relationships between the entities, or objects of interest, of the system.

Potential changes to the system can first be simulated, in order to predict their impact on system performance. Simulation can also be used to study systems in the design stage, before such systems are built. Thus, simulation modelling can be used both as an analysis tool for predicting the effect of changes to existing systems and as a design tool to predict the performance of new systems under varying sets of circumstances.

Therefore in this project we are simulating the Able Baker Call Centre Problem. Able Baker problem is the best example which illustrates more than one service Channel. It contains 2 technical support people Able and Baker. One among it is given more priority than the other ,resulting in providing faster service .Our aim is to find how this two people provide service to the customer calls during various arrival time and display the result in a tabular form for various calls to measure the performance. A suitable Front end is developed to provide, better clarity to solve this problem and make it easier to understand the input requirements.

.

CHAPTER 2 : PROBLEM DEFINITION

Simulation of a Able Baker Call Centre Problem, which involves providing service to the different customer calls ,based on the priority given to one of the 2 technical support people Able or Baker.

This is similar to a computer technical support centre where personnel take calls and provide service.

CHAPTER 3 : DESIGN AND IMPLEMENTATION

3.1 TOOLS AND TECHNOLOGY USED

3.1.1 PYTHON 2.7

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

3.1.2 PYCHARM

PyCharm is a Python IDE with complete set of tools for Python development. In addition, the IDE provides capabilities for professional Web development using the Django framework. Code faster and with more easily in a smart and configurable editor with code completion, snippets, code folding and split windows support.

3.2 DESIGN

The application has been designed as a web server which is used to accept the following requests to

1. Take the input parameters from user (home page)and
2. Display the simulation table result(solution page)

3.2.1 Take I/P (Home page)

Here we take the input parameters from the user to feed into the simulation program to get the correct simulation table.

The following are the input to be taken:

- **Customer Inter-arrival time set**: This set is used to decide from what values the random customer inter-arrival time will be chosen.
- **Able Service time set**: This set is used to decide from what values the service time of the ABLE server will be chosen.
- **Baker Service time set**: This set is used to decide from what values the service time of the BAKER server will be chosen.
- **Highest priority server**: This is used to decide which server to give higher priority than the other. The allowed values are – Able, Baker or Random.
- **Number of Customers**: The number of customers for whom the simulation is to be done is also required.

Input options:

The following are the value sets from which the above time sets can be chosen.

- 1, 2, 4, 8
- 1, 3, 9, 27
- 1, 2, 3, 4
- 2, 4, 6, 8
- 7, 8, 9, 10
- 5, 7, 11, 13

3.2.2 Simulate the ABLE-BAKER model

The Able-Baker model is an improvement on the single queuing system where we use two servers instead of one. The servers are:

- **Able** – This is usually the better server out of the two and is usually given random service time values lower than that of Baker.
- **Baker** – This is usually the backup server which is used to serve customers if Able is busy and not available. The service time values of Baker is usually higher than that of Able.

Every customer is given a set of attributes which gets calculated and stored when the given customer is encountered in the simulation. These values are taken in every record of the simulation table. They are:

Algorithm:

The following are the steps to take to implement the able-baker system.

Step 1 - Start.

Step 2 - For the first customer, assign all default 0 values each of the customer attributes and go to step 8. For any other customer, go to step 2.

Step 3 - Choose a random service time for the customer from the chosen service time set and set it to service_time.

Step 4 - Obtain the inter-arrival time of the customer from the set chosen and assign it to inter_arrival_time.

Step 5 - From the inter-arrival time, calculate the next arrival time and assign it to arrival_time.

Step 6 - Check if ABLE is busy. If not, assign it as the server. If it is busy, then assign BAKER as the server. If both are busy then wait till any one server is idle. When both servers are busy increment delay as long it does not change. (delay = when service begins – arrival time)

Step 7 - Calculate the time in the system by summing up delay and the service time.

Step 8 - Go to step 1 if there are still customers left, else step 9.

Step 9 - End

The customer class:

The customer class has the following attributes to maintain its state information.

- **inter_arrival_time** – the time in between the arrival of any two consequent customers.
- **arrival_time** – the time at which a customer arrives.
- **when_baker_available** – The time at which the baker server will be available.
- **when_able_available** - The time at which the able server will be available.
- **server** – The server that serves this customer.
- **service_time** – the time taken to service this customer.
- **service_begin_time** – The time at which service begins for this customer.
- **able_service_end_time** – the time at which the able server is available for service.
- **baker_service_end_time** – the time at which the baker server is available for service.
- **delay** – the time the customer spent idle in the system.
- **time_in_system** – the total time the customer spent in the system.

