Министерство Образования и Науки РФ

Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И. Ульянова (Ленина)

**Отчет по учебной практике**  
**­­­**

Выполнили: Чирухин А. А.,

Блеес Э. И.,

Шкулев А. А.

Факультет: КТИ

Группа №: 3304

Преподаватель: Фирсов М.А.

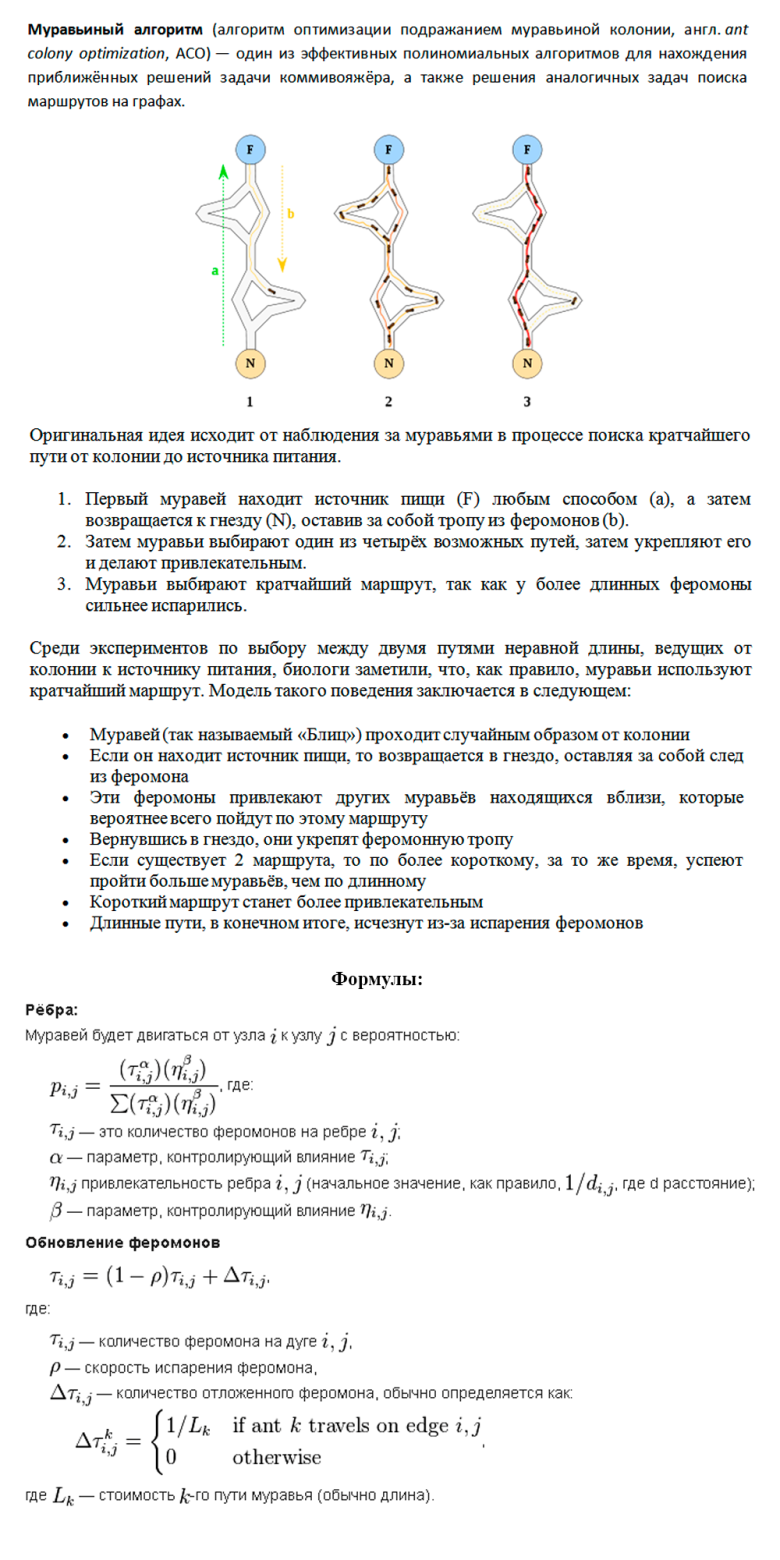
Санкт-Петербург

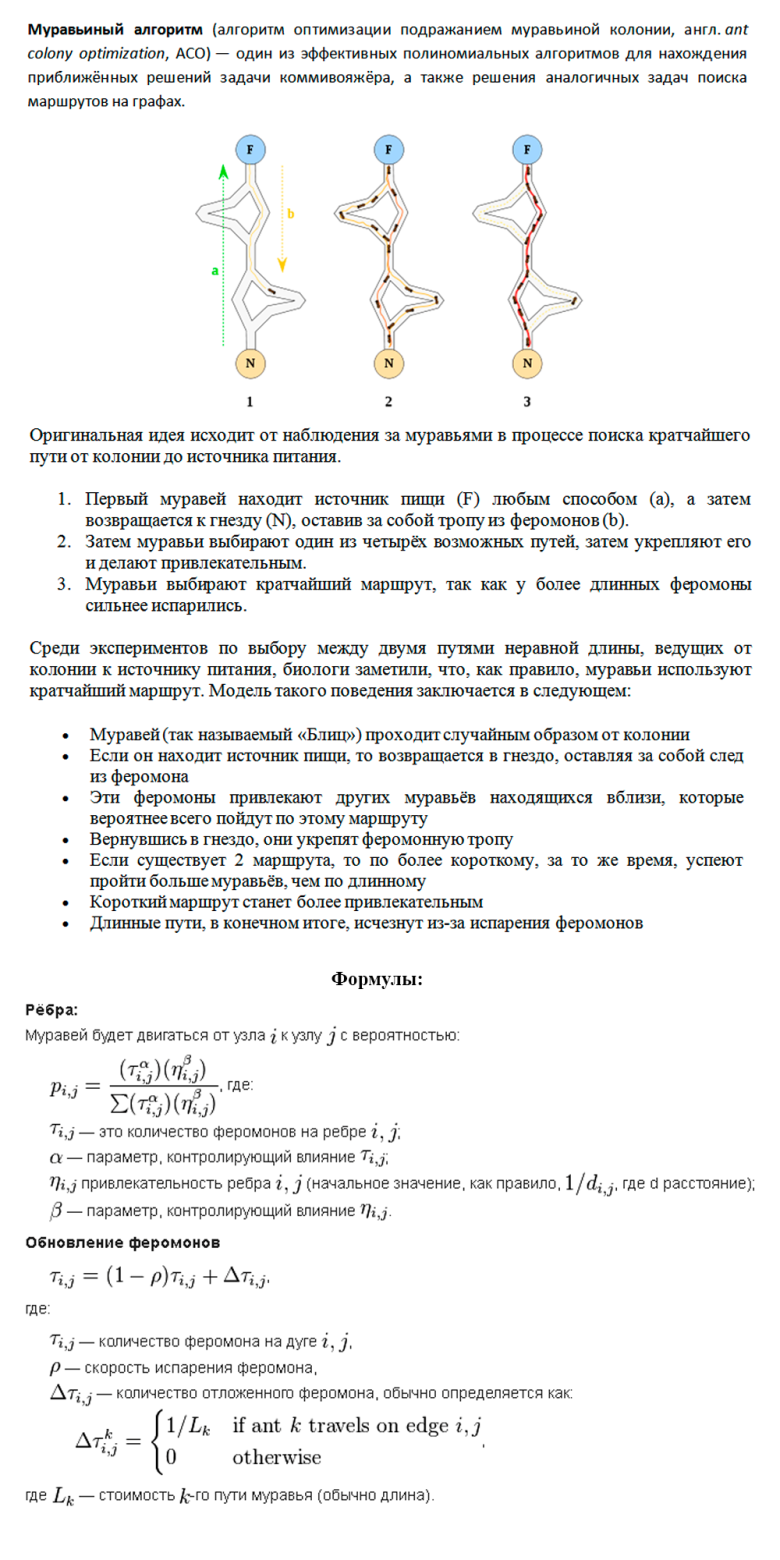
2015г.

**Формулировка задания**

В рамках учебной практики нами был реализован муравьиный алгоритм, для решения задачи нахождения оптимального пути в графе.

**Формальная постановка задачи**



  
  
**Спецификация программы**

**1. Общее описание**

1.1. Видение продукта

Данная программа реализует и визуализирует муравьиный алгоритм на примере задачи поиска оптимального пути в графе. Программа предоставляет пользователю удобный графический интерфейс.

1.2. Функциональность продукта

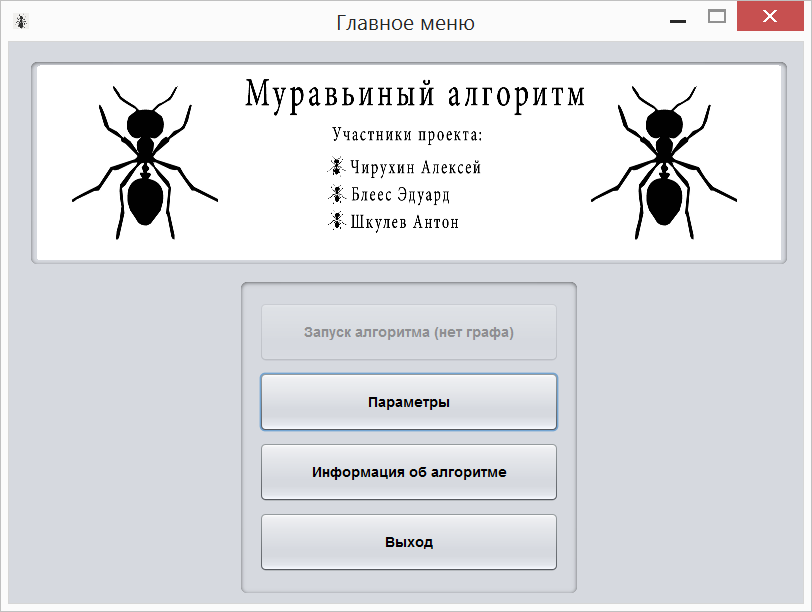
Программа предоставляет пользователю следующий набор функций:

* Ввод графа из файла (в виде матрицы смежности);
* генерация графа с разными параметрами (кол-во вершин, процент ребер, диапазон весов);
* ввод параметров алгоритма (величина, определяющая «жадность» алгоритма, скорость испарения феромона);
* пошаговая визуализация алгоритма.

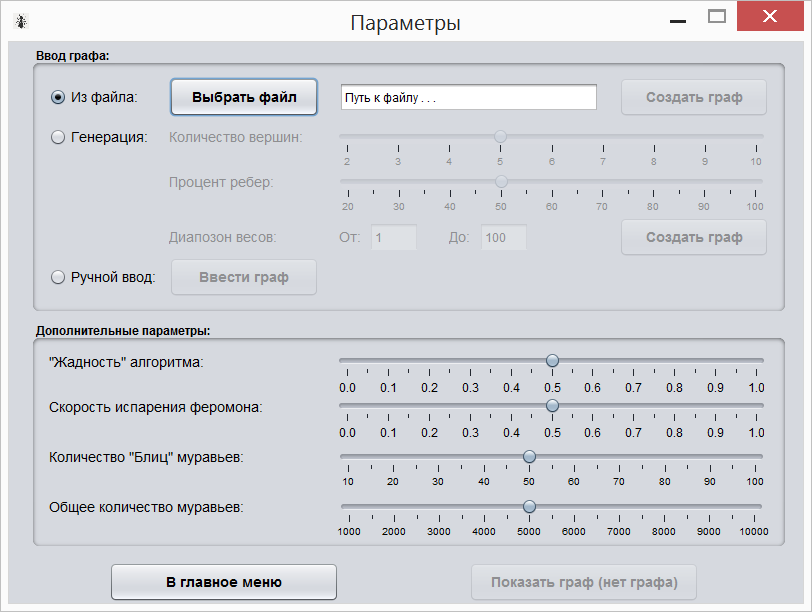
**2. Внешний интерфейс**

2.1. Интерфейсы пользователя (UX)

