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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. В. Сидоренко

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

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

Краснодар

2024

**Цель работы**

Написать программу по заданному ниже ТЗ, используя любой язык программирования.

**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**

Программа запускает имитацию системы, которая включает в себя агентов, взаимодействующих между собой и с клиентами.

1. Симуляция запускается с заданным количеством агентов и максимальным количеством клиентов.

2. Время симуляции начинается с 0.

3. В цикле, пока не будет обслужено maxClients клиентов:

- Генерируется случайное время появления следующего клиента в интервале intervalStart - intervalEnd.

- Текущее время симуляции увеличивается на это время.

- Создается новый клиент.

- Выбирается агент с наименьшей текущей нагрузкой (количество клиентов в очереди).

- Новый клиент добавляется в очередь выбранного агента.

- Счетчик обслуженных клиентов увеличивается на 1.

4. После завершения цикла добавления клиентов, запускается цикл обслуживания:

- В этом цикле, пока есть необслуженные клиенты:

- Каждый агент обслуживает следующего клиента из своей очереди.

5. После завершения обслуживания всех клиентов, агенты сортируются по количеству обслуженных клиентов и времени обслуживания.

6. Выводится отчет о работе каждого агента.

**Код программы**

Client

import java.util.Random;

public class Client {

    private static int counter = 0; *// Статическая переменная для уникальных идентификаторов клиентов*

    private int id; *// Уникальный идентификатор клиента*

    private int complexity; *// Сложность обслуживания клиента*

    public Client() {

        this.id = counter++; *// Присваивание уникального идентификатора, инкрементирование счетчика*

        Random random = new Random();

        this.complexity = 1 + random.nextInt(10); *// Генерация случайной сложности от 1 до 10*

    }

    public int getId() {

        return id;

    }

    public int getComplexity() {

        return complexity;

    }

}

Agent

import java.util.LinkedList;

import java.util.Queue;

public class Agent implements Comparable<Agent> {

    private int id; *// Уникальный идентификатор агента*

    private Queue<Client> queue; *// Очередь клиентов в дисциплине FIFO*

    private int totalServicedClients = 0; *// Общее количество обслуженных клиентов*

    private int totalTimeSpent = 0; *// Общее время, затраченное на обслуживание клиентов*

    private int currentLoad = 0; *// Текущая загрузка агента (суммарная сложность всех клиентов в очереди)*

    public Agent(int *id*) {

        this.id = id;

        this.queue = new LinkedList<>(); *// Инициализация очереди клиентов*

    }

    public int getId() {

        return id;

    }

    public void addClient(Client *client*) {

        queue.offer(client); *// Добавление клиента в очередь*

        currentLoad += client.getComplexity(); *// Увеличение текущей загрузки агента*

    }

    public void serviceClient() {

        if (!queue.isEmpty()) { *// Если в очереди есть клиенты*

            Client client = queue.poll(); *// Взятие клиента из начала очереди*

            totalServicedClients++; *// Увеличение счетчика обслуженных клиентов*

            totalTimeSpent += client.getComplexity(); *// Добавление времени обслуживания клиента*

            currentLoad -= client.getComplexity(); *// Уменьшение текущей загрузки агента*

        }

    }

    public boolean isQueueEmpty() {

        return queue.isEmpty(); *// Проверка, пуста ли очередь агента*

    }

    public int getTotalServicedClients() {

        return totalServicedClients;

    }

    public int getTotalTimeSpent() {

        return totalTimeSpent;

    }

    public int getCurrentLoad() {

        return currentLoad;

    }

    @Override

    public int compareTo(Agent *other*) {

*// Сравнение агентов: сначала по количеству обслуженных клиентов, затем по времени обслуживания*

        if (this.totalServicedClients != other.totalServicedClients) {

            return other.totalServicedClients - this.totalServicedClients;

        }

        return this.totalTimeSpent - other.totalTimeSpent;

    }

}

Environment

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Random;

public class Environment {

    private List<Agent> agents; *// Список агентов*

    private int maxClients; *// Максимальное количество клиентов, которые могут быть обслужены*

    private double intervalStart; *// Начало интервала времени для появления клиентов*

    private double intervalEnd; *// Конец интервала времени для появления клиентов*

    public Environment(int *numberOfAgents*, int *maxClients*, double *intervalStart*, double *intervalEnd*) {

        this.maxClients = maxClients;

        this.intervalStart = intervalStart;

        this.intervalEnd = intervalEnd;

        agents = new ArrayList<>();

        for (int i = 0; i < numberOfAgents; i++) {

            agents.add(new Agent(i)); *// Инициализация агентов с уникальными идентификаторами*

        }

    }

    public void startSimulation() {

        int servicedClients = 0; *// Количество обслуженных клиентов*

        Random random = new Random();

        double currentTime = 0; *// Текущее время*

        while (servicedClients < maxClients) { *// Пока не достигнуто максимальное количество клиентов*

            double nextClientTime = intervalStart + (intervalEnd - intervalStart) \* random.nextDouble();

            currentTime += nextClientTime; *// Обновление текущего времени*

            Client client = new Client(); *// Создание нового клиента*

*// Выбор агента с наименьшей загрузкой*

            Agent selectedAgent = agents.get(0);

            for (Agent agent : agents) {

                if (agent.getCurrentLoad() < selectedAgent.getCurrentLoad()) {

                    selectedAgent = agent;

                } else if (agent.getCurrentLoad() == selectedAgent.getCurrentLoad() && agent.getId() < selectedAgent.getId()) {

                    selectedAgent = agent;

                }

            }

            selectedAgent.addClient(client); *// Добавление клиента в очередь выбранного агента*

            servicedClients++; *// Увеличение счетчика обслуженных клиентов*

        }

*// Обслуживание всех клиентов в очередях агентов после окончания добавления клиентов*

        boolean workRemains = true;

        while (workRemains) {

            workRemains = false;

            for (Agent agent : agents) {

                if (!agent.isQueueEmpty()) {

                    agent.serviceClient(); *// Обслуживание клиента агентом*

                    workRemains = true; *// Продолжаем цикл, пока есть необслуженные клиенты*

                }

            }

        }

        Collections.sort(agents); *// Сортировка агентов для итогового отчета*

        printReport(); *// Вывод отчета*

    }

    private void printReport() {

        for (Agent agent : agents) {

            System.out.println("Agent ID: " + agent.getId() + ", Clients Serviced: " + agent.getTotalServicedClients() + ", Time Spent: " + agent.getTotalTimeSpent());

        }

    }

}

Main

public class Main {

    public static void main(String[] *args*) {

        int numberOfAgents = 5; *// Количество агентов*

        int maxClients = 20; *// Максимальное количество клиентов*

        double intervalStart = 2.0; *// Начало интервала времени для появления клиентов*

        double intervalEnd = 5.0; *// Конец интервала времени для появления клиентов*

        Environment environment = new Environment(numberOfAgents, maxClients, intervalStart, intervalEnd);

        environment.startSimulation(); *// Запуск симуляции*

    }

}

**Примеры вывода**