3.3 IMPLEMENTATION

Main.py

```
from .simulation import *
from test import *

def main():
    # Input data ->
    customerInterArrivalTime = [1, 2, 3, 4]
    ableServiceTime = [5, 6, 7, 8]
    bakerServiceTime = [10, 11, 12, 13]
    ableBakerPriority = 0  # If 0 => Able is first if 1 => baker is first if 2 => randomly chosen

    customerCount = 10
    timeLength = 0

    # Data processing  ->
    count = customerCount
    lili = customerListGenerator(customerInterArrivalTime, ableServiceTime, bakerServiceTime, ableBakerPriority, count)

    # printing output using django ->

    test(lili)

    return


if __name__ == "__main__" :
    main()
```

Simulation.py

```
from .customer import Customer
import random


ABLE = 0
BAKER = 1
RANDOM = 2
DEFAULT = ABLE
SERVERS = [ABLE, BAKER]

def getInterArrivalTime(customerList):
    return random.choice(customerList)


def getServer(previous_customer, priority, arrival_time):
    if priority==ABLE:
        if (previous_customer.able_service_end_time <= arrival_time):
            return ABLE
        else:
            if previous_customer.able_service_end_time <= previous_customer.baker_service_end_time :
                return ABLE
            else:
```

```
            return BAKER
    elif priority==BAKER:
        if (previous_customer.baker_service_end_time <= arrival_time):
            return BAKER
        else:
            if previous_customer.baker_service_end_time <= previous_customer.able_service_end_time :
                return BAKER
            else:
                return ABLE
    else :
        return random.choice(SERVERS)


def getServiceTime(server, able_list, baker_list):
    if server == ABLE:
        return random.choice(able_list)
    else:
        return random.choice(baker_list)


def customerListGenerator(customer_time_list, able_list, baker_list, priority, count):
    customer_list = []

    for i in range(count):

        if i==0 : #the first customer
            cust = Customer()
            if priority == RANDOM :
                priority = random.choice(SERVERS)

            service_time = None
            if priority == ABLE :
                service_time = getServiceTime(priority, able_list, baker_list)
                cust.server = ABLE
                cust.able_service_end_time = service_time
            elif priority == BAKER :
                service_time = getServiceTime(priority, able_list, baker_list)
                cust.server = BAKER
                cust.baker_service_end_time = service_time
            cust.service_time = service_time
            cust.time_in_system = service_time
            customer_list.append(cust)

        else : #all following customers
            previous = len(customer_list) - 1
            previous_customer = customer_list[previous]
            inter_arrival_time = getInterArrivalTime(customer_time_list)

            if priority == RANDOM :
                priority = random.choice(SERVERS)

            cust = Customer()

            arrival_time = previous_customer.arrival_time + inter_arrival_time
            server = getServer(previous_customer, priority, arrival_time)
            service_time = getServiceTime(server, able_list, baker_list)
            able_available_at = previous_customer.able_service_end_time
            baker_available_at = previous_customer.baker_service_end_time
```

```
        cust.arrival_time = arrival_time
        cust.server = server
        cust.inter_arrival_time = inter_arrival_time
        cust.service_time = service_time
        cust.when_able_available = able_available_at
        cust.when_baker_available = baker_available_at

        delay = None
        service_begins_at = None

        if server == ABLE:
            if arrival_time >= able_available_at:
                service_begins_at = arrival_time
                delay = 0
            else:
                service_begins_at = able_available_at
                delay = service_begins_at - arrival_time
            cust.able_service_end_time = service_begins_at + service_time
            cust.baker_service_end_time = baker_available_at
            cust.time_in_system = cust.able_service_end_time - cust.arrival_time

        elif server == BAKER:
            if arrival_time >= baker_available_at:
                service_begins_at = arrival_time
                delay = 0
            else:
                service_begins_at = baker_available_at
                delay = service_begins_at - arrival_time
            cust.able_service_end_time = able_available_at
            cust.baker_service_end_time = service_begins_at + service_time
            cust.time_in_system = cust.baker_service_end_time - cust.arrival_time


        cust.service_begin_time = service_begins_at
        cust.delay = delay
        customer_list.append(cust)

    return customer_list
```

