Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ №1**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Цель работы**

Написать программу на любом языке программирования, симулирующую поведение среды с агентами

**Описание задачи**

Имеется n агентов и окружающая их среда. В окружающей среде в случайные моменты времени появляются клиенты, требующие обслуживания.

Каждый клиент делает публичную заявку на обслуживание с указанием его сложности. Агенты могут предоставить такое обслуживание клиентам.

Работают они индивидуально, но могут координировать между собой действия.

Каждый агент создаёт очередь - места для клиентов, требующих обслуживания. На обслуживание клиенты выбираются из очереди в соответствии

с дисциплиной FIFO (first in - first out). Агенты наблюдают за окружающей средой: как только появляется новый клиент, они принимают

совместное решение о том, кто его возьмёт в свою очередь. Работа всей системы заканчивается по окончании обслуживания m-го клиента

(клиент с номером m+1 уже не принимается ни в какую из очередей).

По окончании работы агенты составляют совместный отчёт, в котором каждый указывает свой id, сколько клиентов он обслужил и какое время

на это затратил. Записи в отчёте упорядочены по убыванию количества клиентов; при одинаковом количестве клиентов - по возрастанию времени.

Правила для окружающей среды:

Время начинается с t=0.

Первый клиент объявляется через случайный интервал времени с равномерным распределением на [a, b]. a и b - вещественные числа.

Интервал между i-ым и (i+1)-ым клиентом также случайный с равномерным распределением на [a, b]. Все интервалы взаимно независимы.

Для каждого клиента задаётся сложность его обслуживания - целочисленная случайная величина в пределах от 1 до 10 единиц времени.

Правила для совещания агентов:

Задачу берёт тот агент, у которого наименьшая текущая загрузка (суммарная сложность обслуживания клиентов, находящихся в очереди, плюс

время дообслуживания клиента, находящегося на обслуживании у агента). При равной загрузке агентов задачу берёт первый в списке id.

Дополнительные сведения (узнал после вопросов):

На обслуживание клиентов у агента тратится время равное сложности обслуживания текущего клиента.

**Описание решения**

Для реализации данной задачи был выбран Python, а также объектно-ориентированный подход.

Было создано 3 файла: task.txt (в котором содержалось описание задачи), classes.py (в котором содержались все необходимые классы) и main.py (в котором содержался основная логика).

Classes.py

Было решено написать 2 класса: Клиент и Агент. Класс Клиента представляет собой конструктор с 3-мя полями: id, complexity (сложность), arrival\_time (время появления). Класс Агента получился более объёмный; он содержит: конструктор с такими полями как id, queue (очередь клиентов), current\_client (текущий клиент), remaining\_time (оставшееся время обслуживания клиента), total\_clients\_served (кол-во обслуженных клиентов), total\_time\_spent (общее потраченное время); метод agent\_info(), который печатает основную информацию об агенте (его id, кол-во обслуженных клиентов и общее потраченное время); метод agent\_load(), который возвращает текущую загрузку клиента; метод add\_to\_queue(), который добавляет клиента в очередь, и если текущий клиент отсутствует – начинает обработку нового клиента; метод start\_next\_client(), который берёт в обработку следующего клиента и обновляет оставшееся время обработки; метод update\_state(), который вызывается как итерационный шаг в цикле, т.е. на единицу уменьшает оставшееся время обслуживания клиента, и если время осталось 0 – пересчитывает общее кол-во обслуженных клиентов, общее затраченное время и вызывает метод start\_next\_client().

Main.py

Данный файл импортирует все вышеописанные классы и содержит основную логику работы симуляции, представленную через функции:

Create\_agents(): функция, которая создаёт нужное количество агентов;

Consilium(): функция, которая как бы «совещает» агентов между собой (агенты сообщают свою текущую загрузку, и после этого выбирают кто возьмёт себе нового клиента);

Main(): главная функция, которая отвечает за проведение симуляции. В ней создаются все агенты, считается время появления нового клиента, запускается цикл по количеству созданных клиентов (не превышая m), в котором проверяется, достигло ли глобальное время времени спавна клиента, спавнится новый клиент, вызывается «консилиум» агентов, обновляется счётчик созданных клиентов, вычисляется время спавна нового клиента, выполняется обновление состояния каждого агента и увеличивается глобальное время. Также присутствует вспомогательный цикл для того, чтобы агенты могли дообслужить своих клиентов, когда новые клиенты уже не появляются (m-ый клиент появился, но m-1 и т.д. ещё не обслужились).

