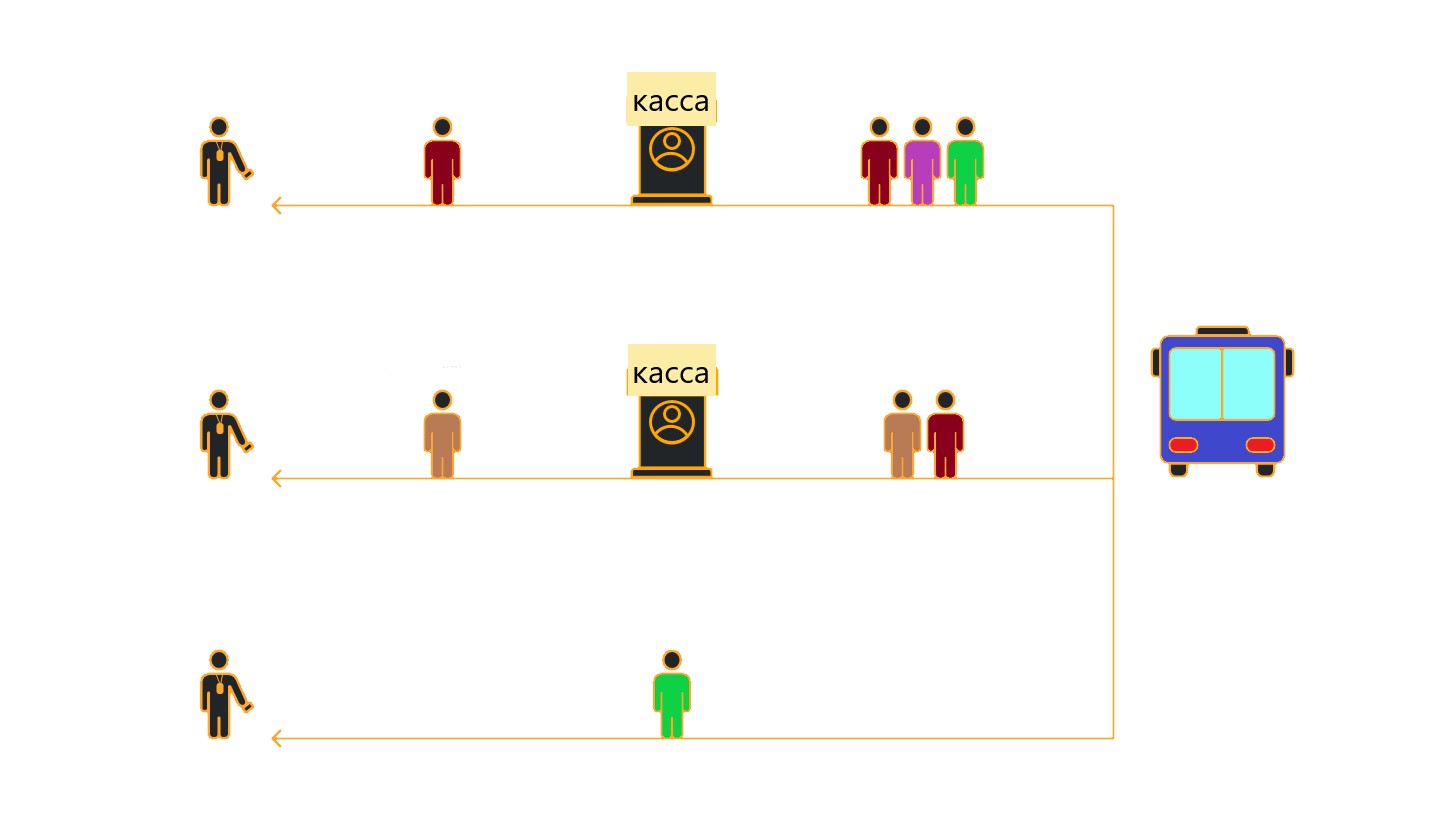
## **Практическая работа «Выполнение экспериментов с имитационной моделью на языке Python»**

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 знакомым библиотекам Python, таким как Matplotlib и tkInter для обеспечения построения графиков и визуализации процесса соответственно.