Customer.py

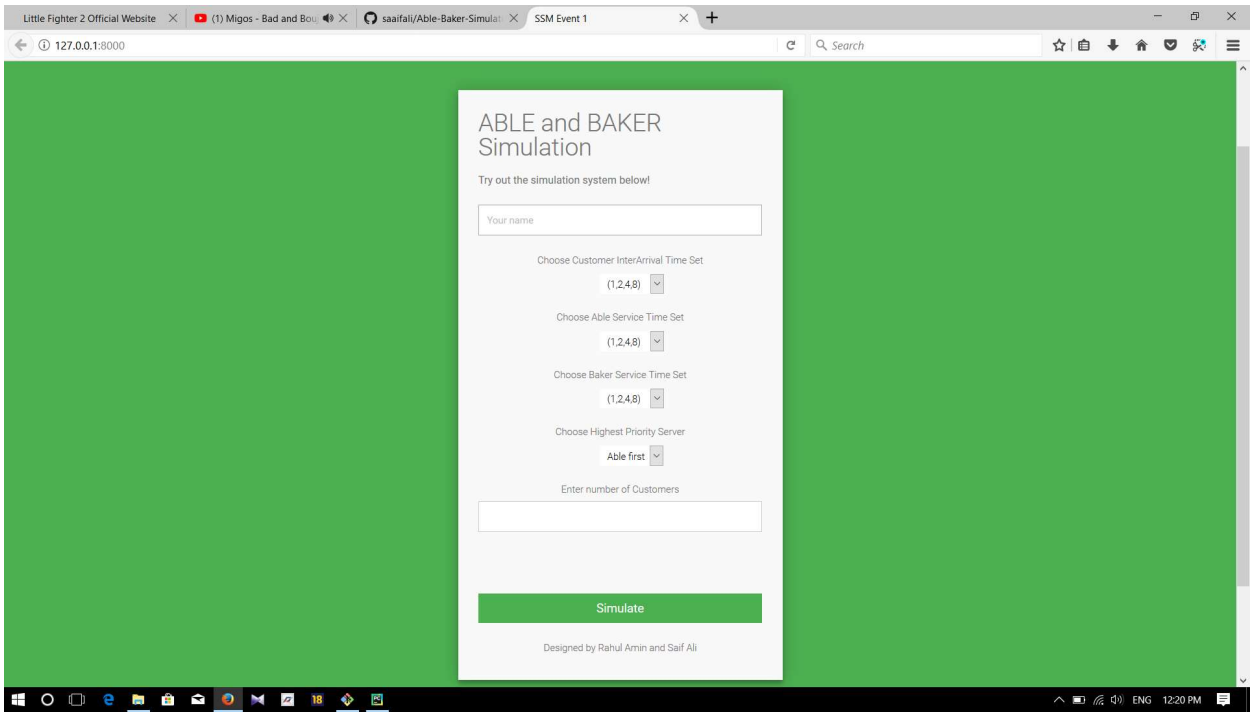
```
DEFAULT = 0

class Customer():
    inter_arrival_time = 0
    arrival_time = 0
    when_able_available = 0
    when_baker_available = 0
    server = DEFAULT
    service_time = 0
    service_begin_time = 0
    able_service_end_time = 0
    baker_service_end_time = 0
    delay = 0
    time_in_system = 0
```

