ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор		Т.М. Татарникова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Вариант 8

по курсу: МОДЕЛИРОВАНИЕ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4128		В.А.Тарапанов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1 Цель и постановка задачи

1.1 Цель работы

Разработать имитационную модель в среде AnyLogic. Выполнить сравнительную оценку результатов имитационного моделирования и аналитического.

1.2 Задание

1. Создать диаграмму процесса, предложенного в варианте задания.

2. Разработать анимацию моделируемого процесса.

3. Продемонстрировать работу имитационной модели на компьютере.

4. Оценить следующие характеристики СМО, полученные в результате имитационного моделирования и аналитического моделирования: коэффициент загрузки p, Тпр, Тож, длину очереди L, вероятность отказа Ротк при необходимости, количество заявок в системе M.

5. Результаты, полученные в п. 4 объединить в таблицу сравнительных характеристик.

1.3 Условия варианта

СМО - обувной магазин в котором покупатели проходят три фазы обслуживания: 1-я - примерка и выбор обуви; 2-я - уплата денег в кассу; 3-я - получение покупки. Поток покупателей простейший I=45 человек/ч. В отделе примерки имеется 4 стула. Среднее время примерки и выбора обуви равно 5 мин. Затем покупатель направляется в кассу, где вторично становится в очередь. Среднее время оплаты в кассе равно 1 мин. После оплаты покупатель идёт на контроль, где становится в новую очередь и получает покупку. На контроле работают три продавца. Среднее время выдачи покупки 2 мин. Все потоки событий - простейшие. Рассматривая магазин как трёхфазную СМО, найти характеристики её эффективности: среднее число покупателей в очереди к первой, второй, третьей фазам обслуживания; среднее время пребывания покупателя в первой, второй, третьей фазах обслуживания;

2 Ход работы

В ходе выполнения работы была разработана диаграмма процесса в AnyLogic, что включало в себя создание различных элементов модели. В каждой системе были определены характеристики в соответствии с вариантом задания.

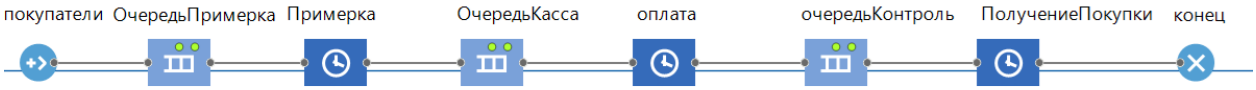

Рисунок 1 – Диаграмма СМО в AnyLogic

После создания диаграмы была разработана анимация, чтобы наглядно продемонстрировать работу модели.

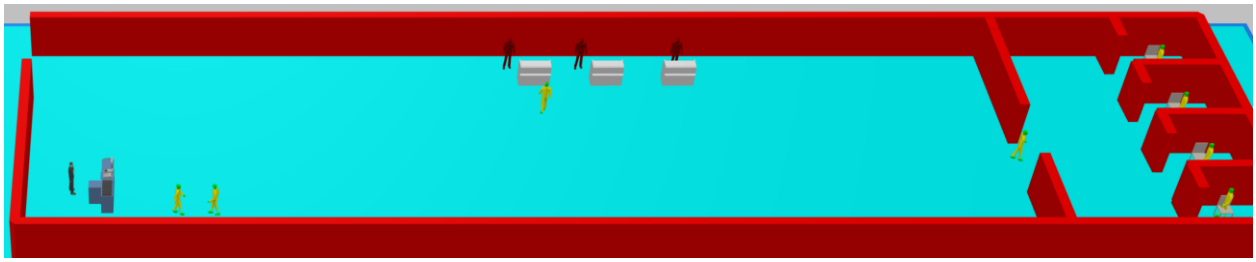

Рисунок 2 – Визуализация модели

После были добавлены гистограммы для отображения характеристик процесса.