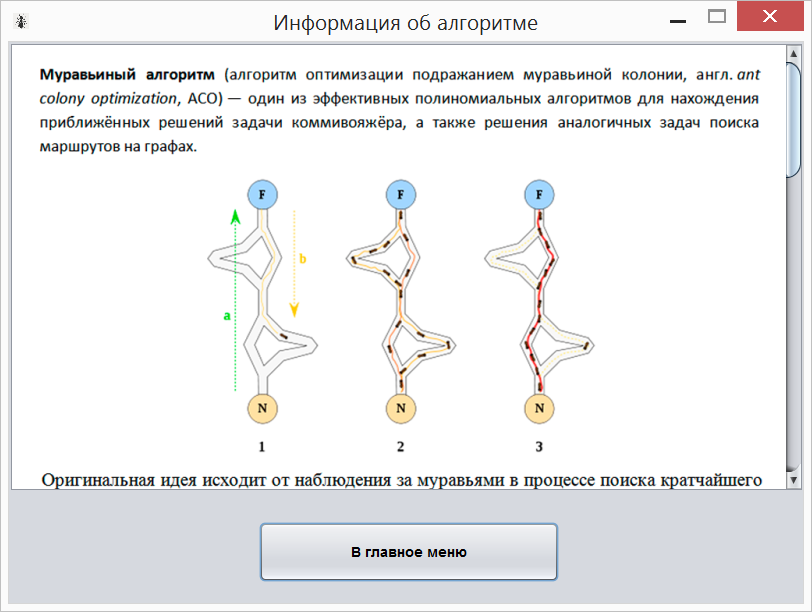
* Главное меню



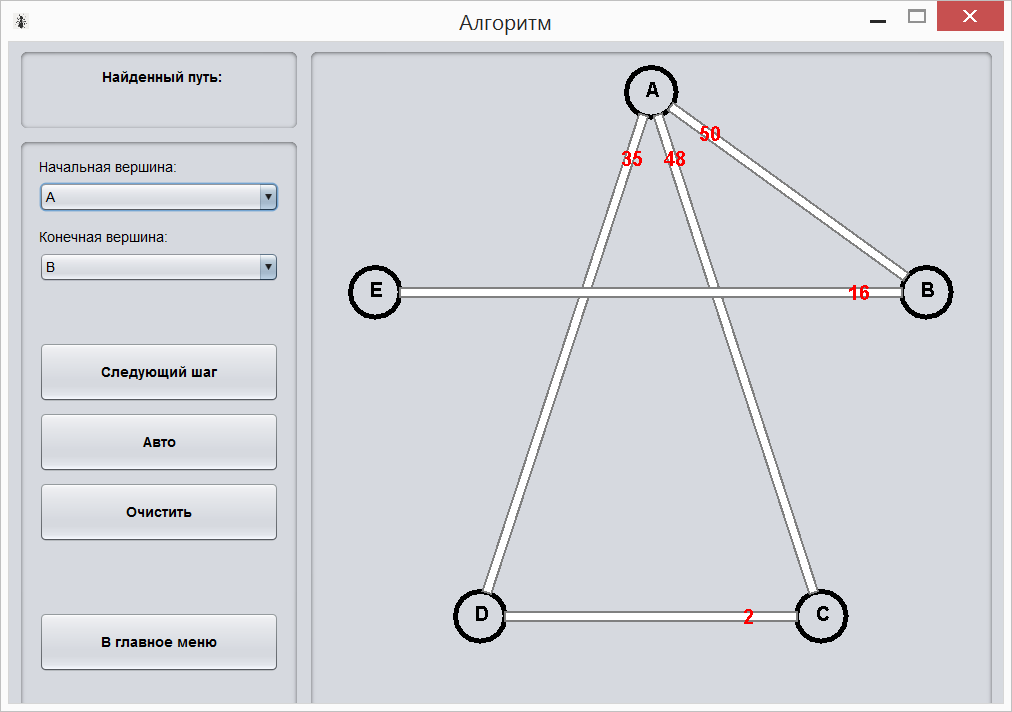
* Параметры



* Информация об алгоритме



* Запуск алгоритма



2.1.1. Объяснение интерфейса

Назначение кнопок:

* **Главное меню:**
* **Запуск алгоритма**

*Открытие формы алгоритма*

* **Параметры**

*Открытие формы параметров*

* **Информация об алгоритме**

*Открытие формы информации*

* **Выход**

*Выход из программы*

* **Параметры:**
* **Выбрать файл**

*Открытие файла с матрицей смежности*

* **Ручной ввод**

*Открытие окна рисования графа*

* **Посмотреть граф**

*Открытие окна с графом*

* **В главное меню**

*Выход в главное меню*

* **Создать граф**

*Создать граф*

* **Информация об алгоритме:**
* **В главное меню**

*Выход в главное меню*

* **Запуск алгоритма:**
* **Следующий шаг**

*Выполняется одна итерация алгоритма (проход одного муравья по пути, выбранному на основе алгоритма, обновление концентрации феромона на всех путях графа).*

***Уточнение:*** *сообщается тип муравья, его путь и вес пути.*

* **Авто**

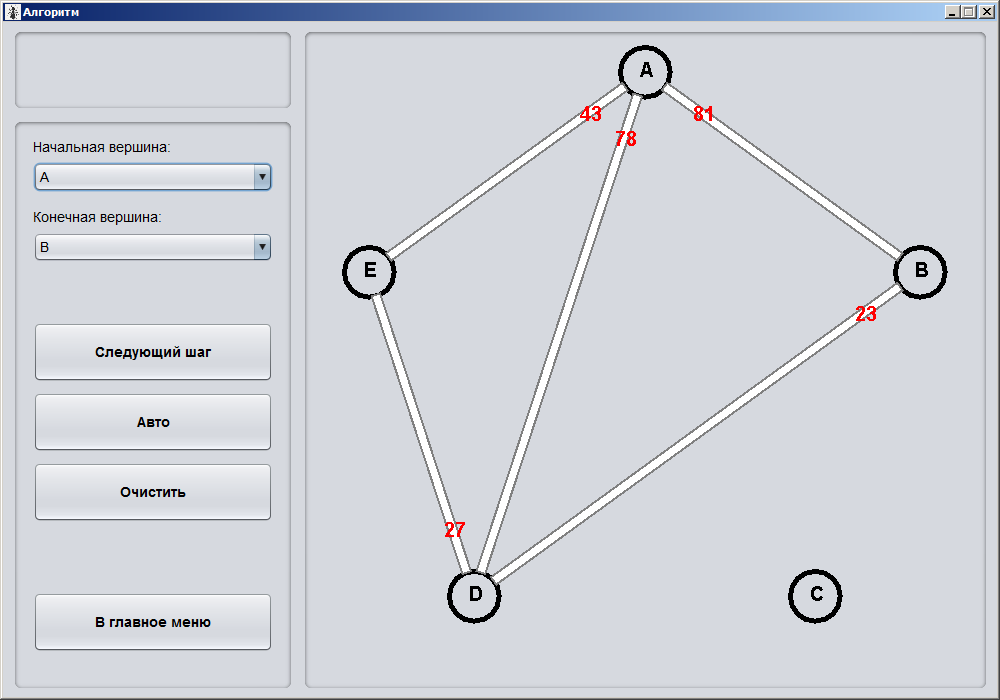
*Автоматический запуск следующего шага алгоритма после завершения предыдущего (быстрая демонстрация алгоритма).*

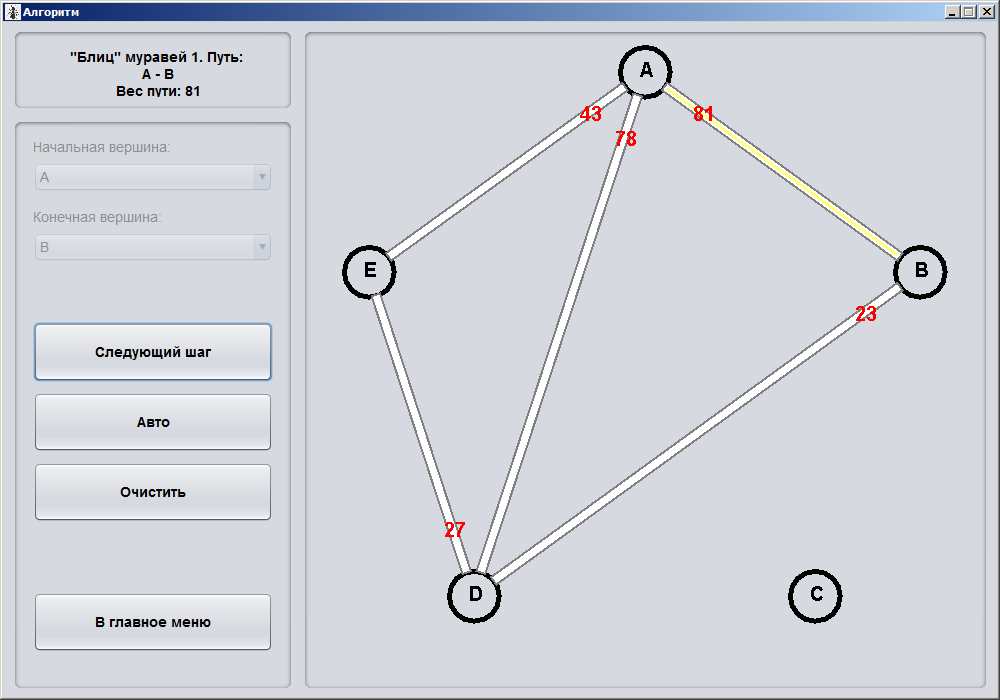
***Уточнение:*** *Выводится найденный путь и его вес.*

* **В главное меню**

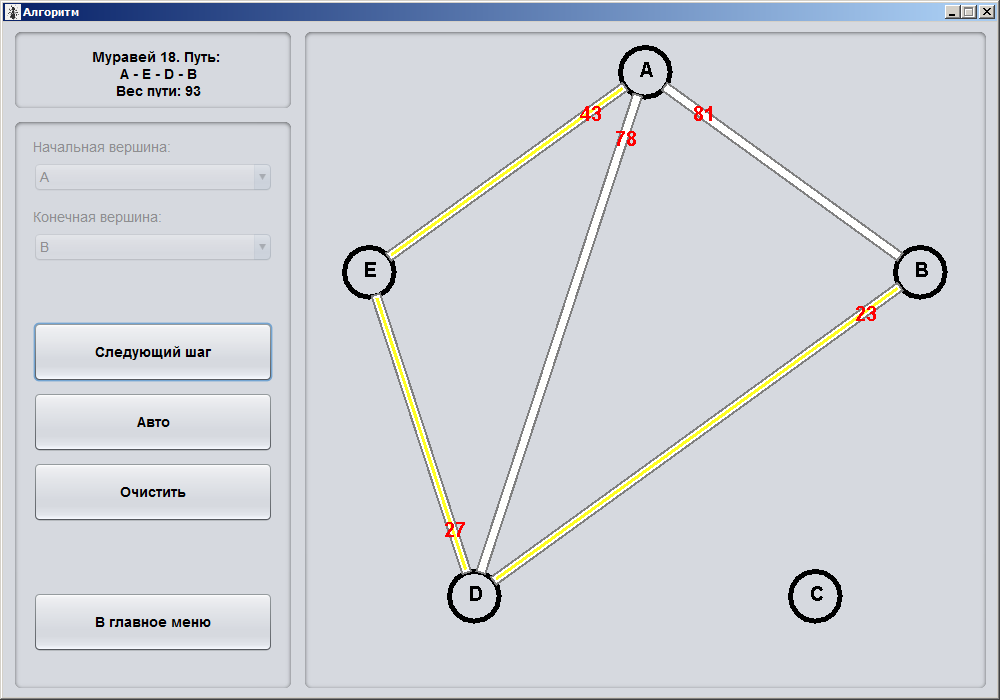
*Выход в главное меню*

2.1.2. Визуализация алгоритма

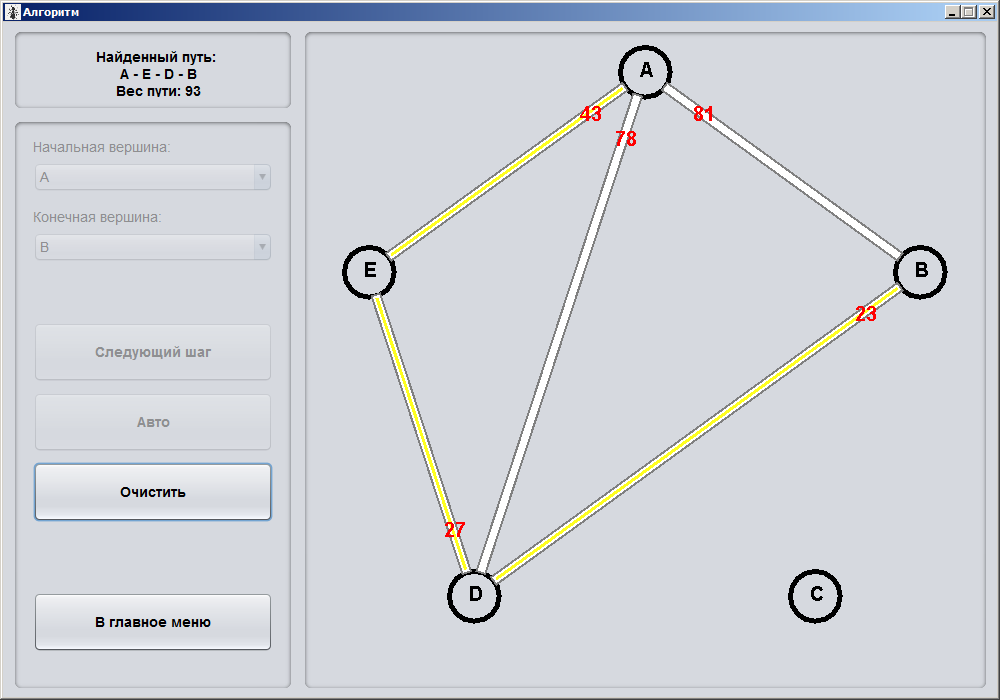
* Начало алгоритма  
  
* Первый шаг

  
Первый муравей случайным образом ищет путь до конечной вершины, при возвращении оставляет след феромонов. Концентрация феромонов обозначена цветом пути.

* В процессе работы алгоритма

  
Концентрация феромонов влияет на выбор пути у муравьев – чем она выше – тем выше вероятность выбора пути.

* Завершение алгоритма



2.2. Программные интерфейсы

При разработке приложения используется библиотека для создания графического интерфейса Swing.

**Распределение обязанностей**

**Чирухин Алексей:**

Лидер. Ответственный за репозиторий. Разработка интерфейса. Тестирование интерфейса.

**Блеес Эдуард:**

Ответственный за отчет. Разработка алгоритма. Тестирование алгоритма. Генерация графа.

**Шкулев Антон:**

Разработка визуализации графа. Тестирование визуализации.

