Практическая работа «Выполнение экспериментов с имитационной моделью на языке Python»

Библиотека Simpy обеспечивает поддержку описания и запуска дискретно-событийных моделей на Python. Simpy не является полноценной графической средой для построения, выполнения и составления отчетов по результатам моделирования, однако предоставляет фундаментальные компоненты модели. Ее можно подключить к известным библиотекам Python, таким как Matplotlib, Tkinter, Streamlit для обеспечения построения графиков и визуализации модельного процесса соответственно.

Для примера опишем процесс организации пропуска потока посетителей с входной очередью на масштабное мероприятие (концерт, ярмарка, парк аттракционов). Другими похожими процессами систем массового обслуживания, которые следуют аналогичной схеме, могут быть супермаркет, кинотеатр, железнодорожный вокзал и т.п.

В этом примере будем моделировать ситуацию входного потока, который полностью обслуживается общественным транспортом (рисунок 1): автобус на регулярной основе будет высаживать несколько посетителей, которым затем нужно будет отсканировать билеты перед входом на мероприятие. У одних посетителей, предположим, будут заранее приобретённые билеты (бейджи), в то время как другим нужно будет сначала подойти к кассам, чтобы купить билеты. Дополнительно усложним модель учетом варианта возможного прихода к кассам группами (имитируя семейную/групповую покупку билетов); но на контрольном пункте каждому человеку нужно будет сканировать свой билет отдельно.

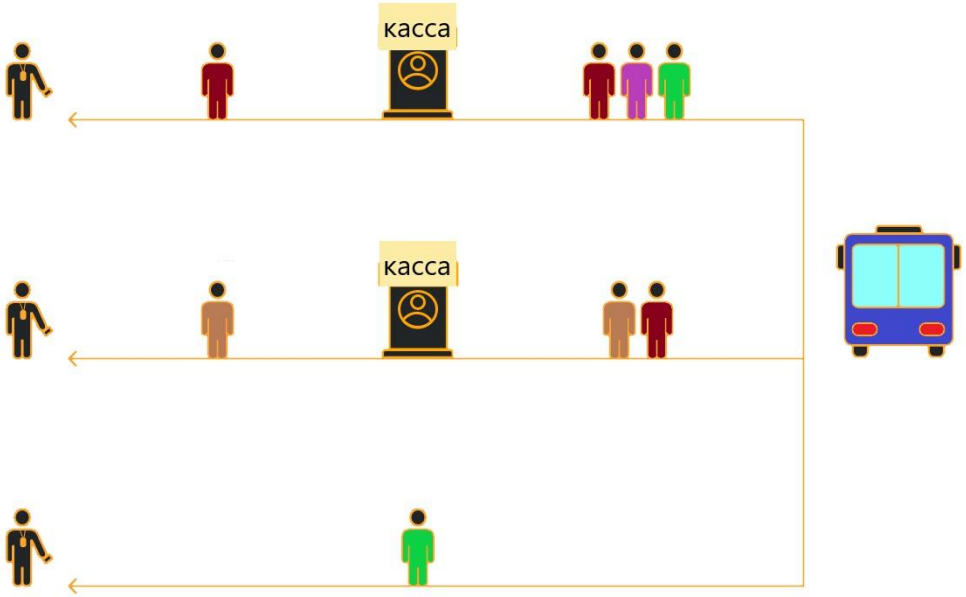


Рисунок 1. Схема движения потока участников

Чтобы смоделировать это, нужно решить, как представить различные события, используя распределения вероятностей. Настройки, которые сделаем в нашей реализации, включают такие исходные данные:

- Автобус прибывает в среднем 1 раз в 3 минуты.
- В каждом автобусе будут находиться от 35 до 125 посетителей, что моделируется с использованием нормального распределения (μ = 80, σ = 15).
- Посетители будут формировать группы из 2,5 ± 1,5 человек, что смоделируем используя нормальное распределение (μ = 2,5, σ = 0,5), и округляя это значение до ближайшего целого числа.
- Предположим, что для 50% посетителей потребуется приобрести билеты в кассах, еще 50% придут с билетами, уже купленными заранее (онлайн).
- Посетителям требуется пройти от автобуса к кассе за время, которое составляет в среднем 1 минуту (нормальное распределение, μ = 1, σ = 0,25).
- Посетителям требуется пройти от кассы до контроля билетов за время, которое составляет около 0,5 минуты (нормальное распределение, μ = 0,5, σ = 0,1).
- Посетители по прибытии всегда выбирают самую короткую очередь, и для каждой очереди есть один продавец или контролёр.
- Для покупки билетов в кассе требуется 1 минута (нормальное распределение, μ = 1, σ = 0,2).
- Сканирование у контролёра занимает 0.4 минут (нормальное распределение, μ = 0,4, σ = 0,1).

