МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторной работе №3

по курсу «Моделирование информационно-вычислительных систем»

Выполнил:

Медов Д.А.

гр.6303

Проверила:

Симонова Е.В.

Самара 2021

Задание

Разработать событийную модель включающее в себя интеррогативное управление с использованием сигнальной синхронизации в сочетании с императивным управлением без приоритетов уведомлений.

Основные задачи событийной модели:

1. Связать наступление события EVi с подачей сигнала Sj (ожидание подачи сигнала Sj).

2. Подать сигнал Sj.

3. Сбросить сигнал Sj.

4. Включить в календарь уведомление о событии EVi в абсолютном времени.

5. Запланировать выполнение действий, соответствующих событиям, до тех пор, пока календарь событий не пуст и в системе имеется хотя бы один активный сигнал, подачи которого ожидает хотя бы одно уведомление.

Описание реализации дискретно-событийного монитора

Дискретно-событийный монитор представляет собой цикл, в котором отслеживается события и в зависимости от них происходит изменение состояний дескрипторов. Дискретно событийный монитор реализован в методе «thread\_modeling».

Класс «Visitor» описывает объект дескриптора посетителя. При его инициализации создается новый посетитель, к которому присваивается изображение посетителя и временные диапазоны заказа блюда, и время обработки заказа.

Дескриптор посетителя имеет три состояния: «выполнение заказа», «ожидание заказа» и «обработка заказа». При завершении одного состояния дескриптор переходит к следующему в порядке от «выполнение заказа» до «обработка заказа», если событие «обработка заказа» завершилось, то дескриптор переходит к событию «выполнение заказа».

Состояние «выполнение заказа» реализовано с помощью двух методов: «init\_order» и «issue\_order». Метод «init\_order» инициализирует начала события «выполнение заказа», в нем создается объект «заказ» и время, за которое будет происходить это событие.

Метод «issue\_order» проверяет модельное время и время завершения события «выполнение заказа». Если время завершение события наступило, то метод возвращает объект заказа и переходит к следующему состоянию «ожидание заказа».

Для состояния «ожидание заказа» реализован один метод «check\_order», который проверяет готовый заказ c заказом, который сделал посетитель. Если объекты равны, то дескриптор посетителя переходит к следующему состоянию «обработка заказа».

К состоянию «обработка заказа» относятся два метода «init\_eat» и «issue\_eat». Метод «init\_eat» инициализирует начала обработки заказа и время выполнения события. Метод «issue\_eat» проверяет модельное время и время завершения события, если время завершения события наступило, то дескриптор посетителя переходит к состоянию «выполнение заказа».

Класс «Cook» описывает объект дескриптора повара. При его инициализации создается новый дескриптор, к которому создается очередь, куда попадают заказы на выполнение.

Когда дескриптор посетителя завершает событие «выполнение заказа», он возвращает объект заказ, которые передается дескриптору повару и попадает в очередь заказов с помощью метода «add\_order», не зависимо от текущего состояния дескриптора.

Дескриптор повара имеет два состояния «ожидание заказа от посетителя» и «выполнение заказа посетителя».

При состоянии «ожидание заказа от посетителя» проверяется очередь на появление нового заказа, если такой заказ появился выполняется метод «cook\_order». Он удаляет из очереди заказ, и устанавливает время приготовления заказа.

Метод «issue\_order» проверяет время приготовления заказа с модельным временем. Если они равны, то заказ считается выполненным, и возвращает объект готового заказа и дескриптор повара переходит к состоянию «ожидание заказа от посетителя».

Класс «Order» описывает объект сигнала. При инициализации нового сигнала, объекту присваивается тип еды, диапазон времени ее приготовления и номер заказа.

Метод «init\_cooking» задает время приготовления еды, а метод «check\_cooking» проверят ее готовность.

Дискретно-событийная модель считается завершенной, если состояние дескриптора повара соответствует «ожидание заказа от посетителя», а состояния дескрипторов посетителей соответствует «выполнение заказа».