**Код программы**

classes.py

import random as rnd

from collections import deque

class Client:

    def \_\_init\_\_(*self*, *client\_id*: int, *client\_arrival\_time*: float) -> None:

        """

        Конструктор, который создаёт объект класса Клиент с нужными полями

        Args:

            client\_id (int): Id, который присваивается каждому клиенту

            client\_arrival\_time (float): Время появления данного клиента

        """

*self*.id = *client\_id*                       *# id клиента*

*self*.complexity = rnd.randint(1, 10)      *# сложность обслуживания клиента*

*self*.arrival\_time = *client\_arrival\_time*   *# время появления клиента*

class Agent:

    def \_\_init\_\_(*self*, *agent\_id*: int) -> None:

        """

        Конструктор, который создаёт объект класса Агент с нужными полями

        Args:

            agent\_id (int): Id, который присваивается каждому агенту

        """

*self*.id = *agent\_id*              *# id агента*

*self*.queue = deque()            *# очередь клиентов данного агента*

*self*.current\_client = None      *# текущий клиент, которого обслуживает агент*

*self*.remaining\_time = 0         *# время, оставшееся до конца обслуживания текущего клиента*

*self*.total\_clients\_served = 0   *# кол-во обслуженных клиентов*

*self*.total\_time\_spent = 0       *# кол-во затраченного времени*

    def agent\_info(*self*) -> None:

        """

        Метод, который выводит краткую информацию об агенте

        """

        print(f"Агент: id = {*self*.id}, обслужено клиентов = {*self*.total\_clients\_served}, потрачено времени = {*self*.total\_time\_spent}")

    def agent\_load(*self*) -> int:

        """

        Метод, который возвращает текущую загрузку агента

        """

        queue\_load = sum(client.complexity for client in *self*.queue)   *# загрузка очереди*

        return *self*.remaining\_time + queue\_load                        *# загрузка очереди + оставшееся время обслуживания*

    def add\_to\_queue(*self*, *client*: Client):

        """

        Метод, который добавляет нового клиента в очередь агенту

        Args:

            client (Client): Клиент, который добавляется в очередь

        """

*self*.queue.append(*client*)         *# добавляем клиента в очередь*

        if *self*.current\_client is None:   *# если сейчас никакой клиент не обслуживается*

*self*.start\_next\_client()      *# то стартуем обработку нового клиента*

    def start\_next\_client(*self*):

        """

        Метод, который берёт в обработку следующего клиента из очереди

        """

        if (len(*self*.queue) > 0):                                  *# проверка очереди на наличие клиентов*

*self*.current\_client = *self*.queue.popleft()             *# делаем активным нового клиента*

*self*.remaining\_time = *self*.current\_client.complexity   *# устанавливаем текущую загрузку клиента*

    def update\_state(*self*) -> None:

        """

        Метод, который обновляет состояние агента

        """

        if *self*.current\_client is not None:                               *# если сейчас есть обслуживаемый клиент*

*self*.remaining\_time -= 1                                      *# уменьшаем оставшееся время обслуживания*

            if *self*.remaining\_time <= 0:                                  *# если оставшееся время <= 0*

*self*.total\_clients\_served += 1                            *# прибавляем одного клиента к обслуженным*

*self*.total\_time\_spent += *self*.current\_client.complexity   *# прибавляем его сложность к затраченному времени*

*self*.remaining\_time = 0                                   *# устанавливаем новое оставшееся время обслуживания*

*self*.current\_client = None                                *# устанавливаем отсутствие текущего клиента*

*self*.start\_next\_client()                                  *# стартуем обслуживание нового клиента*

main.py

from classes import \*

import random as rnd

import time

def create\_agents(*number\_of\_agents*: int) -> None:

    """

    Функция, которая создаёт всех агентов

    Args:

        number\_of\_agents (int): Общее кол-во агентов

    """

    for i in range(*number\_of\_agents*):

        agent = Agent(i+1)

        list\_of\_agents.append(agent)

def consilium(*client*: Client) -> int:

    """

    Функция, которая выполняет роль "консилиума" агентов (они выбирают кому отдать нового клиента)

    Args:

        client (Client): Клиент, которого "делят" агенты

    """

    if len(list\_of\_agents) > 0:                                                                *# если список агентов не пуст*

        list\_of\_messages = []                                                                  *# создаём список сообщений агентов*

        for i in range(len(list\_of\_agents)):                                                   *# проходимся по всем агентам*

            list\_of\_messages.append(list\_of\_agents[i].agent\_load())                            *# каждый агент записывает свою текущую загрузку*

        min\_load = min(list\_of\_messages)                                                       *# находим минимальную текущую загрузку*

        for i in range(len(list\_of\_messages)):                                                 *# проходимся по всем сообщениям*

            if list\_of\_messages[i] == min\_load:                                                *# если нашли минимальную текущую загрузку*

                list\_of\_agents[i].add\_to\_queue(*client*)                                         *# то по этому id (у агентов аналогичные) выдаём нового клиента*

                print(f"Агент id = {list\_of\_agents[i].id} получил клиента id = {*client*.id}")   *# просто уведомление*

                break                                                                          *# выходим из цикла*

def main(*n*: int, *m*: int, *ab*: list, *t\_global*: int, *list\_of\_agents*: list, *list\_of\_clients*: list) -> None:

    """