Параметрами, представляющими интерес для анализа, являются количество касс продажи билетов (SELLER_LINES) и количество контролёров билетов (SCANNER_LINES).

Для подготовки модели нам потребуется Python с дополнительными фреймворками simpy, numpy, matplotlib, scipy, streamlit. Библиотеки random, collections, math обычно входят в дистрибутив Python.

Фаза 1

Начнём разработку модели с базового набора процессов и ресурсов.

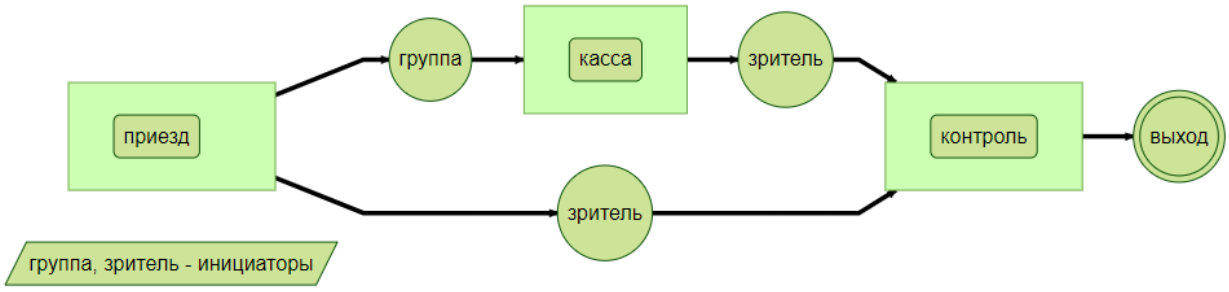


Рисунок 2. Схема логической модели

Объявляем ресурсы:

```
seller_lines = [simpy.Resource(env, capacity = SELLERS_PER_LINE)
for _ in range(SELLER_LINES)]
scanner_lines = [simpy.Resource(env, capacity = SCANNERS_PER_LINE)
for _ in range(SCANNER_LINES)]
```

Запишем процессы -

```
Процесс def bus_arrival() – прибытие автобуса
Процесс def purchasing_customer()  - появление клиента в кассу
Процесс def scanning_customer() – приход клиента на контроль
```

Затем запустим модель на 45 единиц модельного времени (т.е. 45 минут реального процесса).