**Использованные структуры данных**

**Edge:**

**public class** Edge {  
 **public double pheromone**;  
 **public int weight**;  
 **public int firstNode**; *// от вершины* **public int secondNode**; *// до вершины* **public boolean inCurrentPath**; *// принадлежность к пути* **public** Edge(**int** weight\_, **int** firstNode\_, **int** secondNode\_) {  
 **pheromone** = 0.0;  
 **weight** = weight\_;  
 **firstNode** = firstNode\_;  
 **secondNode** = secondNode\_;  
 **inCurrentPath** = **false**;  
 }  
}

Данная структура данных предназначается для хранения сведений (данных) о ребре графа.

**Vertex:**

**public class** Vertex {  
 **public** Character **letter**;  
 **public int coordX**;  
 **public int coordY**;  
  
 **public** Vertex(**int** x, **int** y, Character letter\_) {  
 **coordX** = x;  
 **coordY** = y;  
 **letter** = letter\_;  
 }  
}

Данная структура данных предназначается для хранения сведений (данных) о вершине графа.

**Graph:**

**public class** Graph {  
  
 **public static** Character[] *alphabet* = {**'A'**, *// алфавит для списков вершин* **'B'**, **'C'**, **'D'**, **'E'**, **'F'**, **'G'**, **'H'**, **'I'**, **'J'**};  
  
 **public int numberOfVertices**; *// Number of graph nodes* **public int**[][] **adjacencyMatrix**; *// Adjacency matrix with weights* **public boolean**[][] **linksMatrix**; *// Adjacency matrix with links only. We'll use Warshall algorithm to it.* **public** List<Edge> **edges**; *// Graph's edges* **public** List<Vertex> **vertices**; *// Graph's vertices* **public** Graph() { *// Constructor* **numberOfVertices** = 0;  
 **adjacencyMatrix** = **null**;  
 **linksMatrix** = **null**;  
 **edges** = **null**;  
 **vertices** = **null**;  
 }  
  
 **public** Graph(Graph g) { *// Copy constructor* **numberOfVertices** = g.**numberOfVertices**;  
  
 createMatrix(**numberOfVertices**);  
  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j) {  
 **adjacencyMatrix**[i][j] = g.**adjacencyMatrix**[i][j];  
 **linksMatrix**[i][j] = g.**linksMatrix**[i][j];  
 }  
  
 **if** (g.**edges** != **null**) **edges** = **new** ArrayList<>(g.**edges**);  
 **if** (g.**vertices** != **null**) **vertices** = **new** ArrayList<>(g.**vertices**);  
 }

}

Данная структура данных предназначается для хранения сведений (данных) о графе.

**План разработки**

**Вторник 23.06.15**

Создание спецификации; создание репозитория.

**Среда 24.06.15**

Создание плана разработки; распределение обязанностей; начало разработки прототипа.

**Четверг 25.06.15**

Создание основных внешних интерфейсов; завершение работы над прототипом; создание плана тестирования; тестирование прототипа; начало разработки алгоритма.

**Пятница 26.06.15**

Начало разработки первой версии программы; связывание интерфейса и алгоритма.

Первая версия программы имеет следующий функционал:

* Интерфейс (все меню);
* ввод графа: генерация, ввод из файла;
* рабочий алгоритм;
* вывод графа.

**Суббота 27.06.15**

Разработка первой версии программы.

**Понедельник 29.06.15**

Завершение разработки первой версии программы; тестирование; начало составления отчета.

Планирование функционала второй (финальной) версии программы:

* ручной ввод графа;
* визуализация пошагового алгоритма.

Начало разработки второй (финальной) версии программы; составление (дополнение) отчета.

**Вторник 30.06.15**

Разработка второй (финальной) версии программы; составление (дополнение) отчета.

**Среда 01.07.15**

Завершение разработки второй (финальной) версии программы; тестирование; завершение составления отчета.

Сдача программы.

По необходимости: разработка третьей версии программы.

Резервные дни:

**Четверг 02.07.15**

**Пятница 03.07.15**

**Код**

**AntAlgorithm.java**

**package Algorithm;**

**import java.util.\*;**

**import Graph.Graph;**

**import Graph.Edge;**

**import Staff.Pair;**

**public class AntAlgorithm {**

**private Graph graph; // graph**

**private double greed;**

**private double gregariousness;**

**public double evaporationSpeed;**

**private int count; // steps counter**

**private int startIndex; // starting node index**

**private int finishIndex; // finish node index**

**private int currentIndex; // current node index**

**private int numberOfRandomAnts;**

**private int numberOfAnts;**

**public AntAlgorithm(Graph graph\_,**

**double greed\_,**

**double evaporationSpeed\_,**

**int numberOfRandomAnts\_,**

**int numberOfAnts\_,**

**Pair<Integer, Integer> path) { // constructor**

**graph = graph\_;**

**if (greed\_ <= 1) {**

**greed = greed\_;**

**gregariousness = 1 - greed;**

**} else {**

**greed = 0.5;**

**gregariousness = 0.5;**

**}**

**if (evaporationSpeed < 1)**

**evaporationSpeed = evaporationSpeed\_;**

**else**

**evaporationSpeed = 0.5;**

**numberOfRandomAnts = numberOfRandomAnts\_;**

**numberOfAnts = numberOfAnts\_;**

**startIndex = path.first;**

**finishIndex = path.second;**

**count = 0;**

**}**

**public boolean finished() {**

**return count == numberOfAnts;**

**}**

**public int getCount() {**

**return count;**

**}**

**public List<Integer> step() {**

**List<Integer> wayN = new ArrayList<>(); // way ant goes (nodes)**

**List<Edge> wayE = new ArrayList<>(); // way ant goes (edges)**

**List<Integer> nodesInPath = new ArrayList();;**

**wayN.add(startIndex);**

**currentIndex = startIndex;**

**List<Integer> bannedN = new ArrayList<>();**

**if (count < numberOfRandomAnts) { // first numberOfRandomAnts ants go randomly**

**while (currentIndex != finishIndex) { // while ant not find finish**

**List<Integer> possibleNN = new ArrayList<>(); // list of possible next nodes**

**for (int i = 0; i < graph.numberOfVertices; ++i)**

**if (graph.adjacencyMatrix[currentIndex][i] != 0) {**

**boolean ok = true; // true if an ant not been in possible next node**

**for (Integer j : wayN)**

**if (i == j) {**

**ok = false; // already been here**

**break;**

**}**

**if (ok) {**

**for (Integer j : bannedN)**

**if (j == i)**

**ok = false;**

**if (ok) possibleNN.add(i);**

**}**

**}**

**Random rand = new Random();**

**if (possibleNN.size() != 0) { // random choice**

**currentIndex = possibleNN.get(Math.abs(rand.nextInt()) % possibleNN.size());**

**wayN.add(currentIndex);**

**} else {**

**bannedN.add(currentIndex);**

**currentIndex = wayN.get(wayN.size() - 2);**

**wayN.remove(wayN.size() - 1);**

**wayE.remove(wayE.size() - 1);**

**possibleNN.clear();**

**continue;**

**}**

**for (Edge i : graph.edges) // adding edge to way**

**if ((i.firstNode == Math.min(currentIndex, wayN.get(wayN.size() - 2)))**

**&& (i.secondNode == Math.max(currentIndex, wayN.get(wayN.size() - 2)))) {**

**wayE.add(i);**

**break;**

**}**

**}**

**for(Edge i : wayE)**

**i.inCurrentPath = true;**

**for(Integer i : wayN)**

**nodesInPath.add(i);**

**wayN.clear();**

**double wayWeight = 0.0; // weight of way**

**for (Edge i : wayE) {**

**wayWeight += i.weight;**

**}**

**nodesInPath.add(new Integer((int)wayWeight));**

**for (Edge i : graph.edges) { // pheromone update**

**i.pheromone = (1.0 - evaporationSpeed) \* i.pheromone;**

**if (i.inCurrentPath) {**

**i.pheromone += 1.0 / wayWeight;**

**i.inCurrentPath = false;**

**}**

**}**

**bannedN.clear();**

**} else { // other steps, when the choice depends on the pheromone**

**while (currentIndex != finishIndex) { // while ant not find finish**

**List<Integer> possibleNN = new ArrayList<>(); // list of possible next nodes**

**List<Edge> possibleNE = new ArrayList<>(); // list of possible next edges**

**for (int i = 0; i < graph.numberOfVertices; ++i) {**

**if (graph.adjacencyMatrix[currentIndex][i] != 0) {**

**boolean ok = true; // true if an ant not been in possible next node**

**for (Integer j : wayN)**

**if (i == j) {**

**ok = false; // already been here**

**break;**

**}**

**if (ok) {**

**for (Integer j : bannedN)**

**if (j == i)**

**ok = false;**

**if (ok) possibleNN.add(i);**

**}**

**}**

**}**

**for (int j : possibleNN) {**

**for (Edge i : graph.edges)**

**if ((i.firstNode == Math.min(j, currentIndex))**

**&& (i.secondNode == Math.max(j, currentIndex))) {**

**possibleNE.add(i);**

**break;**

**}**

**}**

**List<Double> probability = new ArrayList<>(); //probabilities of going that way**

**double sum = 0.0;**

**for (Edge i : possibleNE) {**

**double k = Math.pow(i.pheromone,gregariousness) \* Math.pow (1.0 / i.weight, greed);**

**sum += k;**

**probability.add(k);**

**}**

**if (sum == 0.0) {**

**bannedN.add(currentIndex);**

**currentIndex = wayN.get(wayN.size() - 2);**

**wayN.remove(wayN.size() - 1);**

**wayE.remove(wayE.size() - 1);**

**possibleNN.clear();**

**continue;**

**}**

**for (int i = 0; i < probability.size(); ++i) {**

**double temp = probability.get(i);**

**probability.set(i, temp/sum);**

**}**

**for (int i = 1; i < probability.size(); ++i) {**

**double temp = probability.get(i);**

**probability.set(i, temp + probability.get(i - 1));**

**}**

**sum = Math.random(); // random real number [0,1)**

**for (int i = 0; i < probability.size(); ++i) // random choice**

**if ( probability.get(i) >= sum) {**

**if ( possibleNE.get(i).firstNode == wayN.get(wayN.size() - 1) )**

**currentIndex = possibleNE.get(i).secondNode;**

**else**

**currentIndex = possibleNE.get(i).firstNode;**

**break;**

**}**

**if (possibleNN.size() == 0) { // random choice**

**bannedN.add(currentIndex);**

**currentIndex = wayN.get(wayN.size() - 2);**

**wayN.remove(wayN.size() - 1);**

**wayE.remove(wayE.size() - 1);**

**possibleNN.clear();**

**continue;**

**}**

**for ( Edge i : graph.edges) // adding edge to way**

**if ( (i.firstNode == Math.min(currentIndex, wayN.get(wayN.size() - 1)) )**

**&& (i.secondNode == Math.max(currentIndex, wayN.get(wayN.size() - 1)) ) ){**

**wayE.add(i);**

**break;**

**}**

**wayN.add(currentIndex);**

**}**

**for(Edge i : wayE)**

**i.inCurrentPath = true;**

**for(Integer i : wayN)**

**nodesInPath.add(i);**

**wayN.clear();**

**double wayWeight = 0.0; // weight of way**

**for (Edge i : wayE) {**

**wayWeight += i.weight;**

**}**

**nodesInPath.add(new Integer((int)wayWeight));**

**for (Edge i : graph.edges) { // pheromone update**

**i.pheromone = (1.0 - evaporationSpeed) \* i.pheromone;**

**if (i.inCurrentPath){**

**i.pheromone += 1.0 / wayWeight;**

**i.inCurrentPath = false;}**

**}**

**bannedN.clear();**

**}**

**++count;**

**return nodesInPath;**

**}**

**public List<Integer> autoAlgorithm() {**

**while (count < numberOfAnts)**

**step();**

**return findPath();**

**}**

**public List<Integer> findPath() {**

**List<Integer> nodesInPath = new ArrayList<>();**

**List<Edge> probEdges = new ArrayList<>();**

**nodesInPath.add(startIndex);**

**List<Edge> banned = new ArrayList<>();**

**currentIndex = startIndex;**

**int wayWeight = 0; // weight of way**

**while (currentIndex != finishIndex) {**

**for (Edge i : graph.edges)**

**if ( i.firstNode == currentIndex || i.secondNode == currentIndex ){**

**boolean ok = true;**

**for (Edge k : banned)**

**if (k == i)**

**ok = false;**

**if (ok)**

**probEdges.add(i);**

**}**

**Edge edge = probEdges.get(0);**

**for (Edge i : probEdges)**

**if (i.pheromone > edge.pheromone)**

**edge = i;**

**banned.add(edge);**

**probEdges.clear();**

**wayWeight += edge.weight;**

**if (edge.firstNode == currentIndex)**

**nodesInPath.add(edge.secondNode);**

**else**

**nodesInPath.add(edge.firstNode);**

**currentIndex = nodesInPath.get(nodesInPath.size() - 1);**

**}**

**nodesInPath.add(wayWeight);**

**return nodesInPath; // path of ant**

**}**

**}**

**Edge.java**

**package** Graph;  
  
*// ребро***public class** Edge {  
 **public double pheromone**;  
 **public int weight**;  
 **public int firstNode**; *// от вершины* **public int secondNode**; *// до вершины* **public boolean inCurrentPath**; *// принадлежность к пути* **public** Edge(**int** weight\_, **int** firstNode\_, **int** secondNode\_) {  
 **pheromone** = 0.0;  
 **weight** = weight\_;  
 **firstNode** = firstNode\_;  
 **secondNode** = secondNode\_;  
 **inCurrentPath** = **false**;  
 }  
}