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

Мы будем моделировать ситуацию входного потока, который полностью обслуживается общественным транспортом: 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.

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

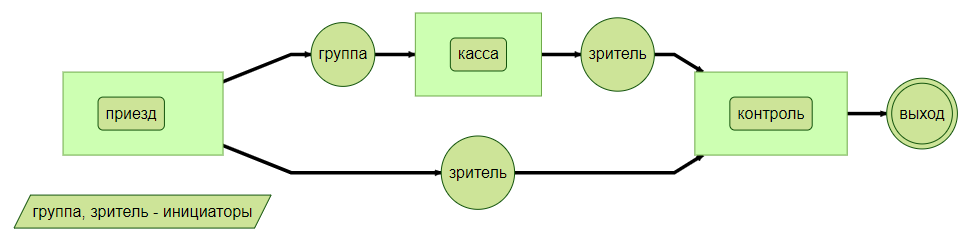
* Автобус прибывает в среднем 1 раз в 3 минуты.
* В каждом автобусе будут находиться от 35 до 125 посетителей, что моделируется с использованием нормального распределения (μ = 80, σ = 15).
* Посетители будут формировать группы из 2,5 ± 1,5 человек, что смоделируем используя норм-альное распределение (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).

Модель сделаем в форме консольного приложения. Для подготовки модели нам потребуется Python с дополнительными фреймворками **simpy, numpy, matplotlib, scipy**. Библиотеки random, collections, tkinter, math обычно входят в дистрибутив Python.

**Фаза 1.**

Рассмотрим логическую схему процесса в виде такой модели:



Начнём разработку модели с базового набора процессов и ресурсов.

Объявим 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*

Для наблюдения запустим модель на 45 единиц модельного времени (т.е. 45 минут реального процесса). Для написания кода можно использовать любой типовой редактор – IDLE, Visual Studio Code, PyCharm, Spyder и т.п. Начнём формирование модели с добавления в программу нижерасположенного кода.

# -\*- coding: utf-8 -\*-

import random as rd

import simpy

# ----------------------------

# Параметры конфигурации модели 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 = 0.50

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

# предварительно запишем времена прибытия автобусов и кол-во пассажиров

# чтобы точно управлять случайностью

rd.seed(4210)

ARRIVALS = [rd.expovariate(1 / BUS\_ARRIVAL\_MEAN) for \_ in range(22)]

ON\_BOARD = [abs(int(rd.gauss(BUS\_OCCUPANCY\_MEAN, BUS\_OCCUPANCY\_STD))) for \_ in range(22)]

# -------------------------

# 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, от которого срабатывают другие события

"""

# уникальные ID для автобуса и людей нужны для последующей визуализации

next\_bus\_id = 0

next\_person\_id = 0

while True:

next\_bus = ARRIVALS.pop()

on\_board = ON\_BOARD.pop()

# ждать следующий автобус

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))

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

# определяем через Random - надо купить билет или сразу идти 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):

"""

моделируем приход клиентов в кассу, клиент тратит -

время TIME\_TO\_WALK\_TO\_SELLERS\_MEAN на подход к кассе от автобуса

время SELLER\_MEAN на обслуживание в кассе

"""

# подойти к кассе

walk\_begin = env.now

yield env.timeout(abs(rd.gauss(TIME\_TO\_WALK\_TO\_SELLERS\_MEAN, TIME\_TO\_WALK\_TO\_SELLERS\_STD)))

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

# обслуживание началось

queue\_end = env.now

# купить билеты

sale\_begin = env.now

yield env.timeout(rd.gauss(SELLER\_MEAN, SELLER\_STD))