Листинг программы

Файл main.py:

from tkinter import font

from transitions import Machine

from tkinter import \*

from PIL import Image, ImageTk

import time

import threading

import visitor

import cook

import order

SIZE = 128

TIME\_LIMIT = 500

states\_visitor = ['order', 'wait', 'eat']

states\_cook = ['wait', 'cook']

transitions\_visitor = [

    { 'trigger': 'set\_order', 'source': 'order', 'dest': 'wait' },

    { 'trigger': 'get\_order', 'source': 'wait', 'dest': 'eat' },

    { 'trigger': 'eat\_up', 'source': 'eat', 'dest': 'order' }]

transitions\_cook = [

    { 'trigger': 'get\_order', 'source': 'wait', 'dest': 'cook' },

    { 'trigger': 'set\_order', 'source': 'cook', 'dest': 'wait' }]

def thread\_modeling():

    time\_model = 0

    num\_order = 0

    order\_ready = None

    lbxLogs.delete(0, END)

    visitors = [

        visitor.Visitor(cvMan1, cvState1),

        visitor.Visitor(cvMan2, cvState2),

        visitor.Visitor(cvMan3, cvState3)]

    for vistr in visitors:

        Machine(vistr, states=states\_visitor, transitions=transitions\_visitor, initial='order')

        vistr.init\_order(time\_model, num\_order)

        num\_order += 1

    chef = cook.Cook(cvKitchen, cvState4)

    Machine(chef, states=states\_cook, transitions=transitions\_cook, initial='wait')

    while len(list(filter(lambda vistr: vistr.state == "eat", visitors))) < 3 and TIME\_LIMIT > time\_model:

        for vistr in visitors:

            if vistr.state == "order":

                ordr = vistr.issue\_order(time\_model)

                if ordr != None:

                    vistr.set\_order()

                    logs = "[{}] {} сделал(а) Заказ №{} : {}".format(time\_model, vistr.visitor, ordr.number\_order, ordr.order)

                    lbxLogs.insert(0, logs)

                    chef.add\_order(ordr)

            if vistr.state == "wait":

                if vistr.check\_order(order\_ready):

                    vistr.init\_eat(time\_model)

                    vistr.get\_order()

                    logs = "[{}] {} получил(а) Заказ №{} : {}".format(time\_model, vistr.visitor, vistr.order.number\_order, vistr.order.order)

                    lbxLogs.insert(0, logs)

            if vistr.state == "eat":

                if vistr.issue\_eat(time\_model):

                    vistr.init\_order(time\_model, num\_order)

                    logs = "[{}] {} делает Заказ №{}".format(time\_model, vistr.visitor, num\_order)

                    lbxLogs.insert(0, logs)

                    num\_order += 1

                    vistr.eat\_up()

        if chef.state == "wait":

            if chef.queue:

                chef.get\_order()

                chef.cook\_order(time\_model)

                logs = "[{}] Повар готовит Заказ №{} : {}".format(time\_model, chef.cook.number\_order, chef.cook.order)

                lbxLogs.insert(0, logs)

        if chef.state == "cook":

            order\_ready = chef.issue\_order(time\_model)

            if order\_ready != None:

                chef.set\_order()

                logs = "[{}] Повар приготовил Заказ №{} : {}".format(time\_model, order\_ready.number\_order, order\_ready.order)

                lbxLogs.insert(0, logs)

        time.sleep(0.2)

        lbTimer["text"] = "Время: {}".format(time\_model)

        time\_model += 1

    print("end")

def modeling():

    thread = threading.Thread(target=thread\_modeling)

    thread.start()

form = Tk()

form.title("Ресторан")

form.config(bg="#65C8F5")

image = Image.open("img/table.png")

table = ImageTk.PhotoImage(image)

image = Image.open("img/oven.png")

kitchen = ImageTk.PhotoImage(image)

lbTimer = Label(form, text="Врeмя:", font=("Helvetica", 18, font.BOLD), bg="#65C8F5")