**Graph.java**

**package** Graph;  
  
**import** java.util.\*;  
  
**import** Staff.Pair;  
  
**public class** Graph {  
  
 **public static** Character[] *alphabet* = {**'A'**, *// алфавит для списков вершин* **'B'**, **'C'**, **'D'**, **'E'**, **'F'**, **'G'**, **'H'**, **'I'**, **'J'**};  
  
 **public int numberOfVertices**; *// Number of graph nodes* **public int**[][] **adjacencyMatrix**; *// Adjacency matrix with weights* **public boolean**[][] **linksMatrix**; *// Adjacency matrix with links only. We'll use Warshall algorithm to it.* **public** List<Edge> **edges**; *// Graph's edges* **public** List<Vertex> **vertices**; *// Graph's vertices* **public** Graph() { *// Constructor* **numberOfVertices** = 0;  
 **adjacencyMatrix** = **null**;  
 **linksMatrix** = **null**;  
 **edges** = **null**;  
 **vertices** = **null**;  
 }  
  
 **public** Graph(Graph g) { *// Copy constructor* **numberOfVertices** = g.**numberOfVertices**;  
  
 createMatrix(**numberOfVertices**);  
  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j) {  
 **adjacencyMatrix**[i][j] = g.**adjacencyMatrix**[i][j];  
 **linksMatrix**[i][j] = g.**linksMatrix**[i][j];  
 }  
  
 **if** (g.**edges** != **null**) **edges** = **new** ArrayList<>(g.**edges**);  
 **if** (g.**vertices** != **null**) **vertices** = **new** ArrayList<>(g.**vertices**);  
 }  
  
 **public boolean** isNull() {  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j)  
 **if** (**linksMatrix**[i][j])  
 **return false**;  
  
 **return true**; *// no edges* }  
  
 **public boolean** isCreated() { *// is graph created* **return numberOfVertices** != 0;  
 }  
  
 **public void** deleteGraph() {  
 **numberOfVertices** = 0;  
 **adjacencyMatrix** = **null**;  
 **linksMatrix** = **null**;  
 **edges** = **null**;  
 **vertices** = **null**;  
 }  
  
 **private void** createMatrix(**int** n) { *// create graph* **numberOfVertices** = n;  
 **adjacencyMatrix** = **new int**[n][n];  
 **linksMatrix** = **new boolean**[n][n];  
 }  
  
 **private void** createVerticesAndEdges() {  
 *// create vertices* **vertices** = **new** ArrayList<>(**numberOfVertices**);  
  
 **int** centerOfPanelX = 342;  
 **int** centerOfPanelY = 330;  
 **int** distanceToCenterOfVertex = 290;  
  
 *// координаты вершины* **int** vertexX = centerOfPanelX;  
 **int** vertexY = centerOfPanelY - distanceToCenterOfVertex;  
  
 *// дополнительные переменные* **double** radNextX = vertexX - centerOfPanelX;  
 **double** radNextY = vertexY - centerOfPanelY;  
 **double** polarX, polarY;  
 **double** tempX, tempY;  
  
 **for**(**int** i = 0; i < **numberOfVertices** ; ++i) {  
 **vertices**.add(**new** Vertex(vertexX - 25, vertexY - 25, *alphabet*[i]));  
  
 polarX = 1 \* Math.*cos*(Math.*acos*(-1.0) \* 2 / **numberOfVertices**);  
 polarY = 1 \* Math.*sin*(Math.*acos*(-1.0) \* 2 / **numberOfVertices**);  
  
 tempX = radNextX \* polarX - radNextY \* polarY;  
 tempY = radNextX \* polarY + radNextY \* polarX;  
  
 radNextX = tempX;  
 radNextY = tempY;  
  
 vertexX = (**int**)(centerOfPanelX + radNextX);  
 vertexY = (**int**)(centerOfPanelY + radNextY);  
 }  
  
 *// create edges* **edges** = **new** ArrayList<>();  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = i; j < **numberOfVertices**; ++j)  
 **if** (**adjacencyMatrix**[i][j] != 0) {  
 **edges**.add(**new** Edge(**adjacencyMatrix**[i][j], i, j));  
 }  
 }  
  
 **private void** makeWarshallAlgorithm() { *// Warshall algorithm* **for** (**int** k = 0; k < **numberOfVertices**; ++k)  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j)  
 **linksMatrix**[i][j] = **linksMatrix**[i][j] || (**linksMatrix**[i][k] && **linksMatrix**[k][j]);  
  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **linksMatrix**[i][i] = **false**;  
 }  
  
 **public void** generateGraph(**int** numberOfVertices, **int** linksPercent,  
 **int** leftBound, **int** rightBound) { *// Graph generation* deleteGraph();  
  
 createMatrix(numberOfVertices);  
  
 Random rand = **new** Random(); *// random generator* **if** (linksPercent > 100) linksPercent = 100;  
  
 List<Pair<Integer, Integer> > list = **new** ArrayList<>();  
  
 **int** edges = numberOfVertices \* (numberOfVertices - 1) \* linksPercent / 200; *// 200 because graph isn't oriented* **for** (**int** i = 0; i < numberOfVertices; ++i)  
 **for** (**int** j = i; j < numberOfVertices; ++j)  
 **if** (i != j) list.add(**new** Pair(i, j));  
  
 Collections.*shuffle*(list); *// Random\_shuffle* **boolean** b = rand.nextBoolean();  
  
 **if**(b) {  
 **for**(**int** i = 0; i < edges; ++i) {  
 **linksMatrix**[list.get(0).**first**][list.get(0).**second**] = **true**;  
 list.remove(0);  
 }  
 } **else** {  
 **for**(**int** i = 0; i < edges; ++i) {  
 **linksMatrix**[list.get( list.size() - 1).**first**][list.get( list.size() - 1).**second**] = **true**;  
 list.remove(list.size() - 1);  
 }  
 }  
  
 **int** range = rightBound - leftBound;  
  
 **for** (**int** i = 0; i < numberOfVertices; ++i) *// Random weights* **for** (**int** j = 0; j < numberOfVertices; ++j)  
 **if** (**linksMatrix**[i][j])  
 **adjacencyMatrix**[i][j] = rand.nextInt(range + 1) + leftBound;  
  
 **for** (**int** i = 0; i < numberOfVertices; ++i) *// make matrix symmetric* **for** (**int** j = i; j < numberOfVertices; ++j) {  
 **linksMatrix**[j][i] = **linksMatrix**[i][j];  
 **adjacencyMatrix**[j][i] = **adjacencyMatrix**[i][j];  
 }  
  
 makeWarshallAlgorithm(); *// WarshallAlgorithm* createVerticesAndEdges();  
 }  
  
 **public boolean** createGraphFromFile(Scanner input) { *// create graph from file* deleteGraph();  
  
 **int**[] massOfDigits = **new int**[100];  
 **int** numberOfDigits = 0;  
  
 **while** (input.hasNextInt()) {  
 massOfDigits[numberOfDigits++] = input.nextInt();  
 }  
  
 **if** (Math.*sqrt*(numberOfDigits) % 1 == 0) {  
 **int** numberOfElements = (**int**)Math.*sqrt*(numberOfDigits);  
 **int**[][] elements = **new int**[numberOfElements][numberOfElements];  
  
 **if** (numberOfElements < 2) *// too few elements* **return false**;  
  
 **for** (**int** i = 0; i < numberOfElements; i++)  
 **for** (**int** j = 0; j < numberOfElements; j++)  
 elements[i][j] = massOfDigits[i \* numberOfElements + j];  
  
 **for** (**int** i = 0; i < numberOfElements; ++i)  
 **if** (elements[i][i] != 0) *// wrong elements on diagonal* **return false**;  
  
 **for** (**int** i = 0; i < numberOfElements; ++i)  
 **for** (**int** j = 0; j < numberOfElements; ++j)  
 **if** (elements[i][j] != elements[j][i]) *// wrong elements* **return false**;  
  
  
 *// real creation of graph* createMatrix(numberOfElements);  
 **for** (**int** i = 0; i < numberOfElements; i++)  
 **for** (**int** j = 0; j < numberOfElements; j++) {  
 **adjacencyMatrix**[i][j] = massOfDigits[i \* numberOfElements + j];  
 **linksMatrix**[i][j] = **adjacencyMatrix**[i][j] != 0;  
 }  
  
 makeWarshallAlgorithm();  
  
 createVerticesAndEdges();  
  
 **return true**; *// graph created* }  
  
 **return false**; *// graph is not created* }  
  
 **private boolean** isNullRow(**int**[][] mm, **int** index){  
 **for**(**int** i = 0; i < 10; ++i)  
 **if** (mm[index][i] != 0)  
 **return false**;  
  
 **for** (**int** i = 0; i < **vertices**.size(); i++)  
 **if** (**vertices**.get(i).**letter** == Graph.*alphabet*[index])  
 **return false**;  
  
 **return true**;  
 }  
  
 **public void** removePheromone() {  
 **for** (Edge i : **edges**)  
 i.**pheromone** = 0;  
 }  
  
 **public void** createMatrixFromEdgesAndVertices() {  
 **int**[][] adjacencyMatrixTmp = **new int**[10][10];  
  
 **for** (**int** i = 0; i < 10; ++i)  
 **for** (**int** j = 0; j < 10; ++j)  
 adjacencyMatrixTmp[i][j] = 0;  
  
 **for** (Edge i : **edges**) {  
 adjacencyMatrixTmp[i.**firstNode**][i.**secondNode**] = i.**weight**;  
 adjacencyMatrixTmp[i.**secondNode**][i.**firstNode**] = i.**weight**;  
 }  
  
 **adjacencyMatrix** = **new int**[**numberOfVertices**][**numberOfVertices**];  
 **linksMatrix** = **new boolean**[**numberOfVertices**][**numberOfVertices**];  
  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j) {  
 **adjacencyMatrix**[i][j] = 0;  
 **linksMatrix**[i][j] = **false**;  
 }  
  
 **int**[] mass = **new int**[**numberOfVertices**];  
  
 **int** ind = 0;  
  
 **for** (**int** i = 0; i < 10; ++i)  
 **if** (!isNullRow(adjacencyMatrixTmp, i))  
 mass[ind++] = i;  
  
 **int** indexI = 0, indexJ = 0;  
 **for** (**int** i = 0; i < 10; i++) {  
 **if** (isNullRow(adjacencyMatrixTmp, i))  
 **continue**;  
  
 **for** ( **int** j : mass) {  
 **adjacencyMatrix**[indexI][indexJ] = adjacencyMatrixTmp[i][j];  
 indexJ++;  
 }  
 indexJ = 0;  
 indexI++;  
 }  
  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = 0; j < **numberOfVertices**; ++j) {  
 **if** (**adjacencyMatrix**[i][j] != 0)  
 **linksMatrix**[i][j] = **true**;  
 }  
  
 makeWarshallAlgorithm();  
  
 *// create edges* **edges** = **new** ArrayList<>();  
 **for** (**int** i = 0; i < **numberOfVertices**; ++i)  
 **for** (**int** j = i; j < **numberOfVertices**; ++j)  
 **if** (**adjacencyMatrix**[i][j] != 0) {  
 **edges**.add(**new** Edge(**adjacencyMatrix**[i][j], i, j));  
 }  
 }  
}

**Vertex.java**

**package** Graph;  
  
**public class** Vertex {  
 **public** Character **letter**;  
 **public int coordX**;  
 **public int coordY**;  
  
 **public** Vertex(**int** x, **int** y, Character letter\_) {  
 **coordX** = x;  
 **coordY** = y;  
 **letter** = letter\_;  
 }  
}

**GraphEnterFrame.java**

**package** GUI.GraphEnter;  
  
**import** javax.swing.\*;  
**import** javax.swing.border.TitledBorder;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.util.ArrayList;  
  
**import** GUI.ParametersFrame;  
**import** Graph.\*;  
**import** Staff.Pair;  
  
*// окно рисования графа***public class** GraphEnterFrame **extends** JFrame  
 **implements** ActionListener, MouseMotionListener, MouseListener {  
  
 **private final** ParametersFrame **parametersFrame**; *// ссылка на окно-родитель* Graph **graph**; *// ссылка на граф  
  
 // панель* **private** JPanel **graphPanel**;  
  
 *// кнопки* **private** JButton **btnAdd**;  
 **private** JButton **btnDel**;  
 **private** JButton **btnAddEdge**;  
 **private** JButton **btnDelEdge**;  
 **private** JButton **btnConfirm**;  
 **private boolean btnDelPressed** = **false**;  
 **private boolean btnAddEdgePressed** = **false**;  
 **private boolean btnDelEdgePressed** = **false**;  
 **private boolean btnConfirmPressed** = **false**;  
  
 *// для ввода веса* **private** JLabel infoLabel;  
 **private** JTextField **weightTextField**;  
  
 **int counterVerMax** = 0;  
 **int counterEdgeMax** = 0;  
 **int counterEdge** = 0;  
  
 **int dx**;  
 **int dy**;  
  
 **boolean check** = **false**;  
  
 **boolean interrupt** = **false**; *// для закрытия потока* **boolean removeEdges** = **false**;  
  
 VertexPanel[] **vertices** = **new** VertexPanel[10]; *// вершины* Pair<Integer, Integer>[] **edges** = **new** Pair[100]; *// ребра* **int**[] **weights** = **new int**[100]; *// веса  
  
 // конструктор* **public** GraphEnterFrame(ParametersFrame parametersFrame\_, Graph graph\_) {  
 **super**(**"Рисование графа"**);  
  