# билеты куплены

sale\_end = env.now

# начать процесс контроля билета

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):

"""

моделируем приход клиентов на контроль билетов, клиент тратит -

время walk\_duration на подход к контролю,

время SCANNER\_MEAN на обслуживание на контроле

"""

# подойти к контролеру

walk\_begin = env.now

yield env.timeout(abs(rd.gauss(walk\_duration, walk\_std)))

walk\_end = env.now

# встать в очередь

queue\_begin = env.now

# клиент всегда выбирает самую короткую очередь

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)))

# контроль билетов пройден

scan\_end = env.now

# 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.**

Модель работает, но результатов не показывает и не собирает.

**Фаза 2.**

Теперь добавим функции сбора статистики для наблюдения результатов.

Добавим в начале кода импортирование из модуля collections:

from collections import defaultdict

Добавим нижерасположенный код после строки с random.seed перед разделом кода # SIMULATION.

# -------------------------

# ANALYTICAL GLOBALS

# списки и словари для аналитики

event\_log = []

arrivals = defaultdict(lambda: 0)

seller\_waits = defaultdict(lambda: [])

scan\_waits = defaultdict(lambda: [])

seller\_queues = {v:[] for v in range(SELLER\_LINES)}

# методы и функции

def avg\_wait(raw\_waits):

waits = [ w for i in raw\_waits.values() for w in i ]

return round(sum(waits)/len(waits), 1) if len(waits) > 0 else 0

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")

def register\_group\_moving\_from\_bus\_to\_seller(people,walk\_begin,walk\_end,seller\_line,queue\_begin,queue\_end,sale\_begin,sale\_end):

wait = queue\_end - queue\_begin

service\_time = sale\_end - sale\_begin

seller\_waits[int(queue\_end)].append(wait)

print(f"Группа клиентов {len(people)} чел ждала {round(wait,2)} мин в очереди\_{seller\_line}, обслужилась за {round(service\_time,2)} мин")

def register\_visitor\_moving\_to\_scanner(person, walk\_begin, walk\_end, scanner\_line, queue\_begin, queue\_end, scan\_begin, scan\_end):

wait = queue\_end - queue\_begin

service\_time = scan\_end - scan\_begin

scan\_waits[int(queue\_end)].append(wait)

print(f"Клиент на контроле ждал {round(wait,2)} м в очереди\_{scanner\_line}, обслуживался {round(service\_time,2)} м")

Используем эти функции для регистрации событий в списки логов.

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, clientIDs)

*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)

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\_queues,seller\_lines

while True:

# запомним текущую длину очередей

for i in range(len(seller\_lines)): seller\_queues[i].append(len(seller\_lines[i].queue))

yield ev.timeout(1.0)

Активируем этот процесс перед запуском модельного процесса.

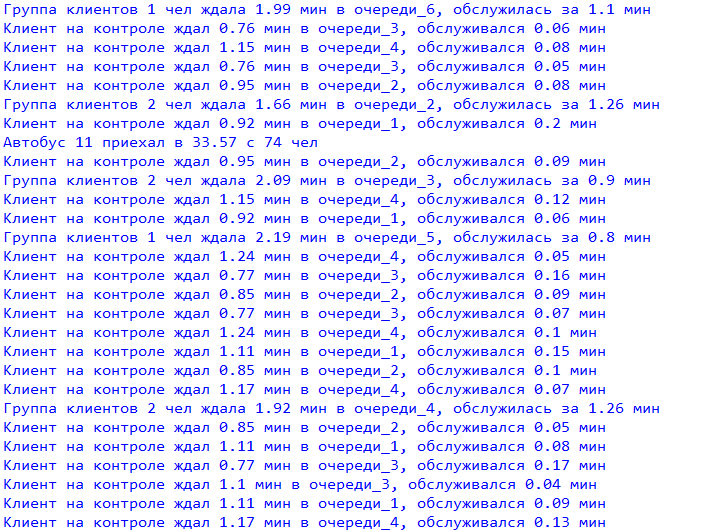
*env.process(bus\_arrival(env, seller\_lines, scanner\_lines))*

env.process(monitor(env))

*env.run(until = 45)*

**Запустим модель для проверки.**

После запуска модели должен наблюдаться вывод на консоль журнала событий.