Начнем формирование модели с добавления нижерасположенного кода в программу.

```
# -*- coding: utf-8 -*-
import random as rd
import simpy
from collections import defaultdict

# --------------------------
#  CONFIGURATION
# --------------------------

BUS_ARRIVAL_MEAN = 3
BUS_OCCUPANCY_MEAN = 80
BUS_OCCUPANCY_STD = 15

PURCHASE_RATIO_MEAN = 0.5
PURCHASE_GROUP_SIZE_MEAN = 2.5
PURCHASE_GROUP_SIZE_STD = 0.5

TIME_TO_WALK_TO_SELLERS_MEAN = 1
TIME_TO_WALK_TO_SELLERS_STD = 0.25
TIME_TO_WALK_TO_SCANNERS_MEAN = 0.5
TIME_TO_WALK_TO_SCANNERS_STD = 0.1

SELLER_LINES = 6
SELLERS_PER_LINE = 1
SELLER_MEAN = 1
SELLER_STD = 0.2

SCANNER_LINES = 6
SCANNERS_PER_LINE = 2
SCANNER_MEAN = 0.4
SCANNER_STD = 0.1

seller_lines, scanner_lines, event_log = [],[],[]
arrivals = defaultdict(lambda: 0)
seller_waits = defaultdict(lambda: [])
scan_waits = defaultdict(lambda: [])
```

```
rd.seed(4210)

# -------------------------
#   ANALYTICAL GLOBALS
class Globals:
    # предварительно запишем времена прибытия автобусов и кол-во пассажиров
    # чтобы точно управлять случайностью
    ARRIVALS = [rd.expovariate(1 / BUS_ARRIVAL_MEAN) for _ in range(50)]
    ON_BOARD = [abs(int(rd.gauss(BUS_OCCUPANCY_MEAN, BUS_OCCUPANCY_STD))) for _ in range(50)]
    ARRIVAL_ORIGIN = ARRIVALS.copy()  # сохраним оригинальные списки для
    ON_BOARD_ORIGIN= ON_BOARD.copy()  # повторного использования в эксперименте
    # словари для аналитики
    seller_queues = {v:[] for v in range(SELLER_LINES)}

# -------------------------
#  SIMULATION процессы модели
# -------------------------
def pick_shortest(lines):
    """
        определяем самую короткую очередь к ресурсам модели -
        функция возвращает кортеж, где 0й элемент - SimPy resource,
        а 1й элемент - номер ресурса (начиная с 1, а не с 0) //
        номер очереди выбирается случайно после перемешивания shuffle, чтобы не всегда начинать с 1ой
    """
    shuffled = list(zip(range(len(lines)), lines)) # list of tuples (i, line)
    rd.shuffle(shuffled)
    shortest = shuffled[0][0]
    for i, line in shuffled:
        if len(line.queue) < len(lines[shortest].queue):
            shortest = i
            break
    return (lines[shortest], shortest + 1)

def bus_arrival(env, seller_lines, scanner_lines):
    """
```

```
      моделируем приезд автобуса через BUS_ARRIVAL_MEAN минут,
      который привозит BUS_OCCUPANCY_MEAN людей.
      это первое событие в модели, после которого срабатывают все другие события
    """
    # уникальные ID для автобуса и людей будут нужны для визуализации
    next_bus_id,next_person_id = 0, 0
    while True:
        next_bus = Globals.ARRIVALS.pop()
        on_board = Globals.ON_BOARD.pop()
        # ждать следующий автобус
        yield env.timeout(next_bus)
        # вызов register_bus_arrival() для записи в логи
        clientIDs = list(range(next_person_id, next_person_id + on_board))
        next_person_id += on_board
        next_bus_id += 1
        #
        while len(clientIDs) > 0:
            group_size = min(round(abs(rd.gauss(PURCHASE_GROUP_SIZE_MEAN, PURCHASE_GROUP_SIZE_STD))), len(clientIDs))
            people_processed = clientIDs[-group_size:] # получить последние элементы из группы
            clientIDs = clientIDs[:-group_size] # оставить id тем кто еще остался
            # кто-то идет в кассу, а кто-то уже с билетом идет на контроль
            if rd.random() > PURCHASE_RATIO_MEAN:
                env.process(scanning_customer(env, people_processed, scanner_lines,
                            TIME_TO_WALK_TO_SELLERS_MEAN + TIME_TO_WALK_TO_SCANNERS_MEAN,
                            TIME_TO_WALK_TO_SELLERS_STD + TIME_TO_WALK_TO_SCANNERS_STD))
            else:
                env.process(purchasing_customer(env, people_processed, seller_lines, scanner_lines))

def purchasing_customer(env, people_processed, seller_lines, scanner_lines):
    # подойти к кассе
    walk_begin = env.now
    sigm3l = TIME_TO_WALK_TO_SELLERS_MEAN - TIME_TO_WALK_TO_SELLERS_STD*3
    if sigm3l < 0: sigm3l=0   # настройка треугольного распределения
    sigm3h = TIME_TO_WALK_TO_SELLERS_MEAN + TIME_TO_WALK_TO_SELLERS_STD*3
    yield env.timeout(rd.triangular(sigm3l, sigm3h, TIME_TO_WALK_TO_SELLERS_MEAN))
    walk_end = env.now
```

```
    # встать в очередь
    queue_begin = env.now
    # клиент всегда выбирает самую короткую очередь
    seller_line = pick_shortest(seller_lines)
    # ждем начала обслуживания
    with seller_line[0].request() as req:
        # подождать в очереди
        yield req
        queue_end = env.now
        # покупка билета
        sale_begin = env.now
        sigm3l = SELLER_MEAN - SELLER_STD*3
        if sigm3l < 0: sigm3l = 0   # настройка треугольного распределения
        sigm3h = SELLER_MEAN + SELLER_STD*3
        yield env.timeout(rd.triangular(sigm3l,sigm3h,SELLER_MEAN))
        sale_end = env.now
        # вызов register_group_moving_() для записи в логи
        env.process(scanning_customer(env, people_processed, scanner_lines, TIME_TO_WALK_TO_SCANNERS_MEAN,
TIME_TO_WALK_TO_SCANNERS_STD))

def scanning_customer(env, people_processed, scanner_lines, walk_duration, walk_std):
    # подойти на контроль билетов
    walk_begin = env.now
    yield env.timeout(abs(rd.gauss(walk_duration, walk_std)))
    walk_end = env.now

    # клиент всегда выбирает самую короткую очередь
    queue_begin = env.now
    scanner_line = pick_shortest(scanner_lines)
    with scanner_line[0].request() as req:
        # подождать в очереди
        yield req
        queue_end = env.now

        # контроль билета у каждого клиента
        for person in people_processed:
```

```
            scan_begin = env.now
            yield env.timeout(abs(rd.gauss(SCANNER_MEAN, SCANNER_STD)))
            scan_end = env.now
            # вызов register_visitor_moving_() для записи в логи

# основная функция для запуска модели
def model_env():
  global seller_lines, scanner_lines
  env = simpy.Environment()
  seller_lines = [simpy.Resource(env, capacity = SELLERS_PER_LINE) for _ in range(SELLER_LINES)]
  scanner_lines = [simpy.Resource(env, capacity = SCANNERS_PER_LINE) for _ in range(SCANNER_LINES)]
  env.process(bus_arrival(env, seller_lines, scanner_lines))
  env.run(until = 45)
  print("OK!",env.now)

model_env()
```

Запустим модель для проверки. Если есть сообщение в консоли -
OK! 45

- значит, модель работает, но результатов не показывает и не собирает.