 **parametersFrame** = parametersFrame\_;  
 **graph** = graph\_;  
  
 setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
 setPreferredSize(**new** Dimension(1000, 700));  
 setResizable(**false**);  
 setLayout(**null**);  
  
 *// действие при закрытии окна* addWindowListener(**new** WindowListener() {  
 **public void** windowClosing(WindowEvent event) {  
 **interrupt** = **true**; *// для остановки потока* event.getWindow().setVisible(**false**);  
 event.getWindow().dispose();  
 **parametersFrame**.setVisible(**true**);  
 }  
 **public void** windowActivated(WindowEvent event) { }  
 **public void** windowClosed(WindowEvent event) { }  
 **public void** windowDeactivated(WindowEvent event) { }  
 **public void** windowDeiconified(WindowEvent event) { }  
 **public void** windowIconified(WindowEvent event) { }  
 **public void** windowOpened(WindowEvent event) { }  
 });  
  
 *// иконка* setIconImage(Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/icon.png"**)));  
  
  
 *// боковая панель* JPanel sidePanel = **new** JPanel();  
 sidePanel.setBounds(10, 10, 280, 660);  
 sidePanel.setLayout(**null**);  
 TitledBorder sidePanelTitle = BorderFactory.*createTitledBorder*(**""**);  
 sidePanel.setBorder(sidePanelTitle);  
  
 **btnAdd** = **new** JButton(**"Добавить вершину"**);  
 **btnAdd**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **btnAdd**.setBounds(20, 200, 240, 60);  
 **btnAdd**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **btnAdd**.setActionCommand(**"AddVertexButton"**);  
 **btnAdd**.addActionListener(**this**);  
 sidePanel.add(**btnAdd**);  
  
 **btnDel** = **new** JButton(**"Удалить вершину"**);  
 **btnDel**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **btnDel**.setBounds(20, 270, 240, 60);  
 **btnDel**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **btnDel**.setActionCommand(**"DeleteVertexButton"**);  
 **btnDel**.addActionListener(**this**);  
 sidePanel.add(**btnDel**);  
  
 **btnAddEdge** = **new** JButton(**"Добавить ребро"**);  
 **btnAddEdge**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **btnAddEdge**.setBounds(20, 340, 240, 60);  
 **btnAddEdge**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **btnAddEdge**.setActionCommand(**"AddEdgeButton"**);  
 **btnAddEdge**.addActionListener(**this**);  
 **btnAddEdge**.setEnabled(**false**);  
 sidePanel.add(**btnAddEdge**);  
  
 **btnDelEdge** = **new** JButton(**"Удалить ребро"**);  
 **btnDelEdge**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **btnDelEdge**.setBounds(20, 410, 240, 60);  
 **btnDelEdge**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **btnDelEdge**.setActionCommand(**"DeleteEdgeButton"**);  
 **btnDelEdge**.addActionListener(**this**);  
 **btnDelEdge**.setEnabled(**false**);  
 sidePanel.add(**btnDelEdge**);  
  
 JButton saveAndExitButton = **new** JButton(**"Сохранить и выйти"**);  
 saveAndExitButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 saveAndExitButton.setBounds(20, 510, 240, 60);  
 saveAndExitButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 saveAndExitButton.setActionCommand(**"SaveAndExitButton"**);  
 saveAndExitButton.addActionListener(**this**);  
 sidePanel.add(saveAndExitButton);  
  
 JButton exitButton = **new** JButton(**"Выйти без сохранения"**);  
 exitButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 exitButton.setBounds(20, 580, 240, 60);  
 exitButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 exitButton.setActionCommand(**"ExitButton"**);  
 exitButton.addActionListener(**this**);  
 sidePanel.add(exitButton);  
  
  
 *// панель для вывода информации* JPanel textPanel = **new** JPanel();  
 textPanel.setBounds(20, 20, 240, 140);  
 textPanel.setLayout(**null**);  
 textPanel.setBorder(BorderFactory.*createTitledBorder*(**""**));  
  
 infoLabel = **new** JLabel(**""**);  
 infoLabel.setHorizontalAlignment(JLabel.***CENTER***);  
 infoLabel.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 infoLabel.setBounds(20, 20, 200, 50);  
 textPanel.add(infoLabel);  
  
 **weightTextField** = **new** JTextField();  
 **weightTextField**.setBounds(30, 90, 70, 30);  
 **weightTextField**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***TEXT\_CURSOR***));  
 **weightTextField**.setVisible(**false**);  
 textPanel.add(**weightTextField**);  
  
 **btnConfirm** = **new** JButton(**"Ок"**);  
 **btnAdd**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **btnConfirm**.setBounds(140, 90, 70, 30);  
 **btnConfirm**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **btnConfirm**.setActionCommand(**"Confirm"**);  
 **btnConfirm**.addActionListener(**this**);  
 **btnConfirm**.setVisible(**false**);  
 textPanel.add(**btnConfirm**);  
  
  
 *// панель для рисовки графа* **graphPanel** = **new** JPanel();  
 **graphPanel**.setBorder(BorderFactory.*createTitledBorder*(**""**));  
 **graphPanel**.setBounds(300, 10, 685, 660);  
 **graphPanel**.setLayout(**null**);  
  
  
 sidePanel.add(textPanel);  
 add(sidePanel);  
 add(**graphPanel**);  
  
  
 *// поток, обновляющий граф* Runnable r1 = () -> {  
 *// задержка для прорисовки* **try** {  
 Thread.*sleep*(200);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **while**(!**interrupt**) {  
 **int** free = 0;  
  
 **while** (**removeEdges**) {  
 **try** {  
 Thread.*sleep*(1);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 Graphics gr = **graphPanel**.getGraphics();  
  
 *// задержка для прорисовки* **if** (**counterEdgeMax** == 1) {  
 **try** {  
 Thread.*sleep*(1);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **while** (free < **counterEdgeMax**) {  
 **if** (**edges**[free] != **null**) {  
  
 **int** x1 = **vertices**[**edges**[free].**first**].getX() + 25;  
 **int** y1 = **vertices**[**edges**[free].**first**].getY() + 25;  
 **int** x2 = **vertices**[**edges**[free].**second**].getX() + 25;  
 **int** y2 = **vertices**[**edges**[free].**second**].getY() + 25;  
  
 **int** x11= (**int**)(x1 + 25 \* (x2 - x1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** y11 = (**int**)(y1 + 25 \* (y2 - y1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** x22 = (**int**)(x2 + 25 \* (x1 - x2) /  
 Math.*sqrt*(Math.*pow*((**double**)(x1 - x2), 2.0)  
 + Math.*pow*((**double**)(y1 - y2), 2.0)));  
  
 **int** y22 = (**int**)(y2 + 25 \* (y1 - y2) /  
 Math.*sqrt*(Math.*pow*((**double**)(x1 - x2), 2.0)  
 + Math.*pow*((**double**)(y1 - y2), 2.0)));  
  
 gr.setColor(Color.***BLACK***);  
 gr.drawLine(x11, y11, x22, y22);  
  
 **int** centerX = (**int**)(x1 + 50 \* (x2 - x1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** centerY = (**int**)(y1 + 50 \* (y2 - y1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 gr.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 15));  
 gr.setColor(Color.***RED***);  
 gr.drawString(Integer.*toString*(**weights**[free]), centerX, centerY);  
 }  
  
 free++;  
 }  
  
 **if** (**counterVerMax** == 10)  
 **btnAdd**.setEnabled(**false**);  
  
 **if** (**counterVerMax** == 0)  
 **btnDel**.setEnabled(**false**);  
 **else  
 btnDel**.setEnabled(**true**);  
  
 **if** ((**counterEdge** == (**counterVerMax**\*(**counterVerMax** - 1) / 2)) || (**counterVerMax** < 2))  
 **btnAddEdge**.setEnabled(**false**);  
 **else  
 btnAddEdge**.setEnabled(**true**);  
  
 **if**(**counterEdge** < 1)  
 **btnDelEdge**.setEnabled(**false**);  
 }  
 };  
  
 *// запуск потока рисовки графа* Thread t1 = **new** Thread(r1);  
 t1.start();  
  
  
 pack();  
 setLocationRelativeTo(**null**); *// центрирование окна* setVisible(**true**);  
  
  
 *// если граф создан - нарисовать его* **if** (**graph**.isCreated()) {  
 *// вершины* **for** (**int** i = 0; i < **graph**.**numberOfVertices**; i++) {  
 **vertices**[i] = **new** VertexPanel(**graph**.**vertices**.get(i).**coordX**, **graph**.**vertices**.get(i).**coordY**);  
 **vertices**[i].setLetter(**graph**.**vertices**.get(i).**letter**);  
 **vertices**[i].setName(String.*valueOf*(i));  
 **vertices**[i].addMouseMotionListener(**this**);  
 **vertices**[i].addMouseListener(**this**);  
 **graphPanel**.add(**vertices**[i]);  
 **counterVerMax**++;  
 }  
  
 *// ребра и веса* **for** (**int** i = 0; i < **graph**.**edges**.size(); i++) {  
 **edges**[i] = **new** Pair<>(**graph**.**edges**.get(i).**firstNode**, **graph**.**edges**.get(i).**secondNode**);  
 **weights**[i] = **graph**.**edges**.get(i).**weight**;  
 **counterEdgeMax**++;  
 }  
 }  
 }  
  
 **private void** saveGraph() {  
 **graph**.**edges** = **new** ArrayList<>();  
 **graph**.**vertices** = **new** ArrayList<>();  
  
 *// вершины* **graph**.**numberOfVertices** = **counterVerMax**;  
 **for** (**int** i = 0; i < 10; i++) {  
 **if** (**vertices**[i] != **null**)  
 **graph**.**vertices**.add(**new** Vertex(**vertices**[i].getX(), **vertices**[i].getY(), **vertices**[i].getLetter()));  
 }  
  
 *// ребра* **for** (**int** i = 0; i < **counterEdge**; i++) {  
 **graph**.**edges**.add(**new** Edge(**weights**[i], **edges**[i].**first**, **edges**[i].**second**));  
 }  
  
 **graph**.createMatrixFromEdgesAndVertices();  
 }  
  
  
 **public void** mouseEntered(MouseEvent e) { }  
 **public void** mouseMoved(MouseEvent e) { }  
 **public void** mouseExited(MouseEvent e) { }  
 **public void** mousePressed(MouseEvent e) { }  
  
 **public void** mouseReleased(MouseEvent e) {  
 **graphPanel**.repaint();  
 }  
  
 **public void** mouseClicked(MouseEvent e) {  
 **if**(**btnDelPressed**) {  
 **removeEdges** = **true**;  
  
 Object ob = e.getSource();  
 VertexPanel vertex = (VertexPanel) ob;  
 **int** count = Integer.*parseInt*(vertex.getName());  
  
 **graphPanel**.remove(**vertices**[count]);  
 **vertices**[count] = **null**;  
 **btnAdd**.setEnabled(**true**);  
 **counterVerMax**--;  
  
 **for** (**int** i = 0; i < **counterEdgeMax**;i++) {  
 **if** (**edges**[i] != **null**)  
 **if** (count == **edges**[i].**first** || count == **edges**[i].**second**) {  
 **edges**[i] = **null**;  
 **counterEdge**--;  
 }  
 }  
  
 **removeEdges** = **false**;  
  
 **graphPanel**.repaint();  
 **btnDelPressed** = **false**;  
  
 infoLabel.setText(**"<html><p align=\"center\">Удалена вершина "** + Graph.*alphabet*[count] + **"</p><html>"**);  
  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***DEFAULT\_CURSOR***));  
 }  
  
 **if**(**btnAddEdgePressed**) {  
 **removeEdges** = **true**;  
  
 **if**(**btnConfirmPressed**) {  
 Object ob = e.getSource();  
 VertexPanel vertex = (VertexPanel) ob;  
 **int** count = Integer.*parseInt*(vertex.getName());  
  
 **if** (!**check**) {  
 infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count] + **".<br>Выберите вторую вершину</p><html>"**);  
 **dx** = count;  
 **check** = **true**;  
 } **else** {  
 **boolean** tmp = **false**;  
 **for** (**int** i = 0; i < **counterEdgeMax**;i++) {  
 **if** (**edges**[i] != **null**) {  
 **if** ((**dx** == **edges**[i].**first** && count == **edges**[i].**second**)  
 || (count == **edges**[i].**first** && **dx** == **edges**[i].**second**)) {  
 tmp = **true**;  
 **break**;  
 } **else if**(**dx** == count) {  
 tmp = **true**;  
 **break**;  
 }  
 }  
 }  
 **if**(!tmp) {  
 infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count]  
 + **".<br>Создано ребро "** + Graph.*alphabet*[**dx**] + **"-"** + Graph.*alphabet*[count] + **"</p><html>"**);  
 **int** free = 0;  
 **for** (**int** i = 0; i < **counterEdgeMax**; i++) {  
 **if** (**edges**[i] == **null**) {  
 free = i;  
 **break**;  
 }  
 free++;  
 }  
  
 **edges**[free] = **new** Pair<>(**dx**, count);  
 **counterEdge**++;  
 **counterEdgeMax**++;  
 } **else** {  
 **if**(**dx** == count)  
 infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count] + **".<br>Действие отменено!</p><html>"**);  
 **else** infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count] + **".<br>Ребро "** + Graph.*alphabet*[**dx**] + **"-"** + Graph.*alphabet*[count] + **" уже существует!</p><html>"**);  
 }  
  
 **check** = **false**;  
 **btnAddEdgePressed** = **false**;  
 **btnConfirmPressed** = **false**;  
 **btnDelEdge**.setEnabled(**true**);  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***DEFAULT\_CURSOR***));  
  