cvMan1 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvMan2 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvMan3 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvKitchen = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvState1 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvState2 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvState3 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

cvState4 = Canvas(form, height=SIZE, width=SIZE, bg="#65C8F5")

lbTable1 = Label(form, image=table, height=SIZE, bg="#65C8F5")

lbTable2 = Label(form, image=table, height=SIZE, bg="#65C8F5")

lbTable3 = Label(form, image=table, height=SIZE, bg="#65C8F5")

lbKitchen = Label(form, image=kitchen, height=SIZE, bg="#65C8F5")

btnStart = Button(form, text="Start", command=modeling, font=("Helvetica", 18, font.BOLD), width=10)

lbxLogs = Listbox(form, height=18, width=45, font=("Helvetica", 18, font.BOLD))

lbTimer.grid(columnspan=4, column=0, row=0)

lbxLogs.grid(rowspan=5, column=4, row=0)

cvMan1.grid(column=0, row=1, padx=10)

cvMan2.grid(column=1, row=1, padx=10)

cvMan3.grid(column=2, row=1, padx=10)

cvKitchen.grid(column=3, row=1, padx=30)

cvState1.grid(column=0, row=2)

cvState2.grid(column=1, row=2)

cvState3.grid(column=2, row=2)

cvState4.grid(column=3, row=2)

lbTable1.grid(column=0, row=3)

lbTable2.grid(column=1, row=3)

lbTable3.grid(column=2, row=3)

lbKitchen.grid(column=3, row=3)

btnStart.grid(columnspan=4, column=0, row=4)

form.mainloop()

Файл visitor.py:

from tkinter import Canvas

from PIL import Image, ImageTk

import random

import json

from order import Order

with open("img/people/people.json", "r") as fl:

    people = json.load(fl)

class Visitor:

    def \_\_init\_\_(self, cvPeople:Canvas, cvStatus:Canvas):

        self.visitor = random.choice(list(people))

        self.img = ImageTk.PhotoImage(Image.open("img/people/" + people[self.visitor]["img"]))

        self.paper = ImageTk.PhotoImage(Image.open("img/menu.png"))

        self.timer = ImageTk.PhotoImage(Image.open("img/timer.png"))

        self.size = self.img.width()

        self.cvPeople = cvPeople

        self.cvPeople.delete("all")

        self.\_print\_image(self.cvPeople, self.img)

        self.cvStatus = cvStatus

        self.time\_order = None

        self.time\_eat = None

        self.order = None

    def init\_order(self, time, num\_order):

        self.cvStatus.delete("all")

        self.\_print\_image(self.cvStatus, self.paper)

        self.time\_order = time + random.randint(people[self.visitor]["min\_time\_order"], people[self.visitor]["max\_time\_order"])

        self.order = Order(num\_order)

    def issue\_order(self, time):

        if self.time\_order == time:

            self.cvStatus.delete("all")

            self.\_print\_image(self.cvStatus, self.timer)

            return self.order

        return None

    def check\_order(self, order):

        return self.order == order

    def init\_eat(self, time):

        img = ImageTk.PhotoImage(Image.open(self.order.img))

        self.cvStatus.delete("all")

        self.\_print\_image(self.cvStatus, img)

        self.time\_eat = time + random.randint(people[self.visitor]["min\_time\_eat"], people[self.visitor]["max\_time\_eat"])

    def issue\_eat(self, time:int):

        if self.time\_eat != None:

            return self.time\_eat <= time

    def \_print\_image(self, cavans:Canvas, image:ImageTk.PhotoImage):

        cavans.create\_image(self.size/2,self.size/2,image=image)

        cavans.image = image

    def visitor\_info(self):

        return people[self.visitor]