Фаза 2

Теперь добавим функции сбора статистики для наблюдения результатов.
Добавим в функцию model_env()

```
event_log =[]
```

перед строкой env = simpy.Environment().

Подготовим специальный мониторинговый процесс сбора статистики по очередям.
Вставим код этой функции, например, перед def bus_arrival().

```
# специальная функция для сбора данных
def monitor(ev):
   global seller_lines
   while True:
      # запомним текущую длину очередей
      for i in range(len(seller_lines)):
         Globals.seller_queues[i].append(len(seller_lines[i].queue))
      yield ev.timeout(1.0)
```

Активируем этот процесс перед запуском модельной системы (курсивом дан написанный ранее код):

env.process(bus_arrival(env, seller_lines, scanner_lines))

```
env.process(monitor(env))
```

env.run(until = 45)

Добавим нижерасположенный код функций после строк с `class Globals` перед разделом кода `SIMULATION`.

```
def avg_wait(raw_waits):
    waits = [ w for i in raw_waits.values() for w in i ]
    ret =round(sum(waits)/len(waits), 1) if len(waits) > 0 else 0
    return ret

def register_bus_arrival(time, bus_id, people_created):
    arrivals[int(time)] += len(people_created)
    print(f"Автобус {bus_id+1} приехал в {round(time, 2)} с {len(people_created)} чел")
    event_log.append({
        "event": "BUS_ARRIVAL", "time": round(time, 2), "busId": bus_id+1, "peopleCreated": people_created
    })

def register_group_moving_from_bus_to_seller(people,walk_begin,walk_end,seller_line,queue_begin,queue_end,sale_begin,sale_end):
    waitq = queue_end - queue_begin
    service_time = sale_end - sale_begin
    seller_waits[int(queue_end)].append(waitq)
    print(f"Группа {len(people)} чел ждала {round(waitq,2)} мин в очереди_{seller_line}, обслужилась за {round(service_time,2)} мин")
    event_log.append({
        "event": "WAIT_IN_SELLER_LINE", "people": people, "sellerLine": seller_line,
        "time": round(queue_begin, 2), "duration": round(waitq, 2)
    })

def register_visitor_moving_to_scanner(person, walk_begin, walk_end, scanner_line, queue_begin, queue_end, scan_begin, scan_end):
    waitq = queue_end - queue_begin
    service_time = scan_end - scan_begin
    scan_waits[int(queue_end)].append(waitq)
    print(f"Клиент на контроле ждал {round(waitq,2)} мин в очереди_{scanner_line}, обслуживался {round(service_time,2)} мин")
    event_log.append({
        "event": "WAIT_IN_SCANNER_LINE", "person": person, "scannerLine": scanner_line,
        "time": round(queue_begin, 2), "duration": round(waitq, 2)
    })
```

Используем эти функции для регистрации событий в списки логов (журналирование).
В функции bus_arrival() дополним подчеркнутую строку здесь (курсивом дан написанный ранее код):

```
yield env.timeout(next_bus)

# автобус прибыл, определяем Id для прибывших клиентов для записи в логи

clientIDs = list(range(next_person_id, next_person_id + on_board))
```

```
register_bus_arrival(env.now, next_bus_id, people_ids)
```

next_person_id += on_board

В функции purchasing_customer() дополним подчеркнутую строку здесь (курсивом дан написанный ранее код):

yield env.timeout(rd.gauss(SELLER_MEAN, SELLER_STD))

sale_end = env.now

```
register_group_moving_from_bus_to_seller(people_processed, walk_begin, walk_end, seller_line[1], queue_begin, queue_end, sale_begin,sale_end)
```

env.process(scanning_customer(env, people_processed, scanner_lines, TIME_TO_WALK_TO_SCANNERS_MEAN, TIME_TO_WALK_TO_SCANNERS_STD))

В функции scanning_customer() дополним подчеркнутую строку здесь (курсивом дан написанный ранее код):

scan_begin = env.now

yield env.timeout(abs(rd.gauss(SCANNER_MEAN, SCANNER_STD)))

 # контроль билетов пройден

scan_end = env.now

```
register_visitor_moving_to_scanner(person, walk_begin, walk_end, scanner_line[1], queue_begin, queue_end, scan_begin, scan_end)
```

Запустим модель для проверки.