**Фаза 3.**

Добавим графический интерфейс модели на основе библиотеки ***tkInter***.

Добавим импорт в начале программы:

import tkinter as tk

Инициализируем глобальный объект canvas в строке перед random seed().

mainpanel = tk.Tk()

Затем добавим нижерасположенный код перед блоком функций модели.

Этот код создает информационную панель (окно *canvas*) с настройками для визуализации процесса моделирования.

# -------------------------

# UI/ANIMATION

# -------------------------

def infoPanel():

global bus\_log, sellers, scanners, clockinfo, mainpanel

mainpanel.title("Nash Sim Proc")

mainpanel.config(bg="#fff")

top\_frame = tk.Frame(mainpanel)

top\_frame.pack(side=tk.TOP, expand = False)

canvas = tk.Canvas(mainpanel, width = 1200, height = 450, bg = "white")

canvas.pack(side=tk.TOP, expand = False)

class QueueGraphics:

text\_height = 30

def \_\_init\_\_(self, icon, icon\_width, queue\_name, num\_lines, canvas, x\_top, y\_top):

self.icon\_file = icon + ".gif"

self.icon\_width = icon\_width

self.queue\_name = queue\_name

self.num\_lines = num\_lines

self.canvas = canvas

self.x\_top = x\_top

self.y\_top = y\_top

dataimg = "R0v"

if icon == "group":

dataimg= <*вставьте сюда код картинки из Приложения А ! одной строкой! без разрывов!*>

if icon == "person":

dataimg= <*вставьте сюда код картинки из Приложения Б ! одной строкой! без разрывов!*>

self.image = tk.PhotoImage(data=dataimg)

self.icons = defaultdict(lambda: [])

for i in range(num\_lines):

canvas.create\_text(x\_top, y\_top + (i \* self.text\_height), anchor = tk.NW, text = f"{queue\_name} {i+1}")

self.canvas.update()

def add\_to\_line(self, service\_number):

count = len(self.icons[service\_number])

x = self.x\_top + 62 + (count \* self.icon\_width)

y = self.y\_top + ((service\_number -1) \* self.text\_height) -8

self.icons[service\_number].append( self.canvas.create\_image(x, y, anchor = tk.NW, image = self.image) )

self.canvas.update()

def remove\_from\_line(self, service\_number):

if len(self.icons[service\_number]) == 0: return

to\_del = self.icons[service\_number].pop()

self.canvas.delete(to\_del)

self.canvas.update()

def Sellers(canvas, x\_top, y\_top):

return QueueGraphics("group", 24, "Касса", SELLER\_LINES, canvas, x\_top, y\_top)

def Scanners(canvas, x\_top, y\_top):

return QueueGraphics("person", 17, "Контроль", SCANNER\_LINES, canvas, x\_top, y\_top)

class BusLog:

TEXT\_HEIGHT = 24

def \_\_init\_\_(self, canvas, x\_top, y\_top):

self.canvas = canvas

self.x\_top = x\_top

self.y\_top = y\_top

self.bus\_count = 0

def next\_bus(self, minutes):

x = self.x\_top

y = self.y\_top + (self.bus\_count \* self.TEXT\_HEIGHT)

self.canvas.create\_text(x, y, anchor = tk.NW, text = f"Прибытие через {round(minutes, 1)} мин")

self.canvas.update()

def bus\_arrived(self, people):

x = self.x\_top + 150

y = self.y\_top + (self.bus\_count \* self.TEXT\_HEIGHT)

self.canvas.create\_text(x, y, anchor = tk.NW, text = f"Привез {people} чел", fill = "blue")

self.bus\_count += 1

self.canvas.update()

bus\_log = BusLog(canvas, 842, 20)

sellers = Sellers(canvas, 428, 20)

scanners = Scanners(canvas, 42, 20)

Теперь будем использовать классы графического интерфейса для отображения состояния очередей.