Файл cook.py:

from tkinter import Canvas

from PIL import Image, ImageTk

import order

class Cook:

    def \_\_init\_\_(self, cvCook:Canvas, cvStatus:Canvas):

        self.queue = []

        self.cook = None

        self.img\_chef = ImageTk.PhotoImage(Image.open("img/chef.png"))

        self.img\_break = ImageTk.PhotoImage(Image.open("img/break.png"))

        self.cvCook = cvCook

        self.cvStatus = cvStatus

        self.size = self.img\_chef.width()

        self.cvCook.delete("all")

        self.\_print\_image(self.cvCook, self.img\_chef)

        self.cvStatus.delete("all")

        self.\_print\_image(self.cvStatus, self.img\_break)

    def add\_order(self, order:order.Order):

        self.queue.insert(0, order)

    def cook\_order(self, time:int):

        self.cook = self.queue.pop()

        img = ImageTk.PhotoImage(Image.open(self.cook.img))

        self.cvStatus.delete("all")

        self.\_print\_image( self.cvStatus, img)

        self.cook.init\_cooking(time)

    def issue\_order(self, time:int):

        if self.cook.check\_cooking(time):

            self.cvStatus.delete("all")

            self.\_print\_image(self.cvStatus, self.img\_break)

            cook = self.cook

            self.cook = None

            return cook

        return None

    def \_print\_image(self, cavans:Canvas, image:ImageTk.PhotoImage):

        cavans.create\_image(self.size/2,self.size/2, image=image)

        cavans.image = image

Файл order.py:

import random

import json

with open("img/food/food.json", "r") as fl:

    food = json.load(fl)

class Order:

    def \_\_init\_\_(self, number\_order):

        self.number\_order = number\_order

        self.order = random.choice(list(food))

        self.img = "img/food/" + food[self.order]["img"]

        self.time\_cooking = None

    def init\_cooking(self, time):

        self.time\_cooking = time + random.randint(food[self.order]["min\_time"] , food[self.order]["max\_time"])

    def check\_cooking(self, time):

        return self.time\_cooking <= time

    def info\_order(self):

        return food[self.order]

Результаты работы программы, реализующей дискретно событийную модель

При запуске программы появляется ее окно представлен на рисунке 1. Интерфейс программы состоит из следующих элементов:

1. Элемент отображения модельного времени;
2. Элементы отображения дескрипторов посетителей и их состояния;
3. Элементы отображения дескриптора повара и его состояния
4. Кнопка «Start» для запуска дискретно-событийного монитора.
5. Список событий, куда записываются изменения состояния дескрипторов в процессе модельного времени.

