Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №6 по дисциплине «Моделирование»

Тема Моделирование выставочного центра

Студент Слепокурова М.Ф.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2023 г.

Постановка задачи

В выставочный центр каждые 5 ± 2 минут приходят посетители. На входе в центр работают 3 кассы, каждая обслуживает одного посетителя за 5 ± 3 минут. С вероятностью 0.6 у посетителя уже есть входной билет, и ему не нужно проходить через кассы. Посетитель при входе выбирает кассу с наименьшей очередью.

После касс посетитель по билету может пойти в один из трех выставочных залов, выбирая тот, в очереди на вход к которому меньше людей. Однако если длина наименьшей очереди составляет 10 и более человек, посетитель покидает выставочный центр недовольным.

В виду ковидных ограничений в каждом зале может одновременно находиться от 5 до 7 человек. Каждые 75 ± 3 минут зал покидают все посетители, и контроллер впускает новую группу из очереди, если набралось нужное количество людей (от 5 до 7).

Смоделировать прием 1000 посетителей выставочным центром, взяв за единицу системного времени значение 0.01. Определить вероятность того, что посетитель покинет выставочный центр недовольным.

Теория

Анализ задачи

Эндогенные переменные: время оформления билета i-м кассиром и время проведения экскурсии в j-м выставочном зале.

Экзогенные переменные: n_0 — число довольных посетителей (посетивших выставочный зал), n_1 — число недовольных посетителей (не посетивших выставочный зал).

Вероятность того, что посетитель окажется недовольным, может быть рассчитана по следующей формуле:

$$\frac{n_1}{n_0 + n_1} \tag{1}$$

На рисунке 1 изображена структурная схема реализуемой модели.

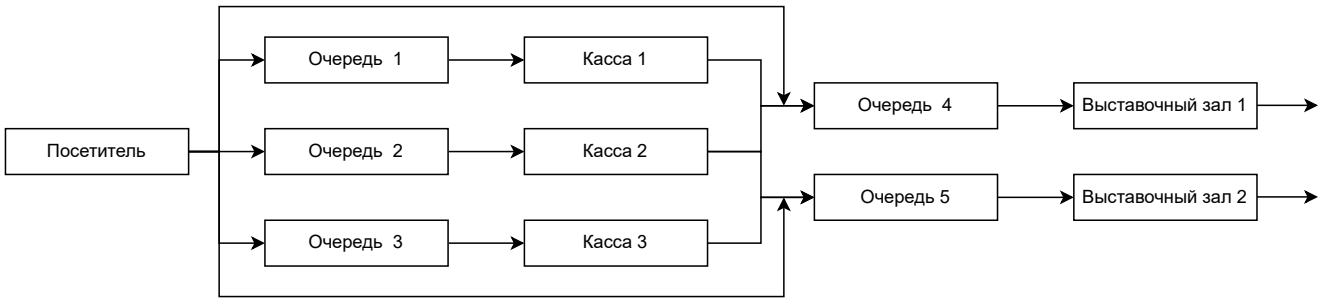

Рисунок 1 – Структурная схема модели

На рисунке 2 изображена схема модели в терминах СМО, где Г — генератор заявок, $К_i$ — канал обработки, $Н_i$ — накопитель.