После запуска модели должен наблюдаться вывод на консоль журнала событий (рисунок 3).

```
Группа клиентов 1 чел ждала 1.99 мин в очереди_6, обслужилась за 1.1 мин
Клиент на контроле ждал 0.76 мин в очереди_3, обслуживался 0.06 мин
Клиент на контроле ждал 1.15 мин в очереди_4, обслуживался 0.08 мин
Клиент на контроле ждал 0.76 мин в очереди_3, обслуживался 0.05 мин
Клиент на контроле ждал 0.95 мин в очереди_2, обслуживался 0.08 мин
Группа клиентов 2 чел ждала 1.66 мин в очереди_2, обслужилась за 1.26 мин
Клиент на контроле ждал 0.92 мин в очереди_1, обслуживался 0.2 мин
Автобус 11 приехал в 33.57 с 74 чел
Клиент на контроле ждал 0.95 мин в очереди_2, обслуживался 0.09 мин
Группа клиентов 2 чел ждала 2.09 мин в очереди_3, обслужилась за 0.9 мин
Клиент на контроле ждал 1.15 мин в очереди_4, обслуживался 0.12 мин
Клиент на контроле ждал 0.92 мин в очереди_1, обслуживался 0.06 мин
Группа клиентов 1 чел ждала 2.19 мин в очереди_5, обслужилась за 0.8 мин
Клиент на контроле ждал 1.24 мин в очереди_4, обслуживался 0.05 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.16 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.09 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.07 мин
Клиент на контроле ждал 1.24 мин в очереди_4, обслуживался 0.1 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.15 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.1 мин
Клиент на контроле ждал 1.17 мин в очереди_4, обслуживался 0.07 мин
Группа клиентов 2 чел ждала 1.92 мин в очереди_4, обслужилась за 1.26 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.05 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.08 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.17 мин
Клиент на контроле ждал 1.1 мин в очереди_3, обслуживался 0.04 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.09 мин
Клиент на контроле ждал 1.17 мин в очереди_4, обслуживался 0.13 мин
```

Рисунок 3. Журнал событий

Фаза 3

Теперь для проведения расчетных экспериментов добавим возможность обмена параметрами между моделью и программой управления экспериментом.
В области объявления глобальных переменных добавим:

```
ОКНО = True    # флаг визуализации журнала модели
```

(замечание: в идентификаторе ОКНО все символы латинские)

Подготовим передачу параметров через аргументы вызова функции модели.

```
def model_env(config=None):
```

Добавим строки объявлений глобальных переменных:

```
    global seller_lines, scanner_lines, event_log
    global SELLER_LINES, SCANNER_LINES, ОКНО
```

В самой функции после строки env = simpy.Environment() добавим код:

```
ОКНО = False  # выводить сообщения в консоль?
finitime = 45
   if config:
      SELLER_LINES = config["num_cashiers"]
      SCANNER_LINES = config["sel_scanners"]
      rd.seed(config["seed_select"])
      finitime = config["simulation_time"]
```

Изменим запуск модели так: `env.run(until = finitime)`

Добавим возвращаемый результат из функции модели:

```
   return event_log, Globals.seller_queues
```

Созданные списки ARRIVALS и ON_BOARD после их создания в коде сохраним для воспроизведения при экспериментировании.
В функции `model_env` добавим в код перед строкой

seller_lines = [simpy.Resource(env, capacity = SELLERS_PER_LINE)

строки

```
   Globals.seller_queues = {v:[] for v in range(SELLER_LINES)}
   Globals.ARRIVALS=Globals.ARRIVAL_ORIGIN.copy()
   Globals.ON_BOARD=Globals.ON_BOARD_ORIGIN.copy()
```

В коде **всех функций** регистрации событий скорректируем выполнение **всех** операторов *print*() с учетом флага ОКНО в виде:

if ОКНО: print(f"Автобус {bus_id+1} приехал в {round(time, 2)} с {len(people_created)} чел")

Запустите для проверки модель как с флагом ОКНО = True, так и ОКНО = False ---

print(model_env())

Сохраним файл с моделью под названием exper_tickets.py

Модель в основном готова для разовых экспериментов. Запускайте модель с разными начальными значениями ГПСЧ random.seed(), попробуйте 3-4 варианта. Убедитесь в различии получаемых результатов, сохраните их в отчет.

Фаза 4

Спроектируем графический интерфейс модели на основе фреймворка Streamlit.

Streamlit – библиотека Python с открытым исходным кодом, которая популярна для проектов в области машинного обучения и анализа данных. Она позволяет публиковать веб-приложения в общем доступе, и включает встроенный веб-сервер с возможностью развертывания в контейнере docker.
Установка Streamlit делается с помощью команды:

pip install streamlit

После установки и запуска такого приложения нужно в браузере перейти по локальному URL localhost:8501, чтобы увидеть Streamlit в действии.