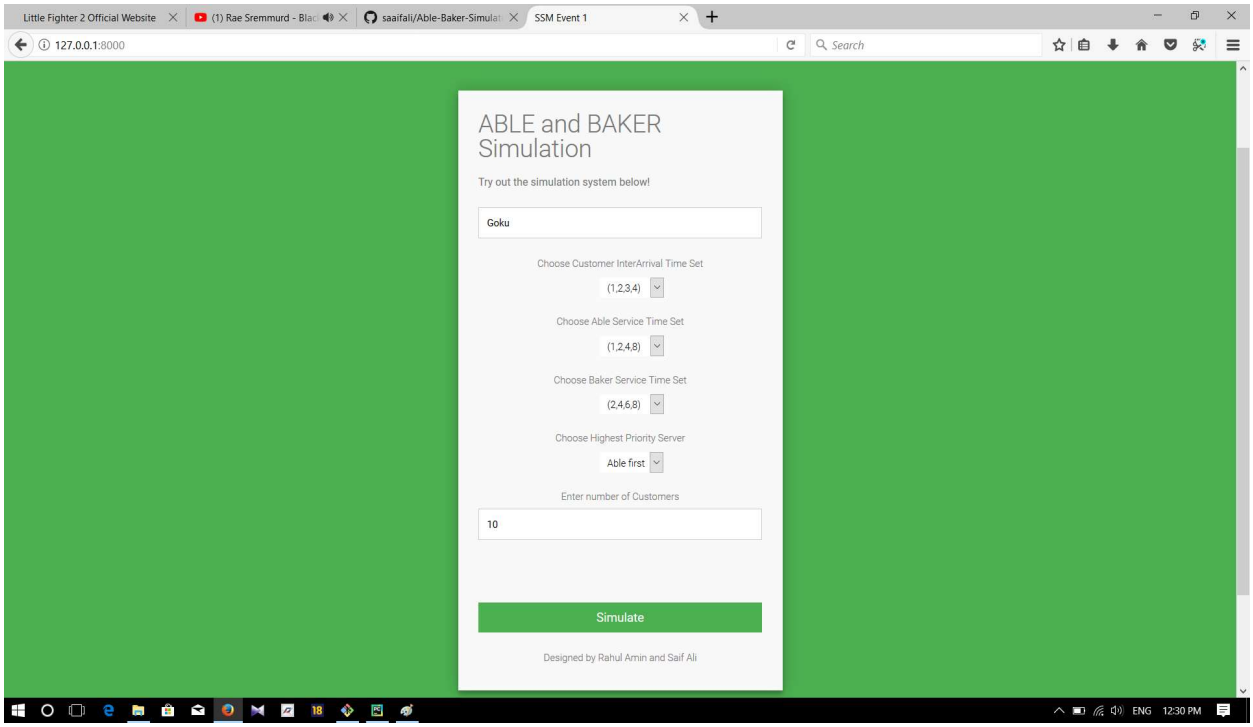
```
  def __init__(self,i1=0,i2=0,i3=0,i4=0,i5=0,i6=0,i7=0,i8=0,i9=0,i10=0, i11=0):
    self.inter_arrival_time = i1
    self.arrival_time = i2
    self.when_able_available = i3
    self.when_baker_available = i4
    self.server = i5
    self.service_time = i6
    self.service_begin_time = i7
    self.able_service_end_time = i8
    self.baker_service_end_time = i9
    self.delay = i10
    self.time_in_system = i11
```