 }  
 }  
  
 **removeEdges** = **false**;  
 }  
  
 **if** (**btnDelEdgePressed**) {  
 **removeEdges** = **true**;  
  
 Object ob = e.getSource();  
 VertexPanel vertex = (VertexPanel) ob;  
 **int** count = Integer.*parseInt*(vertex.getName());  
  
 **if**(!**check**) {  
 infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count] + **".<br>Выберите вторую вершину</p><html>"**);  
 **dx** = count;  
 **check** = **true**;  
 } **else** {  
 **boolean** tmp = **false**;  
 **for**(**int** i = 0; i < **counterEdgeMax**;i++) {  
 **if**(**edges**[i] != **null**) {  
 **if** (**dx** == **edges**[i].**first** && count == **edges**[i].**second**) {  
 **edges**[i] = **null**;  
 tmp = **true**;  
 **counterEdge**--;  
 } **else if** (count == **edges**[i].**first** && **dx** == **edges**[i].**second**) {  
 **edges**[i] = **null**;  
 tmp = **true**;  
 **counterEdge**--;  
 }  
 }  
 }  
  
 **if**(tmp)  
 infoLabel.setText(**"<html><p align=\"center\">Выбрана вершина "** + Graph.*alphabet*[count] + **".<br>Удалено ребро "** + Graph.*alphabet*[**dx**] + **"-"** + Graph.*alphabet*[count] + **"</p><html>"**);  
 **else** infoLabel.setText(**"<html><p align=\"center\">Данного ребра не существует!</p><html>"**);  
  
 **check** = **false**;  
 **btnDelEdgePressed** = **false**;  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***DEFAULT\_CURSOR***));  
 **graphPanel**.repaint();  
 }  
  
 **removeEdges** = **false**;  
 }  
 }  
  
 **public void** mouseDragged(MouseEvent e) {  
 **if** (!**btnAddEdgePressed** && !**btnDelEdgePressed** && !**btnDelPressed**) {  
 Object source = e.getSource();  
 VertexPanel vertex = (VertexPanel) source;  
 **int** count = Integer.*parseInt*(vertex.getName());  
  
 **int** newX = e.getX();  
 **int** newY = e.getY();  
  
 **if** (newX < 25) {  
 **dx** = newX - 25 + **vertices**[count].getX();  
 **if** (**dx** < 0)  
 **dx** = 0;  
 **vertices**[count].setX(**dx**);  
 } **else if** ((newX - 25 + **vertices**[count].getX() + **vertices**[count].getWidth()) > **graphPanel**.getWidth()) {  
 **dx** = **graphPanel**.getWidth() - **vertices**[count].getWidth();  
 **vertices**[count].setX(**dx**);  
 } **else** {  
 **dx** = newX - 25 + **vertices**[count].getX();  
 **vertices**[count].setX(**dx**);  
 }  
  
 **if** (newY < 25) {  
 **dy** = newY - 25 + **vertices**[count].getY();  
 **if** (**dy** < 0)  
 **dy** = 0;  
 **vertices**[count].setY(**dy**);  
 } **else if** ((newY - 25 + **vertices**[count].getY() + **vertices**[count].getHeight()) > **graphPanel**.getHeight()) {  
 **dy** = **graphPanel**.getHeight() - **vertices**[count].getHeight();  
 **vertices**[count].setY(**dy**);  
 } **else** {  
 **dy** = newY - 25 + **vertices**[count].getY();  
 **vertices**[count].setY(**dy**);  
 }  
  
 **vertices**[count].setLocation(**dx**, **dy**);  
 **graphPanel**.repaint();  
 }  
 }  
  
  
 **public void** actionPerformed(ActionEvent e) {  
 String cmd = e.getActionCommand();  
  
 **switch**(cmd) {  
 **case "AddVertexButton"**:  
 **weightTextField**.setVisible(**false**);  
 **btnConfirm**.setVisible(**false**);  
  
 **int** free = 0;  
 **for** (**int** i = 0; i < 10; i++) {  
 **if** (**vertices**[i] == **null**) {  
 free = i;  
 **break**;  
 }  
 }  
  
 infoLabel.setText(**"<html><p align=\"center\">Добавлена вершина "** + Graph.*alphabet*[free] + **"</p><html>"**);  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***DEFAULT\_CURSOR***));  
  
 **vertices**[free] = **new** VertexPanel(0, 0);  
 **vertices**[free].setName(String.*valueOf*(free));  
 **vertices**[free].addMouseMotionListener(**this**);  
 **vertices**[free].addMouseListener(**this**);  
 **vertices**[free].setLetter(Graph.*alphabet*[free]);  
  
 **graphPanel**.add(**vertices**[free]);  
 **counterVerMax**++;  
 **graphPanel**.repaint();  
  
 **btnDelPressed** = **false**;  
 **btnAddEdgePressed** = **false**;  
 **btnDelEdgePressed** = **false**;  
 **break**;  
  
 **case "DeleteVertexButton"**:  
 **weightTextField**.setVisible(**false**);  
 **btnConfirm**.setVisible(**false**);  
  
 infoLabel.setText(**"<html><p align=\"center\">Кликните на вершину,<br>которую хотите удалить</p><html>"**);  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***CROSSHAIR\_CURSOR***));  
  
 **btnDelPressed** = **true**;  
 **btnAddEdgePressed** = **false**;  
 **btnDelEdgePressed** = **false**;  
 **break**;  
  
 **case "AddEdgeButton"**:  
 **weightTextField**.setVisible(**true**);  
 **btnConfirm**.setVisible(**true**);  
  
 infoLabel.setText(**"<html><p align=\"center\">Введите вес ребра<br>(от 1 до 100)</p><html>"**);  
  
 **btnDelPressed** = **false**;  
 **btnAddEdgePressed** = **true**;  
 **btnDelEdgePressed** = **false**;  
 **break**;  
  
 **case "DeleteEdgeButton"**:  
 **weightTextField**.setVisible(**false**);  
 **btnConfirm**.setVisible(**false**);  
  
 infoLabel.setText(**"<html><p align=\"center\">Кликните на 2 вершины,<br>которые связывает данное ребро</p><html>"**);  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***CROSSHAIR\_CURSOR***));  
  
 **btnDelPressed** = **false**;  
 **btnAddEdgePressed** = **false**;  
 **btnDelEdgePressed** = **true**;  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***CROSSHAIR\_CURSOR***));  
 **break**;  
  
 **case "Confirm"**:  
 **try** {  
 **if** (Integer.*parseInt*(**weightTextField**.getText()) <= 100  
 && Integer.*parseInt*(**weightTextField**.getText()) > 0) {  
 **btnConfirmPressed** = **true**;  
 free = 0;  
 **for** (**int** i = 0; i < **counterEdgeMax** + 1; i++) {  
 **if** (**edges**[i] == **null**) {  
 free = i;  
 **break**;  
 }  
 }  
 **weights**[free] = Integer.*parseInt*(**weightTextField**.getText());  
  
 **weightTextField**.setVisible(**false**);  
 **btnConfirm**.setVisible(**false**);  
  
 infoLabel.setText(**"<html><p align=\"center\">Кликните на 2 вершины,<br>которые хотите связать ребром</p><html>"**);  
 setCursor(Cursor.*getPredefinedCursor*(Cursor.***CROSSHAIR\_CURSOR***));  
 }  
  
 **weightTextField**.setText(**""**);  
  
 } **catch** (Exception e1) {  
 **weightTextField**.setText(**""**);  
 }  
 **break**;  
  
 **case "SaveAndExitButton"**:  
 **interrupt** = **true**; *// для остановки потока* saveGraph();  
 setVisible(**false**);  
 dispose();  
 **parametersFrame**.setVisible(**true**);  
 **break**;  
  
 **case "ExitButton"**:  
 **interrupt** = **true**; *// для остановки потока* setVisible(**false**);  
 dispose();  
 **parametersFrame**.setVisible(**true**);  
 **break**;  
 }  
 }  
}

**VertexPanel.java**

**package** GUI.GraphEnter;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** VertexPanel **extends** JPanel {  
 **private int X** = 0;  
 **private int Y** = 0;  
  
 **private** JLabel **letter**; *// название* **public** VertexPanel(**int** x, **int** y)  
 {  
 **X** = x;  
 **Y** = y;  
  
 setLayout(**null**);  
 setBounds(x, y, 50, 50);  
 setOpaque(**false**);  
  
 *// текстура* Image img = Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/vertex.png"**));  
 JLabel picLabel = **new** JLabel(**new** ImageIcon(img));  
 picLabel.setBounds(0, 0, 50, 50);  
 add(picLabel);  
  
 *// название* **letter** = **new** JLabel(**"A"**);  
 **letter**.setBounds(20,15,20,20);  
 add(**letter**);  
 }  
  
 **public int** getX(){  
 **return X**;  
 }  
  
 **public int** getY(){  
 **return Y**;  
 }  
  
 **public** Character getLetter() {  
 **return letter**.getText().toCharArray()[0];  
 }  
  
 **public void** setX(**int** x0){  
 **X** = x0;  
 }  
  
 **public void** setY(**int** y0){  
 **Y** = y0;  
 }  
  
 **public void** setLetter(**char** n) {  
 **letter**.setText(Character.*toString*(n));  
 }  
}

**AlgorithmFrame.java**

**package GUI;**

**import java.util.List;**

**import javax.swing.\*;**

**import javax.swing.border.TitledBorder;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.event.WindowListener;**

**import java.awt.event.WindowEvent;**

**import Algorithm.AntAlgorithm;**

**import Graph.\*;**

**import Staff.Pair;**

**// окно "алгоритм"**

**public class AlgorithmFrame extends JFrame implements ActionListener {**

**private final MainMenuFrame mainMenuFrame; // ссылка на главное окно**

**private Graph graph; // ссылка на граф**

**private Pair<Double, Double> params; // ссылка на параметры (жадность, скорость испарения)**

**private Pair<Integer, Integer> antParams; // ссылка на параметры муравьев**

**private AntAlgorithm algorithm; // алгоритм**

**// списки вершин**

**private JLabel startVertexLabel;**

**private JComboBox startVertexComboBox;**

**private JLabel endVertexLabel;**

**private JComboBox endVertexComboBox;**

**// кнопка "авто"**

**private JButton autoStepButton;**

**// кнопка "следующий шаг"**

**private JButton nextStepButton;**

**// путь муравья**

**private JLabel pathOfAntLabel;**

**private JLabel pathOfAntTextLabel;**

**GraphPanel graphPanel;**

**// панель с графом**

**class GraphPanel extends JPanel {**

**public GraphPanel() {**

**setBounds(300, 10, 685, 660);**

**setLayout(null);**

**TitledBorder graphPanelTitle = BorderFactory.createTitledBorder("");**

**setBorder(graphPanelTitle);**

**}**

**final BasicStroke stroke\_3 = new BasicStroke(3.0f);**

**final BasicStroke stroke\_7 = new BasicStroke(7.0f);**

**final BasicStroke stroke\_5 = new BasicStroke(5.0f);**

**final BasicStroke stroke\_11 = new BasicStroke(11.0f);**

**public void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**Graphics2D g2 = (Graphics2D)g;**

**// вершины**

**for (int i = 0; i < graph.numberOfVertices; i++) {**

**int x = graph.vertices.get(i).coordX;**

**int y = graph.vertices.get(i).coordY;**

**String name = String.valueOf(graph.vertices.get(i).letter);**

**g2.setStroke(stroke\_5);**

**g.drawOval(x, y, 50, 50);**

**g.setFont(new Font("Arial", Font.BOLD, 20));**

**g.drawString(name, x + 20, y + 30);**

**}**

**// ребра**

**for (int i = 0; i < graph.edges.size(); i++) {**

**int x1 = graph.vertices.get(graph.edges.get(i).firstNode).coordX + 25;**

**int y1 = graph.vertices.get(graph.edges.get(i).firstNode).coordY + 25;**

**int x2 = graph.vertices.get(graph.edges.get(i).secondNode).coordX + 25;**

**int y2 = graph.vertices.get(graph.edges.get(i).secondNode).coordY + 25;**

**int x11= (int)(x1 + 29 \* (x2 - x1) /**

**Math.sqrt(Math.pow((double)(x2 - x1), 2.0)**

**+ Math.pow((double)(y2 - y1), 2.0)));**

**int y11 = (int)(y1 + 29 \* (y2 - y1) /**

**Math.sqrt(Math.pow((double)(x2 - x1), 2.0)**

**+ Math.pow((double)(y2 - y1), 2.0)));**

**int x22 = (int)(x2 + 29 \* (x1 - x2) /**

**Math.sqrt(Math.pow((double)(x1 - x2), 2.0)**

**+ Math.pow((double)(y1 - y2), 2.0)));**

**int y22 = (int)(y2 + 29 \* (y1 - y2) /**

**Math.sqrt(Math.pow((double)(x1 - x2), 2.0)**

**+ Math.pow((double)(y1 - y2), 2.0)));**

**g2.setStroke(stroke\_11);**

**g2.setColor(Color.gray);**

**g2.drawLine(x11, y11, x22, y22);**

**g2.setStroke(stroke\_7);**

**g2.setColor(Color.white);**

**g2.drawLine(x11, y11, x22, y22);**

**int maxWeight = 0;**

**for (Edge e : graph.edges)**

**maxWeight = Math.max(maxWeight, e.weight);**

**int coef = (int)(algorithm.evaporationSpeed < 0.1 ? 1 : 10 \* algorithm.evaporationSpeed);**

**int green = (int)Math.max(0.0, ((graph.edges.get(i).pheromone <= 0.25) ? 255 : 255 - (graph.edges.get(i).pheromone - 0.25) \* maxWeight/10 \* coef \* 255));**