В функции **bus\_arrival()** дополним подчеркнутые строки для отображения в блоке ожидания (курсивом дан написанный ранее код):

*# ждать следующий автобус*

bus\_log.next\_bus(next\_bus)

*yield env.timeout(next\_bus)*

bus\_log.bus\_arrived(on\_board)

В функции **purchasing\_customer()** дополним подчеркнутые строки отображения в блоке покупки (курсивом дан написанный ранее код):

*with seller\_line[0].request() as req:*

*# Wait in line*

sellers.add\_to\_line(seller\_line[1])

*yield req*

sellers.remove\_from\_line(seller\_line[1])

*queue\_end = env.now*

В функции **scanning\_customer()** дополним подчеркнутые строки для отображения в блоке контроля (курсивом дан написанный ранее код):

*with scanner\_line[0].request() as req:*

*# подождать в очереди*

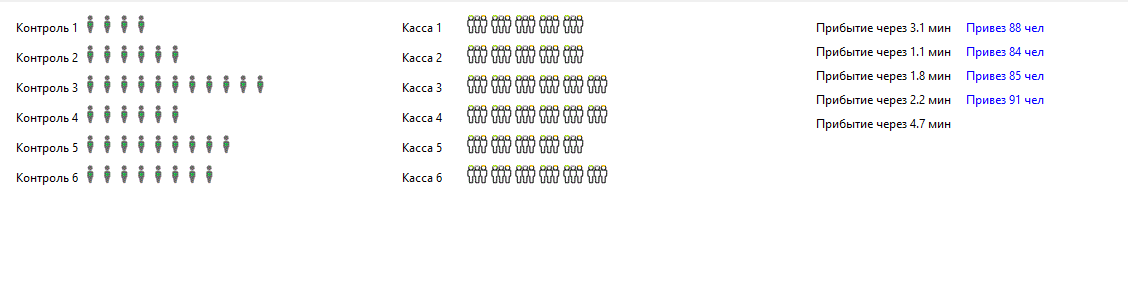
for \_ in people\_processed: scanners.add\_to\_line(scanner\_line[1])

*yield req*

for \_ in people\_processed: scanners.remove\_from\_line(scanner\_line[1])

*queue\_end = env.now*

Добавим вызов функции в основной блок модели model\_env() infoPanel() перед env.run(until = 45).



**Запустим модель для проверки.**

**Фаза 4.**

Теперь добавим информационные панели с динамическим отображением модельного времени и основных характеристик очередей.

Сделаем объявление этих элементов в начале программы в разделе CONFIGURATION.

Это нужно для их реализации на уровне глобальных переменных.

seller\_lines, scanner\_lines = [],[]

bus\_log, sellers, scanners, clockinfo = 0,0,0,0

Добавим нижерасположенный код после объявления класса BusLog и перед инстанцированием классов bus\_log = BusLog().

class ClockData:

def \_\_init\_\_(self, canvas, x1, y1, x2, y2, time):

self.x1 = x1; self.y1 = y1

self.x2 = x2; self.y2 = y2

self.tot\_q\_list = {}

self.canvas = canvas

self.train = canvas.create\_rectangle(self.x1, self.y1, self.x2, self.y2, fill="lavender")

self.options = {"font": "Arial 10", "fill": "red", "activefill": "blue", "anchor": "nw"}

self.time = canvas.create\_text(self.x1 + 10, self.y1 + 10, text = f"ВРЕМЯ = {round(time, 1)}m", \*\*self.options)

self.seller\_wait = canvas.create\_text(self.x1+10, self.y1+30, text ="сред.ожидание в кассу = ", \*\*self.options)

self.scan\_wait = canvas.create\_text(self.x1+10, self.y1+50, text ="сред.ожидание на контроле= ", \*\*self.options)

self.seller\_count = canvas.create\_text(self.x1+10, self.y1+70, text="очереди в кассу: суммарно=",\*\*self.options)

self.scan\_count = canvas.create\_text(self.x1 + 10, self.y1 + 90, text = ">>>", \*\*self.options)

self.canvas.update()

def tick(self, time):