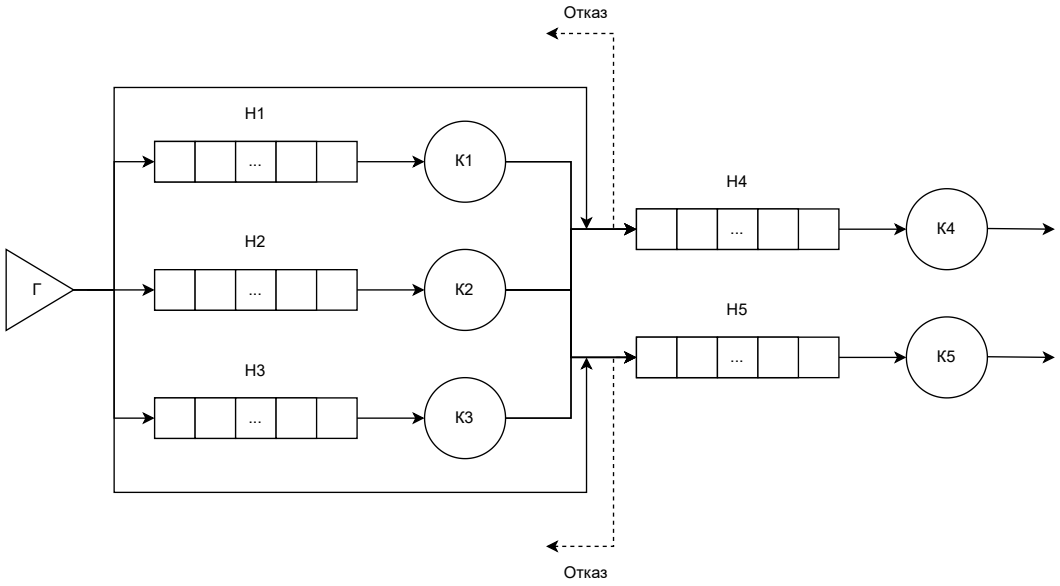

Рисунок 2 – Схема модели в терминах СМО

Принцип Δt

Данный принцип заключается в последовательном анализе состояний всех блоков в момент $t + \Delta t$ по заданному состоянию блоков в момент времени t, при этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на данный момент времени.

Средства реализации

Для реализации приложения был выбран язык программирования Python.