**int blue = (int)Math.max(0.0,((graph.edges.get(i).pheromone >= 0.25) ? 0 : 255 - graph.edges.get(i).pheromone \* maxWeight/10 \* coef \* 255));**

**g2.setStroke(stroke\_3);**

**g2.setColor(new Color(255, green, blue));**

**g2.drawLine(x11, y11, x22, y22);**

**int x= (int)(x1 + 70 \* (x2 - x1) /**

**Math.sqrt(Math.pow((double)(x2 - x1), 2.0)**

**+ Math.pow((double)(y2 - y1), 2.0)));**

**int y = (int)(y1 + 70 \* (y2 - y1) /**

**Math.sqrt(Math.pow((double)(x2 - x1), 2.0)**

**+ Math.pow((double)(y2 - y1), 2.0)));**

**// вес**

**g2.setColor(Color.RED);**

**g2.setFont(new Font("Arial", Font.BOLD, 20));**

**g2.drawString(String.valueOf(graph.edges.get(i).weight), x - 8, y + 8);**

**g2.setColor(Color.BLACK);**

**}**

**}**

**}**

**// конструктор**

**public AlgorithmFrame(MainMenuFrame mainMenuFrame\_, Graph graph\_,**

**Pair<Double, Double> params\_, Pair<Integer, Integer> antParams\_) {**

**super("Алгоритм");**

**mainMenuFrame = mainMenuFrame\_;**

**graph = graph\_;**

**params = params\_;**

**antParams = antParams\_;**

**// окно**

**setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);**

**setPreferredSize(new Dimension(1000, 700));**

**setResizable(false);**

**setLayout(null);**

**// действие при закрытии окна**

**addWindowListener(new WindowListener() {**

**public void windowClosing(WindowEvent event) {**

**event.getWindow().setVisible(false);**

**event.getWindow().dispose();**

**mainMenuFrame.setVisible(true);**

**}**

**public void windowActivated(WindowEvent event) {**

**}**

**public void windowClosed(WindowEvent event) {**

**}**

**public void windowDeactivated(WindowEvent event) {**

**}**

**public void windowDeiconified(WindowEvent event) {**

**}**

**public void windowIconified(WindowEvent event) {**

**}**

**public void windowOpened(WindowEvent event) {**

**}**

**});**

**// иконка**

**setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("/GUI/images/icon.png")));**

**// панель пути**

**JPanel antPathPanel = new JPanel();**

**antPathPanel.setBounds(10, 10, 280, 80);**

**antPathPanel.setLayout(null);**

**TitledBorder pathPanelTitle = BorderFactory.createTitledBorder("");**

**antPathPanel.setBorder(pathPanelTitle);**

**add(antPathPanel);**

**pathOfAntTextLabel = new JLabel();**

**pathOfAntTextLabel.setHorizontalAlignment(JLabel.CENTER);**

**pathOfAntTextLabel.setFont(new Font("Arial", Font.BOLD, 14));**

**pathOfAntTextLabel.setBounds(20, 10, 250, 30);**

**antPathPanel.add(pathOfAntTextLabel);**

**pathOfAntLabel = new JLabel();**

**pathOfAntLabel.setFont(new Font("Arial", Font.BOLD, 14));**

**pathOfAntLabel.setHorizontalAlignment(JLabel.CENTER);**

**pathOfAntLabel.setBounds(20, 35, 250, 30);**

**antPathPanel.add(pathOfAntLabel);**

**// боковая панель**

**JPanel sidePanel = new JPanel();**

**sidePanel.setBounds(10, 100, 280, 570);**

**sidePanel.setLayout(null);**

**TitledBorder sidePanelTitle = BorderFactory.createTitledBorder("");**

**sidePanel.setBorder(sidePanelTitle);**

**// списки вершин**

**startVertexLabel = new JLabel("Начальная вершина:");**

**startVertexLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**startVertexLabel.setBounds(20, 10, 240, 30);**

**sidePanel.add(startVertexLabel);**

**startVertexComboBox = new JComboBox();**

**startVertexComboBox.setFont(new Font("Arial", Font.PLAIN, 14));**

**startVertexComboBox.setBounds(20, 40, 240, 30);**

**// заполнение списка**

**for(int i = 0; i < graph.numberOfVertices; ++i)**

**for(int j = 0; j < graph.numberOfVertices; ++j)**

**if (graph.linksMatrix[i][j]) {**

**startVertexComboBox.addItem(graph.vertices.get(i).letter);**

**break;**

**}**

**if (startVertexComboBox.getItemCount() != 0) startVertexComboBox.setSelectedIndex(0);**

**startVertexComboBox.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**startVertexComboBox.setName("StartVertexComboBox");**

**startVertexComboBox.addActionListener(this);**

**sidePanel.add(startVertexComboBox);**

**endVertexLabel = new JLabel("Конечная вершина:");**

**endVertexLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**endVertexLabel.setBounds(20, 80, 240, 30);**

**sidePanel.add(endVertexLabel);**

**endVertexComboBox = new JComboBox();**

**endVertexComboBox.setFont(new Font("Arial", Font.PLAIN, 14));**

**endVertexComboBox.setBounds(20, 110, 240, 30);**

**// заполнение списка**

**int index = 0;**

**for (int i = 0; i < graph.vertices.size(); i++)**

**if (startVertexComboBox.getSelectedItem() == graph.vertices.get(i).letter) {**

**index = i;**

**break;**

**}**

**for(int i = 0; i < graph.numberOfVertices; ++i)**

**if (graph.linksMatrix[index][i])**

**endVertexComboBox.addItem(graph.vertices.get(i).letter);**

**if (endVertexComboBox.getItemCount() != 0) endVertexComboBox.setSelectedIndex(0);**

**endVertexComboBox.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**endVertexComboBox.setName("EndVertexComboBox");**

**endVertexComboBox.addActionListener(this);**

**sidePanel.add(endVertexComboBox);**

**setAlgorithm(); // настроить алгоритм под списки вершин**

**// кнопки**

**nextStepButton = new JButton("Следующий шаг");**

**nextStepButton.setFont(new Font("Arial", Font.BOLD, 14));**

**nextStepButton.setBounds(20, 200, 240, 60);**

**nextStepButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**nextStepButton.setActionCommand("NextStepButton");**

**nextStepButton.addActionListener(this);**

**sidePanel.add(nextStepButton);**

**autoStepButton = new JButton("Авто");**

**autoStepButton.setFont(new Font("Arial", Font.BOLD, 14));**

**autoStepButton.setBounds(20, 270, 240, 60);**

**autoStepButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**autoStepButton.setActionCommand("AutoStepButton");**

**autoStepButton.addActionListener(this);**

**sidePanel.add(autoStepButton);**

**JButton clearButton = new JButton("Очистить");**

**clearButton.setFont(new Font("Arial", Font.BOLD, 14));**

**clearButton.setBounds(20, 340, 240, 60);**

**clearButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**clearButton.setActionCommand("ClearButton");**

**clearButton.addActionListener(this);**

**sidePanel.add(clearButton);**

**JButton backButton = new JButton("В главное меню");**

**backButton.setFont(new Font("Arial", Font.BOLD, 14));**

**backButton.setBounds(20, 470, 240, 60);**

**backButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**backButton.setActionCommand("BackButton");**

**backButton.addActionListener(this);**

**sidePanel.add(backButton);**

**// панель для графа**

**graphPanel = new GraphPanel();**

**add(sidePanel);**

**add(graphPanel);**

**pack();**

**setLocationRelativeTo(null); // центрирование окна**

**setVisible(true);**

**}**

**// заблокировать компоненты (combo boxes)**

**private void disableComboBoxes() {**

**startVertexLabel.setEnabled(false);**

**startVertexComboBox.setEnabled(false);**

**endVertexLabel.setEnabled(false);**

**endVertexComboBox.setEnabled(false);**

**}**

**// разблокировать компоненты (combo boxes)**

**private void enableComboBoxes() {**

**startVertexLabel.setEnabled(true);**

**startVertexComboBox.setEnabled(true);**

**endVertexLabel.setEnabled(true);**

**endVertexComboBox.setEnabled(true);**

**}**

**// задать вершины для алгоритма**

**private void setAlgorithm() {**

**Integer first = 0;**

**Integer second = 0;**

**for (int i = 0; i < graph.vertices.size(); i++) {**

**if (startVertexComboBox.getSelectedItem() == graph.vertices.get(i).letter)**

**first = i;**

**if (endVertexComboBox.getSelectedItem() == graph.vertices.get(i).letter)**

**second = i;**

**}**

**Pair<Integer, Integer> path = new Pair<>(first, second);**

**graph.removePheromone();**

**algorithm = new AntAlgorithm(new Graph(graph), params.first, params.second,**

**antParams.first, antParams.second, path);**

**}**

**private void showPathOfAnt(List<Integer> pathOfAnt) {**

**String path = "<html><p align=\"center\">";**

**int weight = pathOfAnt.get(pathOfAnt.size()-1);**

**pathOfAnt.remove(pathOfAnt.size() - 1);**

**for (Integer i : pathOfAnt) {**

**if (!path.equals("<html><p align=\"center\">")) path += " - ";**

**path += graph.vertices.get(i).letter;**

**}**

**path += "<br>Вес пути: " + String.valueOf(weight) + "</p><html>";**

**pathOfAntTextLabel.setText("Найденный путь: ");**

**pathOfAntLabel.setText(path);**

**}**

**private void showAnt(List<Integer> pathOfAnt) {**

**String path = "<html><p align=\"center\">";**

**int weight = pathOfAnt.get(pathOfAnt.size()-1);**

**pathOfAnt.remove(pathOfAnt.size() - 1);**

**for (Integer i : pathOfAnt) {**

**if (!path.equals("<html><p align=\"center\">")) path += " - ";**

**path += graph.vertices.get(i).letter;**

**}**

**path += "<br>Вес пути: " + String.valueOf(weight) + "</p><html>";**

**if (algorithm.getCount() <= antParams.first)**

**pathOfAntTextLabel.setText("\"Блиц\" муравей "**

**+ algorithm.getCount() + ". Путь: ");**

**else**

**pathOfAntTextLabel.setText("Муравей "**

**+ algorithm.getCount() + ". Путь: ");**

**pathOfAntLabel.setText(path);**

**}**

**// нажатие кнопки**

**public void actionPerformed(ActionEvent e) {**

**Object source = e.getSource();**

**// списки вершин**

**if (source instanceof JComboBox) {**

**JComboBox cb = (JComboBox) e.getSource();**

**String cmd = cb.getName();**

**switch (cmd) {**

**case "StartVertexComboBox":**

**endVertexComboBox.removeAllItems();**

**int index = 0;**

**for (int i = 0; i < graph.vertices.size(); i++)**

**if (startVertexComboBox.getSelectedItem() == graph.vertices.get(i).letter) {**

**index = i;**

**break;**

**}**

**for(int i = 0; i < graph.numberOfVertices; ++i)**

**if (graph.linksMatrix[index][i])**

**endVertexComboBox.addItem(graph.vertices.get(i).letter);**

**setAlgorithm(); // алгоритм**

**break;**

**case "EndVertexComboBox":**

**setAlgorithm(); // алгоритм**

**break;**

**}**

**}**

**// кнопки**

**if (source instanceof JButton) {**

**JButton btn = (JButton) e.getSource();**

**String cmd = btn.getActionCommand();**

**switch(cmd) {**

**case "NextStepButton":**

**disableComboBoxes();**

**if (algorithm.finished()) {**

**autoStepButton.setEnabled(false);**

**nextStepButton.setEnabled(false);**

**List<Integer> pathOfAnt = algorithm.findPath();**

**showPathOfAnt(pathOfAnt);**

**} else {**

**List<Integer> pathOfAnt = algorithm.step();**

**showAnt(pathOfAnt);**

**}**

**graphPanel.repaint();**

**break;**

**case "AutoStepButton":**

**disableComboBoxes();**

**autoStepButton.setEnabled(false);**

**nextStepButton.setEnabled(false);**

**List<Integer> pathOfAnt = algorithm.autoAlgorithm();**

**showPathOfAnt(pathOfAnt);**

**graphPanel.repaint();**

**break;**

**case "ClearButton":**

**enableComboBoxes();**

**pathOfAntLabel.setText("");**

**pathOfAntTextLabel.setText("");**

**autoStepButton.setEnabled(true);**

**nextStepButton.setEnabled(true);**

**setAlgorithm();**

**graphPanel.repaint();**

**break;**

**case "BackButton":**

**setVisible(false);**

**dispose();**

**mainMenuFrame.setVisible(true);**

**break;**

**}**

**}**

**}**

**}**

**InfoFrame.java**

**package** GUI;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.WindowListener;  
**import** java.awt.event.WindowEvent;  
  
*// окно "информация об алгоритме"***public class** InfoFrame **extends** JFrame **implements** ActionListener {  
  
 **private final** MainMenuFrame **mainMenuFrame**; *// ссылка на главное окно  
  
 // конструктор* **public** InfoFrame(MainMenuFrame mainMenuFrame\_) {  
 **super**(**"Информация об алгоритме"**);  
  
 **mainMenuFrame** = mainMenuFrame\_;  
  
 *// окно* setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
 setPreferredSize(**new** Dimension(800, 600));  
 setResizable(**false**);  
 setLayout(**null**);  
  