Streamlit позволяет отображать в клиентском браузере данные по-разному:

- st.title() - для установки заголовка;
- st.text() - для записи описания для конкретного графика;
- st.markdown() - для отображения текста в виде markdown;
- st.latex() - для отображения математических выражений на панели мониторинга;
- st.write() - помогает отображать все возможные детали, например, график, фрейм данных, функции, модель и т.д.;
- st.sidebar() - для отображения данных на боковой панели;
- st.dataframe() - для отображения фрейма данных;
- st.map() - для отображения карты и т.п.

С веб-приложением обычно работают так: открываем и изменяем файл Python в редакторе кода Notepad++, SublimeText или Visual Studio Code, и наблюдаем за изменениями в браузере, расположенном рядом с окном редактора, обновляя страничку браузера с веб-модулем приложения.

Обычно основной файл с приложением streamlit называют app.py (но необязательно).
Создадим файл app.py и вставим в него этот код:

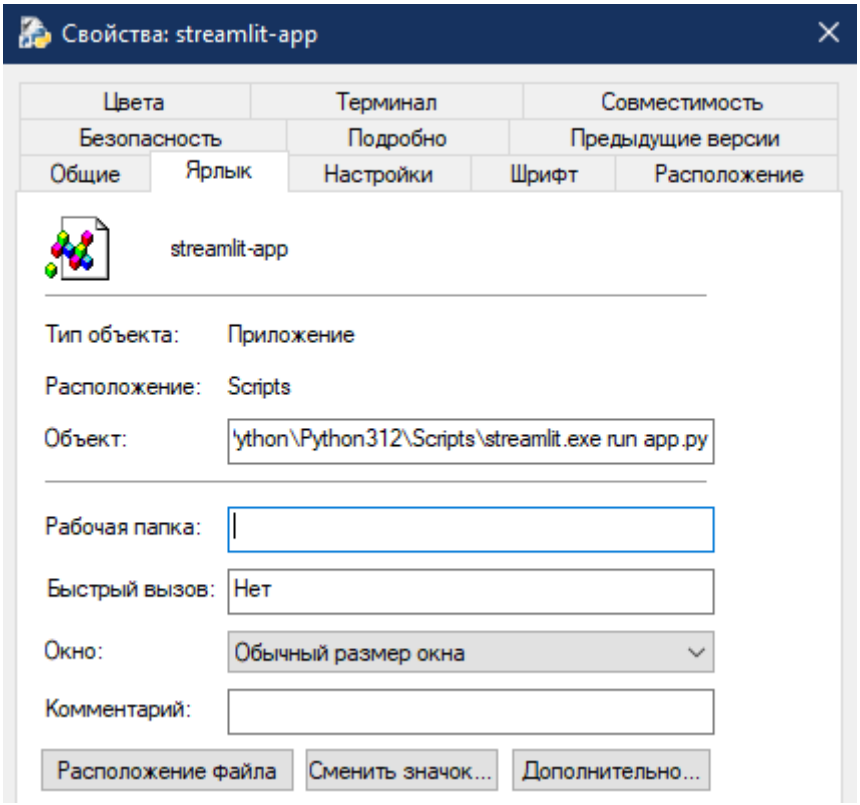
```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Импорт нашей модели simpy
import exper_tickets as mdl

# Функция для запуска симуляции и передачи данных в реальном времени
def run_simulation (params):
    """Запускает моделирование с использованием параметров """
    results = []
    # Запуск модели
    results = mdl.model_env(config=params)
    # возврат результата
    return results

st.set_page_config(
   page_title="моделька",
   page_icon="🦠",
   layout="wide",
   initial_sidebar_state="expanded"
)
```

```
# Интерфейс Streamlit
st.title(":hearts: Моделирование сервиса")

with st.sidebar:
  with st.form(key="input_form"):
    # Параметры модели
    st.header("Параметры модели :clubs:")
    num_cashiers = st.slider("Количество кассиров", 3, 10, 1)
    cashier_processing_time = st.slider(
      "Среднее время обслуживания в кассе (сек)", 30, 100, 10)
    sel_scanners = st.selectbox("Количество контролёров", (4,5,6,7,8,9))
    queue_timeout = st.slider("Среднее время обслуживания на контроле (сек)",15,50,5)
    p_range = ('Magadan', 'Vorkuta', 'UlanUde')
    seed_select = st.radio('seed', p_range)
    simulation_time = st.slider("Время моделирования (мин)", 10, 100, 5)
    st.session_state.pr_click = st.form_submit_button(label=":spades: Запуск модели")

# Подготовка параметров
params = {
    "num_cashiers": num_cashiers,
    "cashier_processing_time": cashier_processing_time,
    "sel_scanners": sel_scanners,
    "queue_timeout": queue_timeout,
    "seed_select": seed_select,
    "simulation_time": simulation_time,
}

# Кнопка для запуска
if st.session_state.pr_click:
    # Контейнеры для графиков и метрик
    progress_placeholder = st.empty()

    # Запуск модели!
    with st.spinner("♻️ Запуск модели..."):
      events, sell_queues = run_simulation(params)
      df1 = pd.DataFrame(events)
      df2 = pd.DataFrame(sell_queues)

      graph1_placeholder = st.empty()
      graph2_placeholder = st.empty()
      st.success("Моделирование завершено!", icon="🗺️")
```

Для запуска приложения Streamlit в ОС Windows используется команда:
C:\Users\USER\AppData\Local\Programs\Python\Python312\Scripts\streamlit.exe run app.py

- здесь вместо USER будет имя вашего пользователя Windows, и вместо Python312 будет название вашей версии Python.

