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

к лабораторной работе №6

По курсу: «Моделирование»

Студент ***ИУ7И-76Б*** _____ **Нгуен Ф. С.**
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Рудаков И.В.**
(Подпись, дата) (И.О. Фамилия)

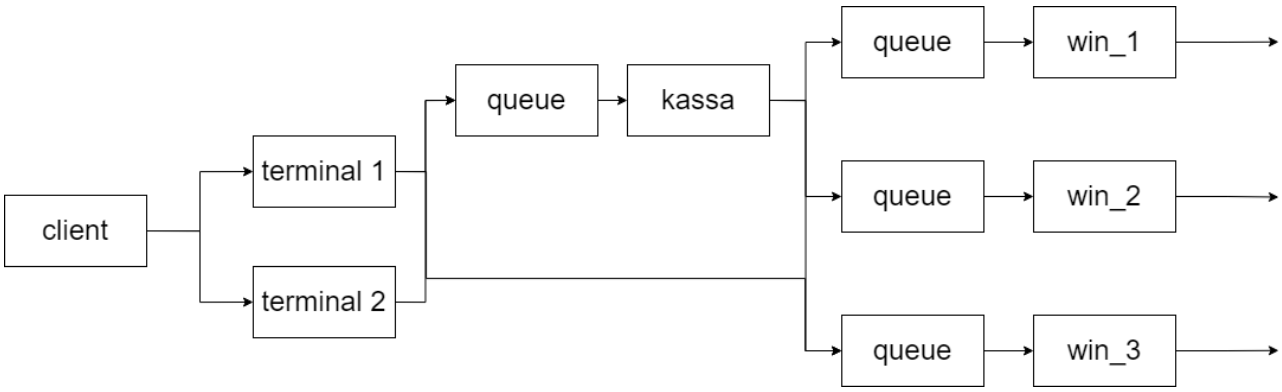
Москва, 2021 г.

Оглавление

I. Задача

В пункте выдачи приходят клиенты каждые 5±2 минуты. Каждому клиенту необходимо получить талон на одном из двух терминалов. Каждый терминал выдает талон с интервалом в 2±1 минуты. Если в очереди к терминалу находится больше 4 человек, клиент уходит. С вероятность 10% терминал не будет работать из-за технических неполадок, и тогда клиенту будет отказано. Клиенты приходят по равномерному распределению.

Если клиент не заплатил за заказ, он идет в кассу, а затем в окно выдачи. Если клиент уже заплатил, нет необходимости проходить кассу, а идти прямо к окну выдачи. . На кассе время работы с клиентом составляет 10 ± 5 минуты. вероятность невыполнения платежа (из-за ошибки клиента или технической ошибки) 0.05, тогда клиент отклоняется. Всего есть 3 окон, которые работают 10±5, 15±5, 15±10 минут. Вероятность отсутствия товара на складе 0.05, тогда клиент отклоняется.