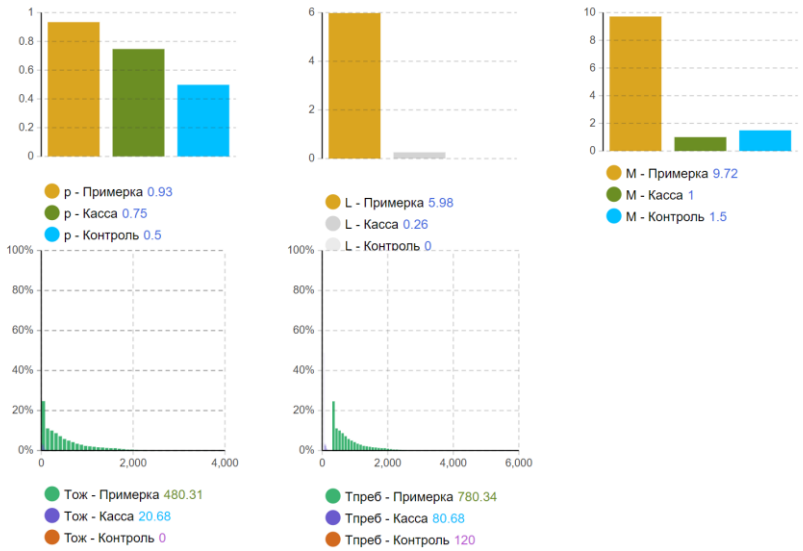

Рисунок 3-Результаты имитационного моделирования

Для нахождения характеристик аналитического моделирования был разработан программный код на языке Python и были получены результаты , представленные на рисунке 4.