Для удобства запуска модели нужно сделать файл-ярлык streamlit-app.lnk



Запустим модель с помощью ярлыка для проверки работы приложения.

Фаза 5

Теперь подготовим модель exper_tickets.py для использования в статистическом расчетном эксперименте, настроив сбор данных для статистического расчёта с помощью `scipy`.
Добавим такой код после всего кода модели:

```
from scipy import stats
import math

def stat_experimentA():
    global SELLER_LINES, SCANNER_LINES, OKHO
    global seller_waits, scan_waits

    OKHO = False
    # эксперимент_1_
    Level = 4  # кол-во уровней фактора
    numReplica = 11  # кол-во реплик на каждом уровне
    print("_Запуск эксперимента_")
    #_1_
    experiment_ds ={o:[] for o in range(Level)}
    # создадим словарь списков результатов моделирования
    for f in range(Level):
        SELLER_LINES = 4+f
        SCANNER_LINES = 6
        print(f'_уровень ++ {f+1} ++ SELLER_LINES= {SELLER_LINES}')
```

```
        for i in range(numReplica):
        # восстановление нач.значений
            Globals.ARRIVALS=Globals.ARRIVAL_ORIGIN.copy()
            Globals.ON_BOARD=Globals.ON_BOARD_ORIGIN.copy()
            # сброс списков статистики
            arrivals = defaultdict(lambda: 0)
            seller_waits = defaultdict(lambda: [])
            scan_waits = defaultdict(lambda: [])
            Globals.seller_queues = {v:[] for v in range(SELLER_LINES)}
            #
            rd.seed(7321 + i*numReplica + f)  # установка нового случ.зерна
            model_env()  # запуск модели
            if OKHO: print(f'+ {f+1} + реплика_{i+1} +')
            # сохраняем результат прогона
            experiment_ds[f].append(avg_wait(seller_waits)+avg_wait(scan_waits))
if OKHO: print (experiment_ds)
```

Запустим эксперимент, но предварительно закомментируем вызов простого эксперимента:

```
#print(model_env())
OKHO=True

stat_experimentA()
```

Сейчас должен отработать цикл сбора статистики без панели визуализации и с показом по завершении работы состояния словаря experiment_ds.

На втором шаге, добавим блок кода обработки собранной статистики. Для этого будем использовать библиотеку ***scipy.stats***.

```
    #_2_
    print('\n_Сводная описательная статистика_\n фактор SELLER_LINES принимал
значения [4,5,6,7]\n')

    statDescript ={
    'N':[0]*Level,'smm':[0]*Level,'sm':[0]*Level,'sv':[0]*Level,
    'ss':[0]*Level,'sk':[0]*Level,'D_KS':[0]*Level,'pval_KS':[0]*Level,
    'F':[0]*Level,'pval_F':[0]*Level
    }
    for z in range(Level):
        nn, (smin, smax), smn, sv, ss, sk = stats.describe(experiment_ds[z])
        statDescript['N'][z]=nn     # размер выборки
        statDescript['smm'][z]=(round(smin,3), round(smax,3))
        statDescript['sm'][z]=round(smn,3)   # среднее выборочное
        statDescript['sv'][z]=round(sv,3)    # дисперсия выборки
        statDescript['ss'][z]=round(ss,3)    # skew/скос
        statDescript['sk'][z]=round(sk,3)    # kurtosis/эксцесс
        if OKHO: print(f'Датасет{z+1}::{nn}: среднее= {round(smn,3)} | дисперсия
= {round(sv,3)} | min={round(smin,3)} | max={round(smax,3)}')
        # нормализуем датасет перед тестом на нормальность
        zx = (experiment_ds[z] - smn) / math.sqrt(sv)
        # тест Колмогорова-Смирнова на нормальность распределения
        dks, pval = stats.kstest(zx, 'norm')
```

```
        print (f'Датасет{z+1}: KS-статистика: D={round(dks,4)} | p-value =
{round(pval,4)} \n')
        statDescript['D_KS'][z]=round(dks,4)
        statDescript['pval_KS'][z]=round(pval,4)
print(statDescript)
```

Запустим этот эксперимент stat_experimentA().
Сейчас должен отработать цикл сбора статистики без панели визуализации и с показом по завершении работы состояния словаря *experiment_ds* и набора данных сводной описательной статистики по собранным наблюдениям.