# метод обновления данных по вызову из процесса clockTablo

for z in [self.time,self.seller\_wait,self.scan\_wait,self.seller\_count,self.scan\_count]:

self.canvas.delete(z)

# подготовка суммарной информации по очередям

self.seller\_que\_str =''; self. total\_queues =0

for i in range(SELLER\_LINES):

self.seller\_que\_str += str(i+1)+'K>>'+str(seller\_queues[i][int(time)])+' '

self.total\_queues += seller\_queues[i][int(time)]

self.tot\_q\_list[int(time)] = self.total\_queues

self.total\_queues = max(list(self.tot\_q\_list.values()))

# вывод на канвас текста состояния характеристик процесса

self.time = canvas.create\_text(self.x1 + 10, self.y1 + 10,

text = f"ВРЕМЯ ::: {round(time, 1)} мин :::", \*\*self.options )

self.seller\_wait = canvas.create\_text(self.x1 + 10, self.y1 + 30,

text = f"сред.ожидание в кассу = {avg\_wait(seller\_waits)}м", \*\*self.options)

self.scan\_wait = canvas.create\_text(self.x1 + 10, self.y1 + 50,

text = f"сред.ожидание на контроле= {avg\_wait(scan\_waits)}м", \*\*self.options)

self.seller\_count = canvas.create\_text(self.x1 + 10, self.y1 + 70,

text = f"очереди в кассу: суммарно(макс)= {self.total\_queues}", \*\*self.options)

self.scan\_count = canvas.create\_text(self.x1+10, self.y1+90, text=self.seller\_que\_str[::-1], \*\*self.options)

self.canvas.update()

**Добавим код инстанцирования этих классов.**

bus\_log = BusLog(canvas, 842, 20)

sellers = Sellers(canvas, 428, 20)

scanners = Scanners(canvas, 42, 20)

clockinfo= ClockData(canvas, 56,284,345,421,0)

**Добавим процесс обновления данных инфопанели на созданном интерфейсе, например, вставив его перед функцией monitor(ev).**

# этот генератор используется для обновления данных и часов на интерфейсе UI

def clockTablo(env):

global clockinfo

while True:

yield env.timeout(0.1)

clockinfo.tick(env.now)

**Активируем этот процесс в коде функции запуска модели model\_env() .**

*infoPanel()*

env.process(clockTablo (env))

*env.run(until = 45)*

*mainpanel.mainloop() # может понадобится для остановки изображения панели после выполнения программы*

**Запустим модель для проверки.**

Добавим построение графиков с использованием библиотеки matplotlib, но учитывая наличие окна tkinter, подсоединим панель графиков в canvas с помощью дополнительного модуля FigureCanvasTkAgg.

Визуализация с помощью tkinter является наиболее простым программным методом презентации, хотя в нашем случае, усложненным включением поддержки matplotlib.