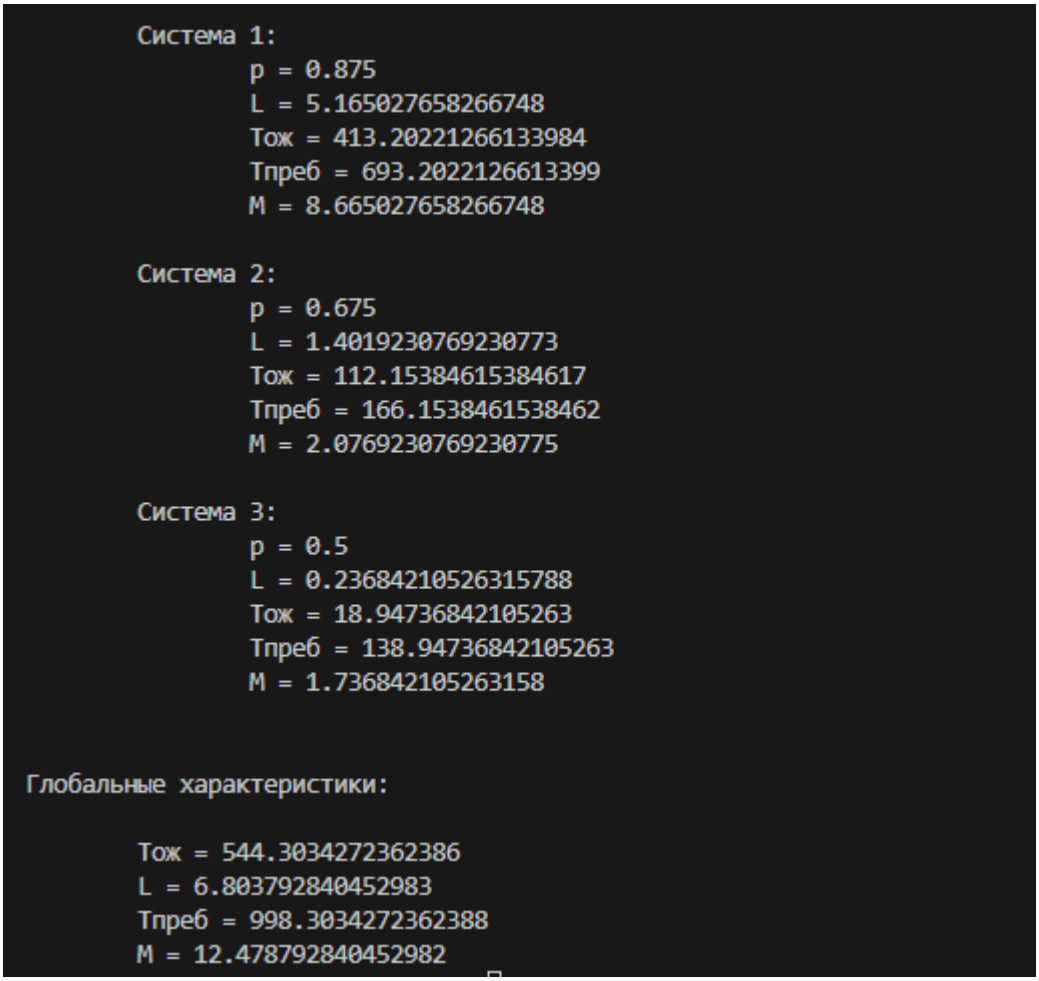

Рисунок 4 -Результаты аналитического моделирования

Таблица 1 – Сравнительная таблица арактеристик СМО

Характеристика	Имитационное моделирование	Аналитическое моделирование	Сравнительная оценка
p	0,93	0,875	+0,055
$T_{ож}$	500,99	544,3	+43,31
L	6,24	6,8	+0,56
$T_{пр}$	981,02	998,3	+17,28
M	12,22	12,48	+0.26

ВЫВОД

Имитационная модель была разработана в среде AnyLogic. В результате разработки была реализована модель трехфазной СМО, состоящей из двух многоканальных и одной одноканальной СМО.

Аналитическое моделирование проводилось при помощи разработанного кода на Python , представленного в Приложении А.

Полученные для сравнения характеристики (таб. 1) показывают незначительные различия между имитационным и аналитическим моделированием. Параметр p (коэффициент загрузки) имеет наименьшее отклонение между двумя методами моделирования, что говорит о их сходстве в оценке этого конкретного аспекта. Однако другие параметры, такие как среднее время пребывания в системе (Тож), среднее число заявок в очереди (L), и среднее число обслуженных заявок (M), показывают более значительные различия.

Сравнительная оценка указывает на превосходство имитационного моделирования в нескольких аспектах. Например, значение Тож и L оказались меньше в имитационной модели, что может указывать на более эффективную работу системы в реальном времени. Однако аналитическое моделирование также предоставляет оценки, которые могут быть использованы для сравнения с результатами имитационного моделирования и для общего понимания процесса.

Таким образом, оба метода моделирования имеют свои преимущества и могут быть полезны в различных ситуациях. Имитационное моделирование обеспечивает более точное представление о работе системы, но аналитическое моделирование также ценно для оценки параметров системы