Листинг кода

```
from numpy.random import uniform
from random import random

REFUSAL_QUEUE_SIZE = 10

MIN_QUEUE_SIZE = 5
MAX_QUEUE_SIZE = 7

HAS_TICKET_PROBABILITY = 0.6

class TimeGenerator:
  def __init__(self, time, delta):
    self.time = time
    self.delta = delta

  def randomTime(self):
    return uniform(self.time - self.delta, self.time + self.delta)

class RequestGenerator:
  def __init__(self, timeGenerator, count, receivers = []):
    self.timeGenerator = timeGenerator
    self.requestCount = count
    self.receivers = receivers
    self.next = 0
    self.hasTicketProbability = HAS_TICKET_PROBABILITY

  def generateRequest(self, currTime):
    self.requestCount -= 1
    self.next = currTime + self.generateDuration()

    hasTicket = random() < self.hasTicketProbability
    if hasTicket: return self.receivers[0].getReceiver()

    minQueueSize = self.receivers[0].queueSize
    minReceiverId = 0
    for index, receiver in enumerate(self.receivers):
      if receiver.queueSize < minQueueSize:
        minQueueSize = receiver.queueSize
        minReceiverId = index
    return self.receivers[minReceiverId]

  def generateDuration(self):
    return self.timeGenerator.randomTime()
```

```
class RequestProcessor:
  def __init__(self, timeGenerator, receivers = []):
    self.timeGenerator = timeGenerator
    self.queueSize = 0
    self.next = 0
    self.receivers = receivers

  def pushRequest(self):
    self.queueSize += 1

  def popRequest(self, currTime):
    if self.queueSize > 0:
      self.queueSize -= 1
      self.next = currTime + self.generateDuration()
      return True
    else:
      self.next = 0
      return False

  def getReceiver(self):
    return self.receivers[0]

  def generateDuration(self):
    return self.timeGenerator.randomTime()

class Exhibition(RequestProcessor):
  def __init__(self, timeGenerator):
    super().__init__(timeGenerator)

  def pushRequest(self):
    super().pushRequest()

  def popRequest(self, currTime):
    if self.queueSize >= MIN_QUEUE_SIZE:
      self.queueSize -= min(self.queueSize, MAX_QUEUE_SIZE)
      self.next = currTime + self.generateDuration()
      return True
    else:
      self.next = 0
      return False

  def getReceiver(self):
    return None

  def generateDuration(self):
    return super().generateDuration()
```

```
class BoxOffice(RequestProcessor):
  def __init__(self, timeGenerator, receivers = []):
    super().__init__(timeGenerator, receivers)

  def pushRequest(self):
    super().pushRequest()

  def popRequest(self, currTime):
    return super().popRequest(currTime)

  def getReceiver(self):
    minQueueSize = self.receivers[0].queueSize
    minReceiverId = 0

    for index, receiver in enumerate(self.receivers):
      if receiver.queueSize < minQueueSize:
        minQueueSize = receiver.queueSize
        minReceiverId = index
    if minQueueSize >= REFUSAL_QUEUE_SIZE: return None
    return self.receivers[minReceiverId]

  def generateDuration(self):
    return super().generateDuration()

class Model:
  def __init__(self, generator, processors):
    self.generator = generator
    self.processors = processors

  def simulate(self, delta):
    refusalCount = 0
    generatedRequests = self.generator.requestCount

    blocks = [self.generator, *self.processors]

    currTime = 0
    while self.generator.requestCount > 0:
      for block in blocks:
        if block.next <= currTime:
          if isinstance(block, RequestGenerator):
            receiver = self.generator.generateRequest(currTime)
            if not receiver: refusalCount += 1
            else: receiver.pushRequest()
          elif isinstance(block, BoxOffice):
            if block.popRequest(currTime):
              receiver = block.getReceiver()
              if receiver is None: refusalCount += 1
              else: receiver.pushRequest()
          elif isinstance(block, Exhibition):
```

```
            block.popRequest(currTime)
        currTime += delta
    return { "refusalProbability": refusalCount / generatedRequests, "
        refusalCount": refusalCount }

requestCount = 1000

exhibition1 = Exhibition(TimeGenerator(75, 3))
exhibition2 = Exhibition(TimeGenerator(75, 3))

boxOffice1 = BoxOffice(TimeGenerator(5, 3), [exhibition1, exhibition2])
boxOffice2 = BoxOffice(TimeGenerator(5, 3), [exhibition1, exhibition2])
boxOffice3 = BoxOffice(TimeGenerator(5, 3), [exhibition1, exhibition2])

requestGenerator = RequestGenerator(TimeGenerator(5, 2), requestCount, [
    boxOffice1, boxOffice2, boxOffice3])

model = Model(requestGenerator, [boxOffice1, boxOffice2, boxOffice3,
    exhibition1, exhibition2])
result = model.simulate(0.01)

refusalCount, refusalProbability = result["refusalCount"], result["
    refusalProbability"]

print("Visitors: ", requestCount)
print("Probability of having a ticket: ", HAS_TICKET_PROBABILITY)
print("Refusals: ", refusalCount)
print("Refusal probability: ", round(refusalProbability, 2))
```

Демонстрация работы программы

На рисунке 3 изображен пример работы программы для 1000 посетителей.

```
Visitors:  1000
Probability of having a ticket:  0.6
Refusals:  53
Refusal probability:  0.05
```

Рисунок 3 – Пример работы программы — 1

На рисунке 4 изображен пример работы программы для 3000 посетителей. Заметим, что вероятность того, что посетитель уйдет недовольным, практически не меняется.

```
Visitors:  3000
Probability of having a ticket:  0.6
Refusals:  209
Refusal probability:  0.07
```

Рисунок 4 – Пример работы программы — 2

На рисунке 5 изображен пример работы программы для 5000 заявок. Заметим, что вероятность того, что посетитель уйдет недовольным, практически не меняется.

```
Visitors:  5000
Probability of having a ticket:  0.6
Refusals:  301
Refusal probability:  0.06
```

Рисунок 5 – Пример работы программы — 3