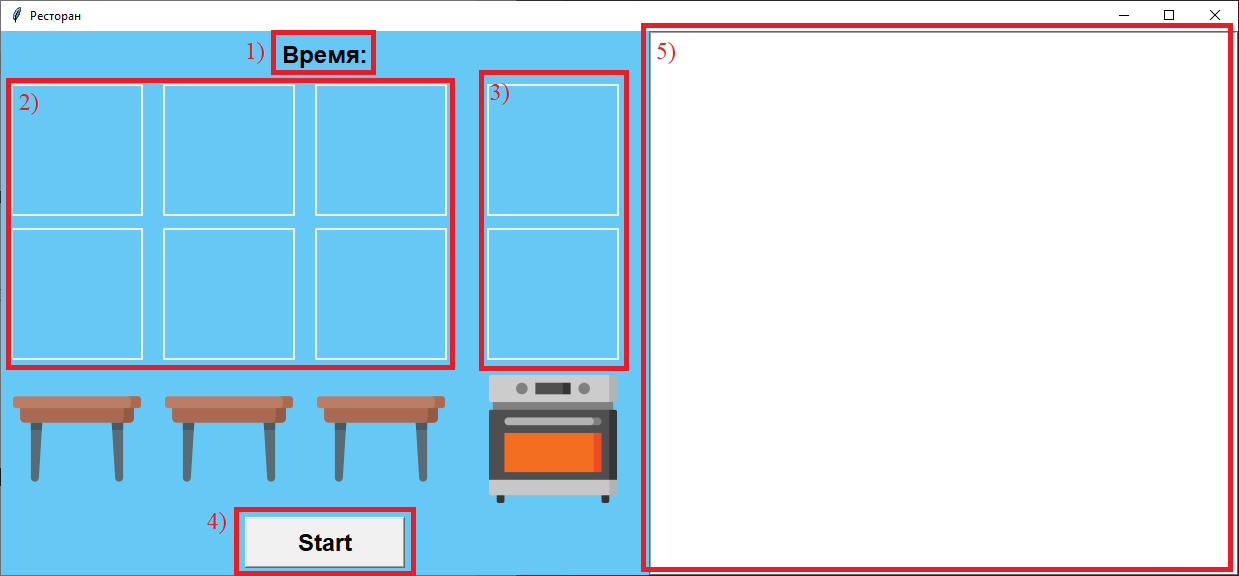


Рисунок 1 – Окно программы

При нажатии кнопки «Start» генерируется новый дискретно-событийный монитор с своими дескрипторами и сигналами. Дескрипторы посетителей переходят в состояние «выполнение заказа», а дескриптор повар переходит в состояние «ожидание заказа от посетителя». Окно программы в момент запуска дискретно-событийного монитора представлен на рисунке 2.

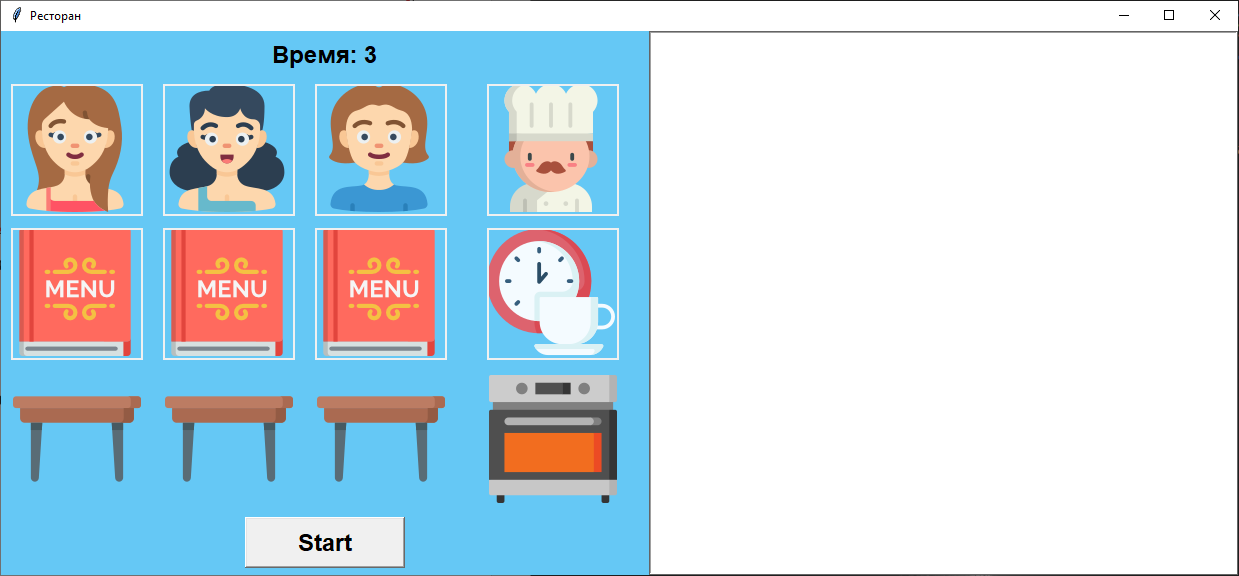


Рисунок 2 – Окно программы в момент запуска дискретно-событийного монитора

В моменты модельного времени 23 и 27, дескрипторы посетителей сделали свои заказы и перешли в состояние «ожидание заказа», окно программы представлено на рисунке 3. В очередь заказов дескриптора повара попали сделанные заказы посетителей, и повар начал выполнение заказа, первого попавшего в очередь (в списке событий видно, как посетитель Gabriel сделала первой заказ на udon и повар начал его выполнение).