ПРИЛОЖЕНИЕ А

```
from math import factorial
import copy

class QTransition:
    def __init__(self, system_id, probability):
        self.system_id = system_id
        self.probability = probability

class QSystem:
    def __init__(self, channels, service_time, requests_in_second, type):
        self.channels = channels
        self.service_time = service_time
        self.requests_in_second = requests_in_second
        self.type = type
        self.lambd = 0
        self.p = 0
        self.p0 = 0
        self.L = 0
        self.M = 0
        self.Twait = 0
        self.Texist = 0
        self.transitions = []

    def add_transition(self, index, p):
        self.transitions.append(QTransition(index, p))

    def calculate_load_ratio(self):
        self.p = self.lambd * self.service_time / self.channels

    def calculate_stationary_probability(self):
        arg1 = (self.lambd * self.service_time) ** self.channels / \
               (factorial(self.channels) * (1 - self.lambd * self.service_time /
self.channels))
        arg2 = sum([(self.lambd * self.service_time) ** m / factorial(m) for m in
range(self.channels)])
        self.p0 = 1 / (arg1 + arg2)

    def calculate_queue_length(self):
        arg1 = (self.lambd * self.service_time) ** (self.channels + 1)
        arg2 = factorial(self.channels) * self.channels * (1 - self.lambd *
self.service_time / self.channels) ** 2
        self.L = self.p0 * (arg1 / arg2)

    def calculate_request_amount(self):
        self.M = self.L + self.channels * self.p

    def calculate_waiting_time(self):
        self.Twait = self.L / self.lambd
```

```
    def calculate_existence_time(self):
        self.Texist = self.Twait + self.service_time

    def clone(self):
        return copy.deepcopy(self)

class QNetwork:
    def __init__(self, systems):
        self.systems = systems
        self.transition_matrix = self.transition_matrix_build(systems)
        self.L = 0
        self.M = 0
        self.Twait = 0
        self.Texist = 0
        self.D = []

        self.init_system_output_params()
        self.find_queue_length()
        self.find_request_amount()
        self.find_waiting_time()
        self.find_existence_time()
        self.find_capacity_reserves()

    def find_capacity_reserves(self):
        for system in self.systems:
            if system.type == "Entrance":
                denominator = 1 / system.requests_in_second
                is_limit = False
                while not is_limit:
                    denominator -= 1
                    max_requests = 1 / denominator

                    test_systems = [system.clone() for system in self.systems]
                    for test_system in test_systems:
                        test_system.lambd = 0
                        test_system.p = 0
                    for i, test_system in enumerate(test_systems):
                        if test_system.type == "Entrance":
                            test_system.lambd = max_requests
                        for transition in test_system.transitions:
                            test_systems[transition.system_id].lambd += \
                                test_system.lambd *
self.transition_matrix[i][transition.system_id]
                    for test_system in test_systems:
                        test_system.calculate_load_ratio()
                        if test_system.p > 1:
                            is_limit = True
                            break
                denominator += 1
```

```
                self.D.append((1 / denominator) - system.requests_in_second)

    def find_queue_length(self):
        self.L = sum(system.L for system in self.systems)

    def find_request_amount(self):
        self.M = sum(system.M for system in self.systems)

    def find_waiting_time(self):
        self.Twait = sum(system.lambd * system.Twait for system in self.systems)
/ self.systems[0].requests_in_second

    def find_existence_time(self):
        self.Texist = sum(system.lambd * system.Texist for system in
self.systems) / self.systems[0].requests_in_second

    def init_system_output_params(self):
        for i, system in enumerate(self.systems):
            if system.type == "Entrance":
                system.lambd = system.requests_in_second
            for transition in system.transitions:
                self.systems[transition.system_id].lambd += \
                    system.lambd *
self.transition_matrix[i][transition.system_id]
        for system in self.systems:
            system.calculate_load_ratio()
            system.calculate_stationary_probability()
            system.calculate_queue_length()
            system.calculate_request_amount()
            system.calculate_waiting_time()
            system.calculate_existence_time()

    @staticmethod
    def transition_matrix_build(systems):
        transition_matrix = []
        for system in systems:
            transition_row = [0] * (len(systems) + 1)
            probability_sum = 0
            for transition in system.transitions:
                probability_sum += transition.probability
                transition_row[transition.system_id] = transition.probability
            if system.type == "Exit":
                transition_row[len(systems)] = 1 - probability_sum
            transition_matrix.append(transition_row)
        return transition_matrix

    def print_output(self):
        print("Локальные характеристики:")
        for i, system in enumerate(self.systems):
            print(f"\n\tСистема {i + 1}:")
```

```
            print(f"\t\tp = {system.p}")
            print(f"\t\tL = {system.L}")
            print(f"\t\tТож = {system.Twait}")
            print(f"\t\tТпреб = {system.Texist}")
            print(f"\t\tM = {system.M}")
        print("\n\nГлобальные характеристики:\n")
        print(f"\tТож = {self.Twait}")
        print(f"\tL = {self.L}")
        print(f"\tТпреб = {self.Texist}")
        print(f"\tM = {self.M}")


def main():
    system1 = QSystem(4, 280, 0.0125, "Entrance")
    system2 = QSystem(1, 54, 0, "Intermediate")
    system3 = QSystem(3, 120, 0, "Exit")

    system1.add_transition(1, 1)
    system2.add_transition(2, 1)

    systems = [system1, system2, system3]
    q_network = QNetwork(systems)
    q_network.print_output()

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