II. Экспериментальная часть

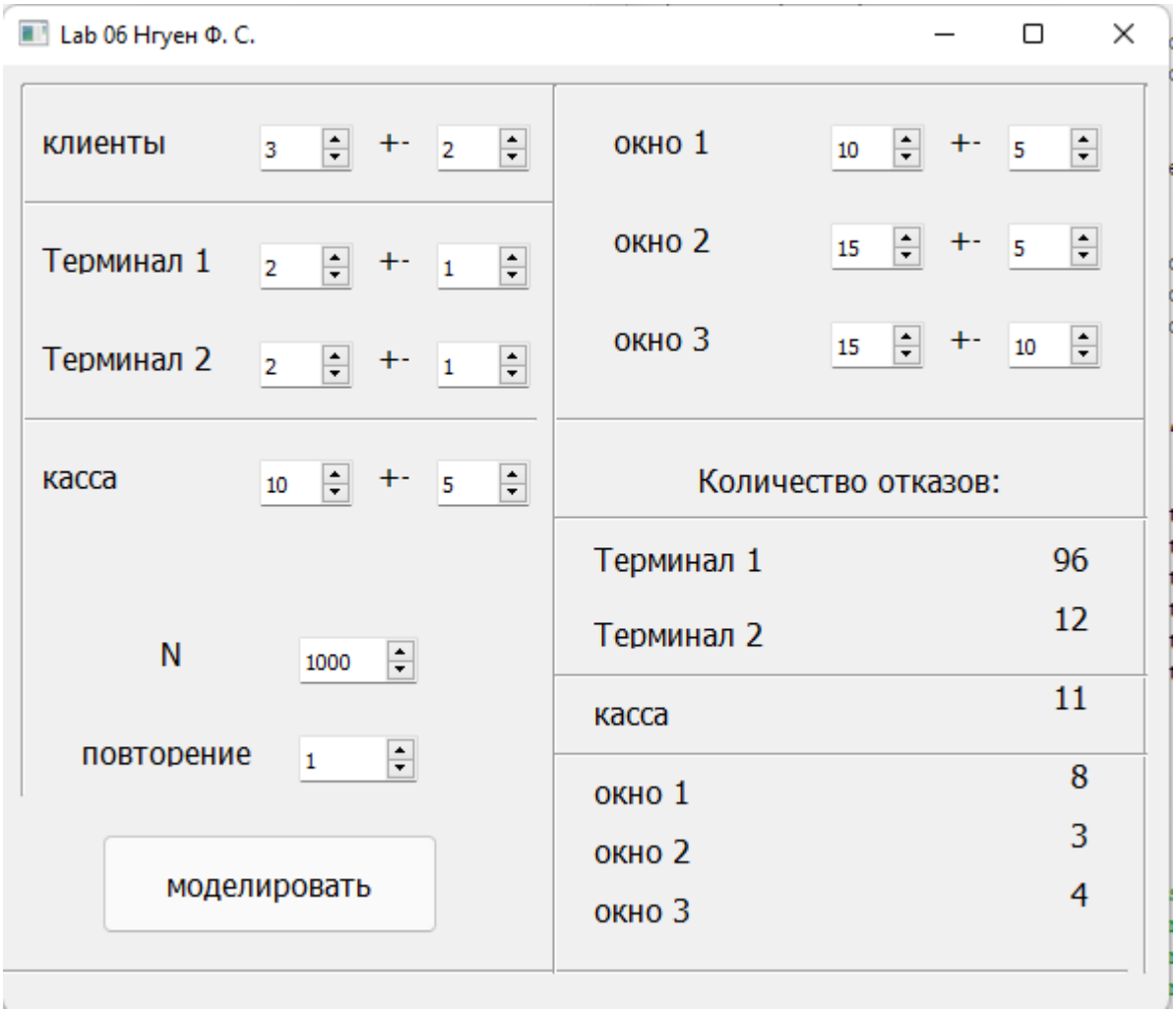


Рисунок 1. пример работы программы

III. Код программы:

```
from numpy import random as nr
from random import shuffle, random
from PyQt5 import QtWidgets, uic
from PyQt5.QtWidgets import QTableWidgetItem
from PyQt5.QtGui import QPen, QColor, QImage, QPixmap, QPainter
from PyQt5.QtCore import Qt, QTime, QCoreApplication, QEventLoop, QPointF


class RandomGenerator:
    def __init__(self, begin, delta=0):
        self.begin = begin
        self.d = delta

    def generate(self):
        if (self.d == 0):
            return self.begin
        return nr.uniform(self.begin - self.d, self.begin + self.d)


class GenerateRequest:
    def __init__(self, generator, name, count):
        self.random_generator = generator
```

```
        self.num_requests = count
        self.receivers = []
        self.next = 0
        self.name = name

    def generate_request(self):
        self.num_requests -= 1
        for receiver in self.receivers:
            if receiver.receive_request():
                return receiver
        return None

    def delay(self):
        return self.random_generator.generate()


class ProcessRequest:
    def __init__(self, generator, name, ban_probality=0, max_queue_size=-1,
end=False):
        self.random_generator = generator
        self.queue = 0
        self.received = 0
        self.max_queue = max_queue_size
        self.processed = 0
        self.next = 0
        self.receivers = []
        self.end = end
        self.name = name
        self.ban_probality = ban_probality

    def receive_request(self):
        if self.max_queue == -1 or self.max_queue > self.queue:
            self.queue += 1
            self.received += 1
            return True
        return False

    def process_request(self):
        if nr.sample() < self.ban_probality:
            return "ERR"
        if self.queue > 0:
            self.queue -= 1
            self.processed += 1
        shuffle(self.receivers)
        for receiver in self.receivers:
            if receiver.receive_request():
                return receiver
        return None

    def delay(self):
        return self.random_generator.generate()


class Model:
    def __init__(self, clients, terminals, kassa, wins):
        self.clients = clients
        self.terminals = terminals
        self.kassa = kassa
        self.wins = wins

    def event_mode(self):
        clients = self.clients
```

```
        clients.receivers = self.terminals.copy()
        self.terminals[0].receivers = [self.kassa, self.wins[0], self.wins[1],
self.wins[2]]
        self.terminals[1].receivers = [self.kassa, self.wins[0], self.wins[1],
self.wins[2]]
        self.kassa.receivers = [self.wins[0], self.wins[1], self.wins[2]]

        clients.next = clients.delay()
        self.terminals[0].next = self.terminals[0].delay()
        self.terminals[1].next = self.terminals[1].delay()

        blocks = []
        blocks += [clients]
        blocks += self.terminals
        blocks += [self.kassa]
        blocks += self.wins

        refusals = {}
        for block in blocks:
            refusals[block.name] = 0

        while clients.num_requests >= 0:
            current_time = clients.next
            for block in blocks:
                if 0 < block.next < current_time:
                    current_time = block.next

            for block in blocks:
                if current_time == block.next:
                    if not isinstance(block, ProcessRequest):
                        next_clients = clients.generate_request()

                        if next_clients is not None:
                            next_clients.next = current_time +
next_clients.delay()
                        else:
                            refusals[block.name] += 1
                        clients.next = current_time + clients.delay()
                    else:
                        next_process = block.process_request()
                        if block.queue == 0:
                            block.next = 0
                        else:
                            block.next = current_time + block.delay()

                        if next_process == "ERR":
                            refusals[block.name] += 1
                            continue

                        if block.end:
                            continue



                        if next_process is not None:
                            next process.next = \
                                current_time + next_process.delay()
                        else:
                            refusals[block.name] += 1

        return refusals

class Window(QtWidgets.QMainWindow):
```

```
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        uic.loadUi("window.ui", self)
        self.BtnModeling.clicked.connect(lambda: startModeling(self))


def startModeling(win):
    print("Getting Data")
    clients_number = win.n.value()
    repeat = win.repeat.value()
    #client
    time_client = win.time_client.value()
    d_time_client = win.d_time_client.value()
    print("Get Client OK {} {}".format(time_client, d_time_client))

    #terminal
    time_ter_1 = win.time_ter_1.value()
    d_time_ter_1 = win.d_time_ter_1.value()
    print("Test")
    time_ter_2 = win.time_ter_2.value()
    d_time_ter_2 = win.d_time_ter_2.value()
    print("Get Terminate OK")

    #kassa
    time_kassa = win.time_kassa.value()
    d_time_kassa = win.d_time_kassa.value()
    print("Get Kasssa OK")

    #wins
    time_win_1 = win.time_win_1.value()
    d_time_win_1 = win.d_time_win_1.value()
    time_win_2 = win.time_win_2.value()
    d_time_win_2 = win.d_time_win_2.value()
    time_win_3 = win.time_win_3.value()
    d_time_win_3 = win.d_time_win_3.value()
    print("Get Data OK")


    clients = GenerateRequest(RandomGenerator(time_client, d_time_client),
"entry", clients_number)

    terminals = [
        ProcessRequest(RandomGenerator(time_ter_1, d_time_ter_1), "terminal1",
ban_probality=0.1, max_queue_size=3),
        ProcessRequest(RandomGenerator(time_ter_2, d_time_ter_2), "terminal2",
ban_probality=0.1, max_queue_size=3),
    ]

    kassa = ProcessRequest(RandomGenerator(time_kassa, d_time_kassa), "kassa",
0.05, max_queue_size=10)

    wins = [
        ProcessRequest(RandomGenerator(time_win_1, d_time_win_1), "win1", 0.05,
end=True),
        ProcessRequest(RandomGenerator(time_win_2, d_time_win_2), "win2", 0.05,
end=True),
        ProcessRequest(RandomGenerator(time_win_3, d_time_win_3), "win3", 0.05,
end=True),
    ]

    model = Model(clients, terminals, kassa, wins)
    result = model.event_mode()
```

```
        win.res_ter_1.setText("{}".format(result['terminal1']))
        win.res_ter_2.setText("{}".format(result['terminal2']))
        win.res_win_1.setText("{}".format(result['win1']))
        win.res_win_2.setText("{}".format(result['win2']))
        win.res_win_3.setText("{}".format(result['win3']))
        win.res_kassa.setText("{}".format(result['kassa']))

        #print(result)
        print()
        print('Количество отказов:')
        print('Терминал 1: %d' % (result['terminal1']))
        print('Терминал 2: %d' % (result['terminal2']))
        print('касса : %d' % (result['kassa']))
        print('окно 1: %d' % (result['win1']))
        print('окно 2: %d' % (result['win2']))
        print('окно 3: %d' % (result['win3']))

if __name__ == "__main__":
        import sys
        app = QtWidgets.QApplication(sys.argv)
        w = Window()
        w.show()
        sys.exit(app.exec_())
```