На третьем шаге, добавим код расчёта по методу однофакторного дисперсионного анализа, который может показать степень влияния изменчивости фактора на результат.

```
    #_3_
    print("\n_Расчет однофакторного дисперсионного анализа ANOVA (alpha=
0.05)_\n")

    F, pval = stats.f_oneway(experiment_ds[0],experiment_ds[1],experiment_ds[2],
experiment_ds[3])
    print (f'значение критерия F: {round(F,4)} | p-value: {round(pval,4)}')
    statDescript['F'][0]=round(F,4)
    statDescript['pval_F'][0]=round(pval,4)

    return statDescript
```

Модель с экспериментальным исследованием подготовлена.
Запустим этот эксперимент print(stat_experimentA()).

Теперь уточните по рассчитанной дисперсии количество прогонов (реплик) для обеспечения качества оценки на уровне 95% с точностью 0,5. Внесите изменения в количество прогонов и повторите уточнённый расчётный эксперимент.

Фаза 6

Добавим построение графиков с использованием библиотеки matplotlib в файл приложения app.py. Добавим такой код с исправлением блока запуска модели:

```
    # Запуск модели!
    with st.spinner("♻ Запуск модели..." ):
      events, sell_queues = run_simulation(params)
      df1 = pd.DataFrame(events)
      df2 = pd.DataFrame(sell_queues)

      graph1_placeholder = st.empty()
      graph2_placeholder = st.empty()
      st.success("Моделирование завершено!", icon="🗺")

    fig, ax = plt.subplots(figsize=(8,6), layout='constrained', facecolor='0.7')
```

```
    for sc in df2.columns:
       ax.step(df2.index, df2[sc], label=f'Q_{sc}')
    ax.legend(loc='upper left') #, borderaxespad=0.1) #
    # Обновление графика
    graph1_placeholder.scatter_chart(df1[["person"]])
    graph2_placeholder.pyplot(fig)

    # Итоговые метрики
    st.write("Итоговые метрики:")
    st.write(df1)
    st.dataframe(df2)

    st.download_button(
        label=":diamonds: Скачать результаты",
        data=df1.to_csv(index=False),
        file_name="result.csv", mime="text/csv"
    )

if statest :
   with st.spinner('Выполняется...'):
      results = mdl.stat_experimentA()
      t_df = pd.DataFrame(results)
      st.write(t_df)
      ff = t_df[['F'][0]]
      fp = t_df[['pval_F'][0]]

   st.metric("критерий F", ff[0], fp[0])

   if fp[0]>0.05:
      st.markdown("**Гипотеза H0 не отклоняется!**")
   else:
      st.markdown("**Гипотеза H0 отклоняется!**")
   st.success(" Статистический анализ завершен!", icon="🗺")
```

Проведите **свой** эксперимент по оценке влияния другого параметра модели SCANNER_LINES на тот же результат.
Сравните с результатом первого эксперимента, сделайте письменно в отчете выводы.

Пришлите код программы модели и отчёт о проведённых экспериментах на проверку преподавателю.

Примечание 1:

Односторонний анализ ANOVA использует следующие нулевые и альтернативные гипотезы:
H0 (нулевая гипотеза): μ1 = μ2 = μ3 = ... = *μk* (все средние значения совокупности равны).
H1 (альтернативная гипотеза): по крайней мере, одно среднее значение популяции отличается от остальных.

Если рассчитанное значение *p-value* менее 0,05, то можно отвергнуть нулевую гипотезу. Это означает, что у нас достаточно оснований, чтобы утверждать, что *существует* статистически значимая разница между выборками.
Если отвергается нулевая гипотеза, это значит, что, по крайней мере, одно из средних значений совокупности отличается от других, но в ANOVA не указано, какие средние значения совокупности отличаются. Если нужно определить это, то необходимо выполнить специальные (post hoc) тесты, также известные как тесты множественных сравнений, например, тест Тьюки, метод Хольма, тест Даннета.
Сейчас по плану эксперимента нам этого делать не нужно.

Примечание 2:

Для настройки веб-приложения streamlit можно изменить конфигурацию через файл config.toml

Пример файла config.toml:

```
[server]
folderWatchBlacklist = []
fileWatcherType = "auto"
headless = false
runOnSave = false
port = 8501
[client]
showErrorDetails = "none"
showSidebarNavigation = true
toolbarMode = "auto"
[runner]
magicEnabled = true
fastReruns = true
enforceSerializableSessionState = false
enumCoercion = "nameOnly"
[browser]
serverAddress = "localhost"
gatherUsageStats = true
serverPort = 8501
[theme]
# This can be one of the following: "light" or "dark"
base = "light"
primaryColor = "#263543"
backgroundColor = "peachpuff"
```