Добавим импорт matplotlib:

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

**Затем в разделе UI/ANIMATION в функции infopanel() после строки:**

canvas.pack(side=tk.TOP, expand = False)

**добавим блок кода подготовки панелей графиков:**

f = plt.Figure(figsize=(2, 2), dpi=72)

a3 = f.add\_subplot(221)

a3.plot()

a1 = f.add\_subplot(222)

a1.plot()

a4 = f.add\_subplot(223)

a4.plot()

a2 = f.add\_subplot(224)

a2.plot()

data\_plot = FigureCanvasTkAgg(f, master=mainpanel)

data\_plot.get\_tk\_widget().config(height = 400)

data\_plot.get\_tk\_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

**Затем в методе tick() класса ClockData перед строкой self.canvas.update() добавим:**

# вывод графиков динамически по мере получения результатов

a1.cla()

a1.set\_xlabel("время (мин)")

a1.set\_ylabel("сред.ожидание в кассу")

a1.plot(seller\_waits.keys(),[sum(waits)/len(waits) for (t,waits) in seller\_waits.items()])

a2.cla()

a2.set\_xlabel("время (мин)")

a2.set\_ylabel("сред.ожидание на контроле")

a2.plot(scan\_waits.keys(),[sum(waits)/len(waits) for (t,waits) in scan\_waits.items()])

a3.cla()

a3.set\_xlabel("время")

a3.set\_ylabel("прибытие")

a3.bar(arrivals.keys(), arrivals.values(), color='salmon')

a4.cla()

a4.set\_xlabel("время (мин)")

a4.set\_ylabel("общая очередь в кассу")

a4.step(self.tot\_q\_list.keys(), self.tot\_q\_list.values())

data\_plot.draw()

**Запустим модель для проверки работы.**



Модель в основном готова для наблюдения и добавления экспериментов. Запускайте модель с разными начальными значениями ГПСЧ random.seed(), попробуйте 4 варианта. Убедитесь по графикам в различии получаемых результатов, сохраните их в отчет.

**Фаза 5.**

Теперь подготовим модель для использования в статистическом расчётном эксперименте.

Для этого понадобится отключать визуализацию, сохранять данные моделирования от нескольких прогонов, выполнить обработку собранной статистики методами статистического анализа.

В области объявления глобальных переменных добавим флаг:

*seller\_lines, scanner\_lines = [],[]*

*bus\_log, sellers, scanners, clockinfo = 0,0,0,0*

OKHO = True # флаг визуализации модели

*mainpanel = tk.Tk()*

Созданные списки ARRIVALS и ON\_BOARD сразу после их создания в коде сохраним для воспроизведения при экспериментировании.

ARRIVAL\_ORIGIN = ARRIVALS.copy() # сохраним оригинальные списки для

ON\_BOARD\_ORIGIN= ON\_BOARD.copy() # повторного использования в эксперименте

**Вставим код заглушки инфопанели для использования его во время эксперимента.**

**Добавим нижерасположенный код перед кодом описания раздела UI/ANIMATION.**

# код заглушки вместо инфопанели для эксперимента

def dummyPanel():

global bus\_log,sellers,scanners

class BusLog:

def \_\_init\_\_(self):

pass

def next\_bus(self, minutes):

pass

def bus\_arrived(self, people):

pass

class Sellers:

def \_\_init\_\_(self):

pass

def add\_to\_line(self, seller\_number):

pass

def remove\_from\_line(self, seller\_number):

pass

class Scanners:

def \_\_init\_\_(self):

pass

def add\_to\_line(self, seller\_number):

pass

def remove\_from\_line(self, seller\_number):

pass

bus\_log = BusLog()

sellers = Sellers()

scanners = Scanners()

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

***if OKHO***: print(f"Автобус {bus\_id+1} приехал в {round(time, 2)} с {len(people\_created)} чел")

**Код основного метода запуска модели представим в виде:**

def model\_env():

global seller\_lines,scanner\_lines,OKHO

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.process(monitor(env))

if OKHO:

infoPanel()

env.process(clockTablo(env))

else:

dummyPanel()

env.run(until = 45)

if OKHO: mainpanel.mainloop()

***Теперь функция запуска модели будет выглядеть так:***

def main(variant = 0):

global OKHO

### простой запуск модели

if variant == 1:

OKHO = True # выводить окно визуализации процесса!

model\_env()

***Запустите для проверки модель как с флагом OKHO = True, так и OKHO = False, сделав вызов main(1).***

**Фаза 6.**

Теперь подготовим функцию сбора данных и статистического эксперимента.