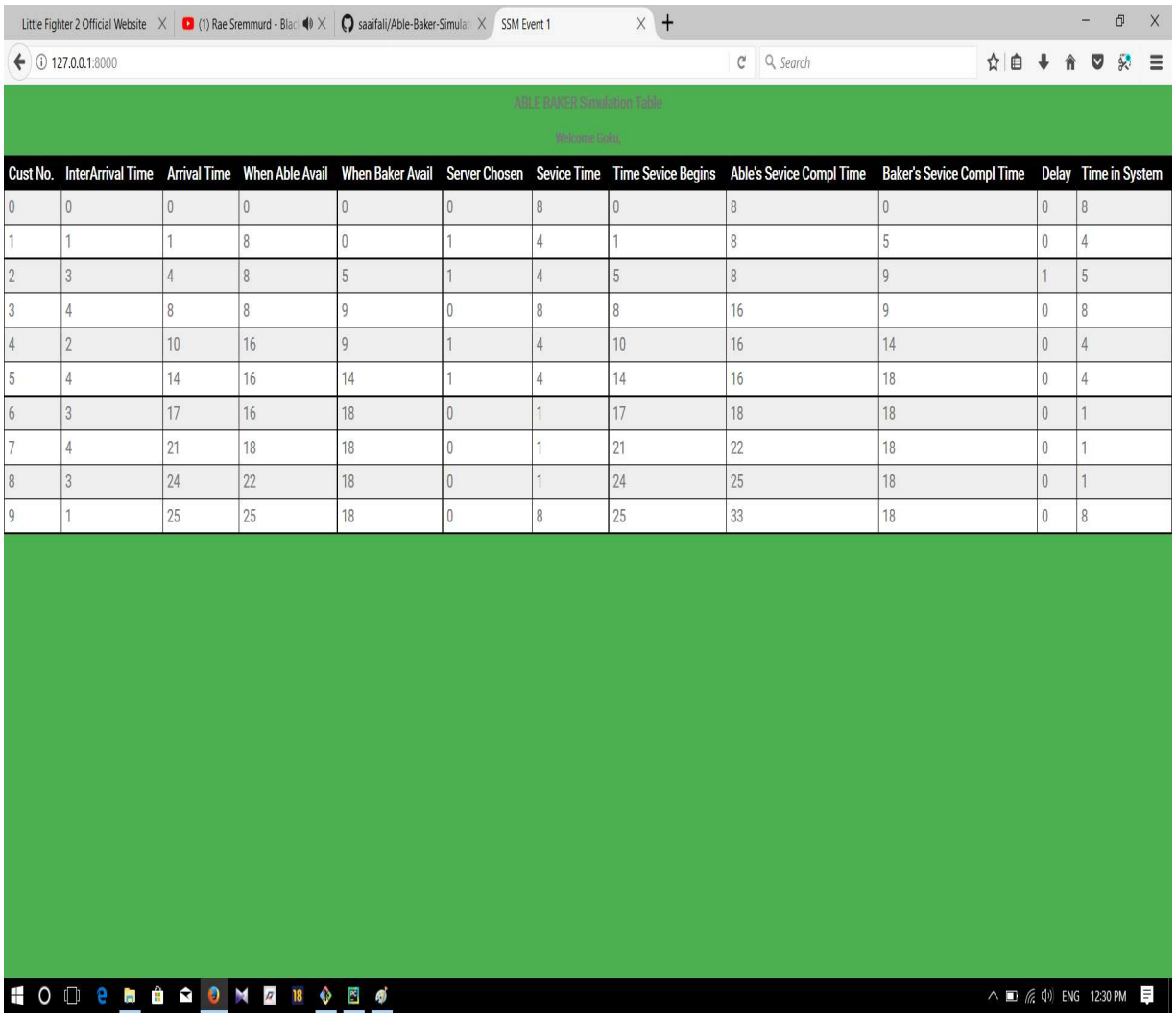
test.py

```
from copy import deepcopy

def test(lili):
  print
("no.\t|\tInterArrivalTime\t|\tArrivalTime\t|\tAbleAvailable\t|\tBakerAvailable\t|\tServer\t|\tServiceTime\t|\tBegin\t|\t
AbleEnd\t|\tBakerEnd\t|\tDelay\t|\tTimeInSYstem")
  for i in range(len(lili)):
    k = lili[i]
    print (str(i) + "\t|\t\t" + str(k.inter_arrival_time) + "\t\t\t|\t\t" + str(k.arrival_time) + "\t\t|\t\t" + str(
      k.when_able_available) + "\t\t\t|\t\t" + str(k.when_baker_available) + "\t\t\t|\t\t" + str(k.server) + "\t|\t\t" + str(
      k.service_time) + "\t\t|\t\t" + str(k.service_begin_time) + "\t|\t\t" + str(
      k.able_service_end_time) + "\t|\t\t" + str(k.baker_service_end_time) + "\t\t|\t\t" + str(
      k.delay) + "\t|\t\t" + str(k.time_in_system) + "\t\t|")

  allData = []
  for i in range(len(lili)):
    k = lili[i]
    s = [str(i), str(k.inter_arrival_time), str(k.arrival_time), str(k.when_able_available), str(k.when_baker_available),
       str(k.server), str(k.service_time), str(k.service_begin_time), str(k.able_service_end_time),
       str(k.baker_service_end_time), str(k.delay), str(k.time_in_system)]
    allData.append(deepcopy(s))
  return allData
```

CHAPTER 4 : OUTPUT







Cust No.	InterArrival Time	Arrival Time	When Able Avail	When Baker Avail	Server Chosen	Sevice Time	Time Sevice Begins	Able's Sevice Compl Time	Baker's Sevice Compl Time	Delay	Time in System
0	0	0	0	0	0	8	0	8	0	0	8
1	1	1	8	0	1	4	1	8	5	0	4
2	3	4	8	5	1	4	5	8	9	1	5
3	4	8	8	9	0	8	8	16	9	0	8
4	2	10	16	9	1	4	10	16	14	0	4
5	4	14	16	14	1	4	14	16	18	0	4
6	3	17	16	18	0	1	17	18	18	0	1
7	4	21	18	18	0	1	21	22	18	0	1
8	3	24	22	18	0	1	24	25	18	0	1
9	1	25	25	18	0	8	25	33	18	0	8

CHAPTER 5 : CONCLUSION

This simulation of an Able and Baker server shows us how a multi-server system is used and the how it affects the total service time, delay and the time a customer spends in the system.

This kind of a simulation can be used to represent a lot of real-world systems like the call centre system, a grocery store, a supermarket and so on. The table can be further used to calculate the efficiency and working of the simulation with the given parameters.

REFERENCES

1. Jerry Banks .John Carson, Barry Nelson, David Nicol, ”Discrete Event Simulation”.
2. https://en.wikipedia.org/wiki/Discrete_event_simulation.