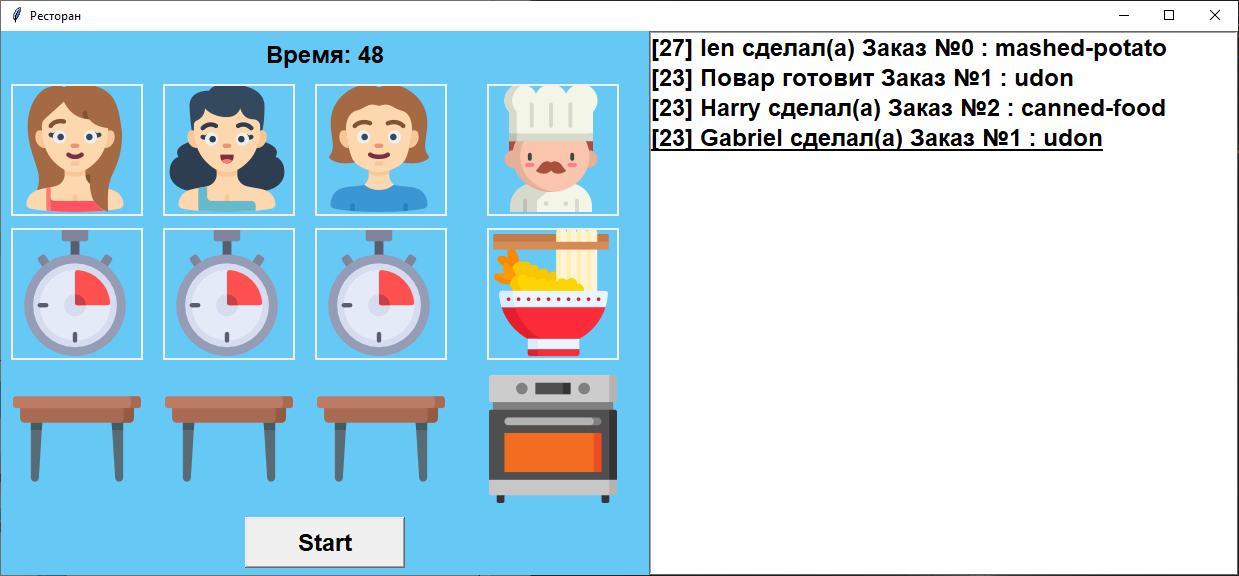


Рисунок 2 – Окно программы в момент времени 48

В моменты модельного времени 53 дескриптор повара приготовил заказ №1 и перешел к выполнению следующего заказа, в момент модельного времени 54 посетитель Gabriel получила свой заказ и перешла в состояние «Обработка заказа», окно программы представлено на рисунке 4.