На первом шаге, добавим после всех операторов программы блок кода организации цикла сбора данных статистики.

def stat\_experimentA():

global ARRIVALS,ARRIVAL\_ORIGIN,ON\_BOARD,ON\_BOARD\_ORIGIN,OKHO

global SELLER\_LINES,SCANNER\_LINES,seller\_waits,scan\_waits,seller\_queues

#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

ARRIVALS=ARRIVAL\_ORIGIN.copy()

ON\_BOARD=ON\_BOARD\_ORIGIN.copy()

# сброс списков статистики

arrivals = defaultdict(lambda: 0)

seller\_waits = defaultdict(lambda: [])

scan\_waits = defaultdict(lambda: [])

seller\_queues = {v:[] for v in range(SELLER\_LINES)}

#

rd.seed(7321 + i\*numReplica + f) # установка нового случ.зерна

model\_env() # запуск модели

print(f'+ {f+1} + реплика\_{i+1} +')

experiment\_ds[f].append(avg\_wait(seller\_waits)+avg\_wait(scan\_waits))

# сохраняем результат прогона

print (experiment\_ds)

**Изменим функцию main() c учетом запуска эксперимента:**

def main(variant = 0):

global OKHO

### простой запуск модели

if variant == 1:

OKHO = True # выводить окно визуализации процесса!

model\_env()

elif variant == 2:

OKHO = False # не выводить окно визуализации процесса!

mainpanel.destroy()

stat\_experimentA()

else:

print ("старт main(1) или main(2)")

Запустим эксперимент вызовом main(2). Сейчас должен отработать цикл сбора статистики без панели визуализации и с показом по завершении работы состояния словаря experiment\_ds.

На втором шаге, добавим далее после print (experiment\_ds) блок кода обработки собранной статистики. Для этого будем использовать библиотеку ***scipy***.

#\_2\_

print('\n\_Сводная описательная статистика\_\n')

print('фактор SELLER\_LINES принимал значения [4,5,6,7]')

from scipy import stats

import math

# подготовим коллекцию для сохранения результатов эксперимента

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}

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]=(smin, smax)

statDescript['sm'][z]=smn # среднее выборочное

statDescript['sv'][z]=sv # дисперсия выборки

statDescript['ss'][z]=ss # skew/скос

statDescript['sk'][z]=sk # kurtosis/эксцесс

print(f'Датасет{z}::{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)

# тестКолмогорова**-**Смирнова на нормальность распределения

dks, pval = stats.kstest(zx, 'norm')

print (f'Датасет{z}: KS-статистика: D={round(dks,4)} | p-value= {round(pval,3)}')

statDescript['D\_KS'][z]=dks

statDescript['pval\_KS'][z]=pval

print(statDescript)

Запустим этот эксперимент через main(2). 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)}')

Модель с экспериментальным исследованием подготовлена.

Для формирования правильных оценок характеристик обслуживания скорректируйте окончание моделирования по завершении обслуживания всех посетителей, но ограничив 45 минутами прибытие автобусов.

Теперь уточните по рассчитанной дисперсии количество прогонов (реплик) для обеспечения качества оценки ожидания на уровне 95% с точностью 0,25. Внесите изменения в количество прогонов и повторите уточнённый расчётный эксперимент.

***Примечание:***

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.

В нашем случае непринятие гипотезы ***Н0*** означает, что вносимые изменением фактора SELLER\_LINES воздействия *существенно* изменяют значение установленного критерия – суммарного времени ожидания в очередях (касса и контроль).

Если отвергается нулевая гипотеза, это значит, что, по крайней мере, одно из средних значений совокупности отличается от других, но метод ANOVA не указывает, 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.*

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.

Приложение А. Изображение группы людей

""

Приложение Б. Изображение человечка

""