    Основная функция, которая отвечает за симуляцию

    Args:

        n (int): Общее количество агентов

        m (int): Общее количество клиентов

        ab (list): Интервал появления клиентов

        t\_global (int): Общее время симуляции

        list\_of\_agents (list): Общий список агентов

        list\_of\_clients (list): Общий список клиентов

    """

    client\_count = 0                                                                                                         *# кол-во созданных клиентов*

    create\_agents(*n*)                                                                                                         *# создание всех агентов*

    next\_client\_time\_spawn = rnd.uniform(*ab*[0], *ab*[1])                                                                       *# задаём время появления следующего клиента*

    while client\_count < *m*:                                                                                                  *# запускаем цикл пока не создастся m клиентов*

        print(f"Текущий момент времени: {*t\_global*}")                                                                         *# отмечаем текущий момент времени*

        if *t\_global* >= next\_client\_time\_spawn and client\_count < *m*:                                                          *# проверяем время на предмет появления и общее количество*

            client = Client(client\_count + 1, *t\_global*)                                                                      *# создаём клиента*

*list\_of\_clients*.append(client)                                                                                   *# добавляем клиента в общий список*

            print(f"Клиент {client.id} появился в момент времени {*t\_global*}, сложность обслуживания: {client.complexity}")   *# информация*

            consilium(client)                                                                                                *# совещания агентов при появлении нового клиента*

            client\_count += 1                                                                                                *# увеличиваем счётчик клиентов*

            next\_client\_time\_spawn = *t\_global* + rnd.uniform(*ab*[0], *ab*[1])                                                    *# создаём время для появления следующего клиента*

        for agent in *list\_of\_agents*:                                                                                         *# перебираем всех агентов*

            agent.update\_state()                                                                                             *# обновляем состояния всех агентов*

*t\_global* += 1                                                                                                        *# увеличиваем глобальное время*

        time.sleep(0.05)                                                                                                     *# приятный вывод*

    while any(agent.agent\_load() > 0 for agent in *list\_of\_agents*):                                                           *# вспомогательный цикл для дообработки клиентов*

        print(f"Текущий момент времени: {*t\_global*} (ожидаем обработки оставшихся клиентов)")                                 *# информация*

        for agent in *list\_of\_agents*:                                                                                         *# снова перебираем всех агентов*

            agent.update\_state()                                                                                             *# обновляем состояния всех агентов*

*t\_global* += 1                                                                                                        *# увеличиваем общее время*

        time.sleep(0.05)                                                                                                     *# приятный вывод*

n = 5                                                *# кол-во агентов*

m = 20                                               *# кол-во клиентов*

ab = sorted([rnd.uniform(1, 6) for i in range(2)])   *# интервал появления клиентов*

t = 0                                                *# глобальное время*

list\_of\_agents = []                                  *# хранилище агентов*

list\_of\_clients = []                                 *# хранилище клиентов*

if \_\_name\_\_ == "\_\_main\_\_":

    main(n, m, ab, t, list\_of\_agents, list\_of\_clients)                                                                *# запуск симуляции*

    sorted\_agents = sorted(list\_of\_agents, *key*=lambda *agent*: (-*agent*.total\_clients\_served, *agent*.total\_time\_spent))   *# сортировка агентов по убыванию (обслуженные) и по возрастанию (время)*

    print()

    print("Отчёт об агентах:")

    for agent in sorted\_agents:

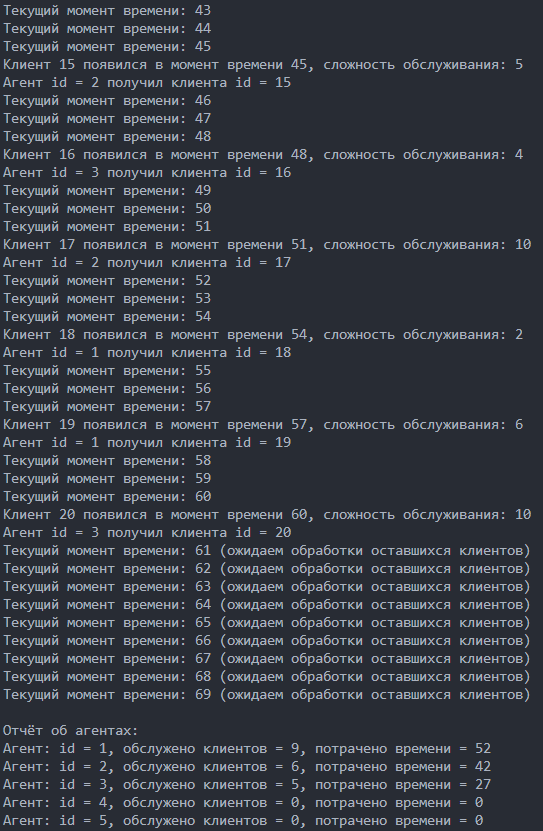
        agent.agent\_info()

**Примеры вывода**

n = 5                                                *# кол-во агентов*

m = 20                                               *# кол-во клиентов*

ab = sorted([rnd.uniform(1, 6) for i in range(2)])   *# интервал появления клиентов*



n = 3                                                *# кол-во агентов*

m = 25                                               *# кол-во клиентов*

ab = sorted([rnd.uniform(1, 4) for i in range(2)])   *# интервал появления клиентов*