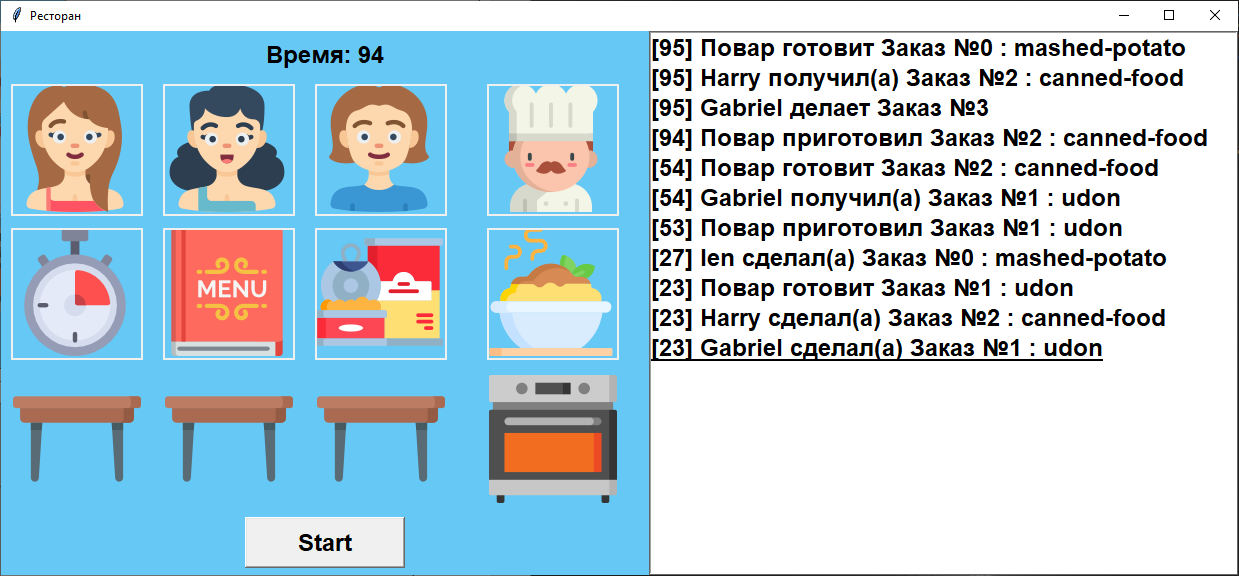


Рисунок 2 – Окно программы в момент времени 94

В моменты модельного времени 159 дескриптор повара приготовил последний заказ из своей очереди и передал его посетителю Gabriel, в это время остальные посетители еще находятся в состоянии «обработки заказа» и еще не успели перейти к состоянию «выполнение заказ», таким образом, дискретно-событийная модель завершает свою работу. Окно программы представлено на рисунке 5.

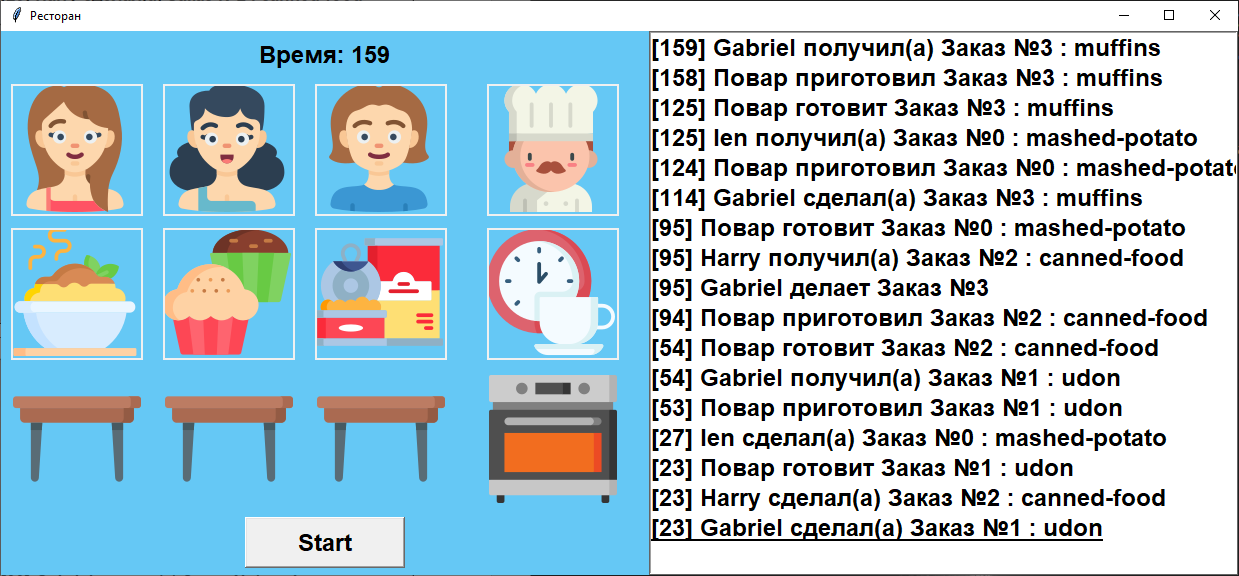


Рисунок 5 – Окно программы в момент времени 159