 *// действие при закрытии окна* addWindowListener(**new** WindowListener() {  
 **public void** windowClosing(WindowEvent event) {  
 event.getWindow().setVisible(**false**);  
 event.getWindow().dispose();  
 **mainMenuFrame**.setVisible(**true**);  
 }  
 **public void** windowActivated(WindowEvent event) { }  
 **public void** windowClosed(WindowEvent event) { }  
 **public void** windowDeactivated(WindowEvent event) { }  
 **public void** windowDeiconified(WindowEvent event) { }  
 **public void** windowIconified(WindowEvent event) { }  
 **public void** windowOpened(WindowEvent event) { }  
 });  
  
 *// иконка* setIconImage(Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/icon.png"**)));  
  
 *// кнопка* JButton backButton = **new** JButton(**"В главное меню"**);  
 backButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 backButton.setBounds(250, 480, 300, 60);  
 backButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 backButton.setActionCommand(**"BackButton"**);  
 backButton.addActionListener(**this**);  
 add(backButton);  
  
 *// картинка с информацией об алгоритме* Image img = Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/info.png"**));  
 ImageIcon infoImage = **new** ImageIcon(img);  
 *// панель с прокруткой* JScrollPane panel = **new** JScrollPane(**new** JLabel(infoImage));  
 panel.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_NEVER***);  
 panel.setBounds(0, 0, 795, 450);  
 panel.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 *// скорость прокрутки* JScrollBar jsp = panel.getVerticalScrollBar();  
 jsp.setUnitIncrement(25);  
 add(panel);  
  
 pack();  
 setLocationRelativeTo(**null**); *// центрирование окна* setVisible(**true**);  
 }  
  
 *// нажатие кнопки* **public void** actionPerformed(ActionEvent e) {  
 String cmd = e.getActionCommand();  
  
 **if**(cmd.equals(**"BackButton"**)) {  
 setVisible(**false**);  
 dispose();  
 **mainMenuFrame**.setVisible(**true**);  
 }  
 }  
}

**MainMenuFrame.java**

**package** GUI;  
  
**import** javax.swing.\*;  
**import** javax.swing.border.TitledBorder;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.WindowEvent;  
**import** java.awt.event.WindowListener;  
  
**import** Graph.Graph;  
**import** Staff.Pair;  
  
*// окно "главное меню"***public class** MainMenuFrame **extends** JFrame **implements** ActionListener {  
  
 **private** Graph **graph** = **new** Graph(); *// граф* **private** Pair<Double, Double> **params** = **new** Pair<> (0.5, *// жадность* 0.5); *// скорость испарения феромонов* **private** Pair<Integer, Integer> **antParams** = **new** Pair<> (50, *// кол-во блиц муравьев* 5000); *// общее кол-во муравьев  
  
 // окно параметров создается один раз (для сохранения параметров при деактивации окна)* **private** ParametersFrame **parametersFrame** = **new** ParametersFrame(**this**, **graph**, **params**, **antParams**);  
  
 **private** JButton **startButton**;  
  
 *// конструктор* **public** MainMenuFrame() {  
 **super**(**"Главное меню"**);  
  
 *// скрыть созданное окно параметров* **parametersFrame**.setVisible(**false**);  
  
 *// окно* setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 setPreferredSize(**new** Dimension(800, 600));  
 setResizable(**false**);  
 setLayout(**null**);  
  
 *// иконка* setIconImage(Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/icon.png"**)));  
  
 *// картинка для заголовка* JPanel titlePanel = **new** JPanel();  
 titlePanel.setLayout(**null**);  
 TitledBorder title = BorderFactory.*createTitledBorder*(**""**);  
 titlePanel.setBorder(title);  
 titlePanel.setBounds(20, 20, 760, 206);  
  
 Image img = Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/title.png"**));  
 JLabel picLabel = **new** JLabel(**new** ImageIcon(img));  
 picLabel.setBounds(0, 0, 760, 200);  
 titlePanel.add(picLabel);  
  
 add(titlePanel); *// добавление заголовка на окно  
  
 // панель для кнопок* JPanel buttonPanel = **new** JPanel();  
 buttonPanel.setLayout(**null**);  
 buttonPanel.setBorder(title);  
 buttonPanel.setBounds(230, 240, 340, 315);  
  
 *// кнопки* **startButton** = **new** JButton(**"Запуск алгоритма"**);  
 **startButton**.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 **startButton**.setBounds(20, 20, 300, 60);  
 **startButton**.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 **startButton**.setActionCommand(**"StartButton"**);  
 **startButton**.addActionListener(**this**);  
 buttonPanel.add(**startButton**);  
  
 JButton settingsButton = **new** JButton(**"Параметры"**);  
 settingsButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 settingsButton.setBounds(20, 90, 300, 60);  
 settingsButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 settingsButton.setActionCommand(**"ParametersButton"**);  
 settingsButton.addActionListener(**this**);  
 buttonPanel.add(settingsButton);  
  
 JButton infoButton = **new** JButton(**"Информация об алгоритме"**);  
 infoButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 infoButton.setBounds(20, 160, 300, 60);  
 infoButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 infoButton.setActionCommand(**"InfoButton"**);  
 infoButton.addActionListener(**this**);  
 buttonPanel.add(infoButton);  
  
 JButton exitButton = **new** JButton(**"Выход"**);  
 exitButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 exitButton.setBounds(20, 230, 300, 60);  
 exitButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 exitButton.setActionCommand(**"ExitButton"**);  
 exitButton.addActionListener(**this**);  
 buttonPanel.add(exitButton);  
  
 add(buttonPanel); *// добавление панели на окно  
  
 // при открытии окна* addWindowListener(**new** WindowListener() {  
 **public void** windowOpened(WindowEvent event) { }  
 **public void** windowClosing(WindowEvent event) { }  
 **public void** windowActivated(WindowEvent event) {  
 **if** (!**graph**.isCreated()) {  
 **startButton**.setText(**"Запуск алгоритма (нет графа)"**);  
 **startButton**.setEnabled(**false**);  
 } **else** {  
 **startButton**.setText(**"Запуск алгоритма"**);  
 **startButton**.setEnabled(**true**);  
 }  
 }  
 **public void** windowClosed(WindowEvent event) { }  
 **public void** windowDeactivated(WindowEvent event) { }  
 **public void** windowDeiconified(WindowEvent event) { }  
 **public void** windowIconified(WindowEvent event) { }  
 });  
  
 pack();  
 setLocationRelativeTo(**null**); *// центрирование окна* setVisible(**true**);  
 }  
  
 *// нажатие кнопок* **public void** actionPerformed(ActionEvent e) {  
 String cmd = e.getActionCommand();  
  
 **switch**(cmd) {  
 **case "StartButton"**:  
 **new** AlgorithmFrame(**this**, **graph**, **params**, **antParams**);  
 setVisible(**false**);  
 **break**;  
  
 **case "ParametersButton"**:  
 **parametersFrame**.setVisible(**true**);  
 setVisible(**false**);  
 **break**;  
  
 **case "InfoButton"**:  
 **new** InfoFrame(**this**);  
 setVisible(**false**);  
 **break**;  
  
 **case "ExitButton"**:  
 dispatchEvent(**new** WindowEvent(**this**, WindowEvent.***WINDOW\_CLOSING***));  
 **break**;  
  
 **default**:  
 **break**;  
 }  
 }  
  
 *// main* **public static void** main(String[] args) {  
  
 SwingUtilities.*invokeLater*(() -> {  
 *// изменить стиль окна* **try** {  
 UIManager.*setLookAndFeel*(**"com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"**);  
 } **catch** (Exception e1) {  
 JOptionPane.*showMessageDialog*(**null**,  
 **"Error while changing Look and Feel!"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 *// показать главное окно* **new** MainMenuFrame();  
 });  
 }  
}

**ParametersFrame.java**

**package GUI;**

**import java.io.\*;**

**import java.util.\*;**

**import java.text.NumberFormat;**

**import java.util.Hashtable;**

**import javax.swing.\*;**

**import javax.swing.border.\*;**

**import javax.swing.event.ChangeEvent;**

**import javax.swing.event.ChangeListener;**

**import javax.swing.filechooser.FileNameExtensionFilter;**

**import javax.swing.text.NumberFormatter;**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import java.beans.PropertyChangeEvent;**

**import java.beans.PropertyChangeListener;**

**import Graph.Graph;**

**import Staff.Pair;**

**import GUI.GraphEnter.GraphEnterFrame;**

**// окно "параметры"**

**public class ParametersFrame extends JFrame**

**implements ActionListener, ChangeListener, PropertyChangeListener {**

**private final MainMenuFrame mainMenuFrame; // ссылка на главное окно**

**private Graph graph; // ссылка на граф**

**private Pair<Double, Double> params; // ссылка на параметры (жадность, скорость испарения)**

**private Pair<Integer, Integer> antParams; // ссылка на параметры муравьев**

**// параметры генерации**

**private int numberOfVertices = 5; // количестов вершин**

**private int percentOfEdges = 50; // процент ребер**

**private Pair<Integer, Integer> rangeOfWeights**

**= new Pair<>(1,100); // диапозон весов**

**// компоненты окна**

**// группа "из файла"**

**private JFileChooser fileChooser;**

**private File file;**

**private JTextField filePathTextField;**

**private JButton chooseFileButton;**

**private JButton createGraphFromFileButton;**

**// группа "генерация"**

**private JLabel numberOfVerticesLabel;**

**private JSlider numberOfVerticesSlider;**

**private JLabel percentOfEdgesLabel;**

**private JSlider percentOfEdgesSlider;**

**private JLabel rangeOfWeightsLabel;**

**private JLabel rangeFromLabel;**

**private JLabel rangeToLabel;**

**private JFormattedTextField rangeFromField;**

**private JFormattedTextField rangeToField;**

**private JButton createGraphGenerateButton;**

**// группа "ручной ввод"**

**private JButton handEnterGraphButton;**

**// кнопка "показать граф"**

**private JButton showGraphButton;**

**// конструктор**

**public ParametersFrame(MainMenuFrame mainMenuFrame\_, Graph graph\_,**

**Pair<Double, Double> params\_, Pair<Integer, Integer> antParams\_) {**

**super("Параметры");**

**mainMenuFrame = mainMenuFrame\_;**

**graph = graph\_;**

**params = params\_;**

**antParams = antParams\_;**

**// окно**

**setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);**

**setPreferredSize(new Dimension(800, 600));**

**setResizable(false);**

**setLayout(null);**

**// действие при закрытии и открытии окна**

**addWindowListener(new WindowListener() {**

**public void windowClosing(WindowEvent event) {**

**event.getWindow().setVisible(false);**

**event.getWindow().dispose();**

**mainMenuFrame.setVisible(true);**

**}**

**public void windowActivated(WindowEvent event) {**

**if (graph.isCreated()) {**

**showGraphButton.setText("Показать граф");**

**showGraphButton.setEnabled(true);**

**} else {**

**showGraphButton.setText("Показать граф (нет графа)");**

**showGraphButton.setEnabled(false);**

**}**

**}**

**public void windowClosed(WindowEvent event) { }**

**public void windowDeactivated(WindowEvent event) { }**

**public void windowDeiconified(WindowEvent event) { }**

**public void windowIconified(WindowEvent event) { }**

**public void windowOpened(WindowEvent event) { }**

**});**

**// иконка**

**setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("/GUI/images/icon.png")));**

**// панель "ввод графа"**

**JPanel enterGraphPanel = new JPanel();**

**enterGraphPanel.setLayout(null);**

**enterGraphPanel.setBounds(20, 5, 760, 270);**

**TitledBorder enterGraphPanelTitle = BorderFactory.createTitledBorder("Ввод графа:");**

**enterGraphPanel.setBorder(enterGraphPanelTitle);**

**// радио кнопки выбора ввода**

**JRadioButton fromFileRadioButton = new JRadioButton("Из файла:");**

**fromFileRadioButton.setFont(new Font("Arial", Font.PLAIN, 14));**

**fromFileRadioButton.setBounds(20, 40, 90, 20);**

**fromFileRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**fromFileRadioButton.setActionCommand("FromFileRadioButton");**

**fromFileRadioButton.addActionListener(this);**

**fromFileRadioButton.setSelected(true);**

**JRadioButton generateGraphRadioButton = new JRadioButton("Генерация:");**

**generateGraphRadioButton.setFont(new Font("Arial", Font.PLAIN, 14));**

**generateGraphRadioButton.setBounds(20, 80, 100, 20);**

**generateGraphRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**generateGraphRadioButton.setActionCommand("GenerateRadioButton");**

**generateGraphRadioButton.addActionListener(this);**

**JRadioButton handEnterGraphRadioButton = new JRadioButton("Ручной ввод:");**

**handEnterGraphRadioButton.setFont(new Font("Arial", Font.PLAIN, 14));**

**handEnterGraphRadioButton.setBounds(20, 220, 110, 20);**

**handEnterGraphRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**handEnterGraphRadioButton.setActionCommand("HandEnterRadioButton");**

**handEnterGraphRadioButton.addActionListener(this);**

**// группа кнопок выбора**

**ButtonGroup enterGraphGroup = new ButtonGroup();**

**enterGraphGroup.add(fromFileRadioButton);**

**enterGraphGroup.add(generateGraphRadioButton);**

**enterGraphGroup.add(handEnterGraphRadioButton);**

**// добавить на панель**

**enterGraphPanel.add(fromFileRadioButton);**

**enterGraphPanel.add(generateGraphRadioButton);**

**enterGraphPanel.add(handEnterGraphRadioButton);**

**// ввод графа из файла**

**fileChooser = new JFileChooser();**

**fileChooser.setFileFilter(new FileNameExtensionFilter("Text files", "txt"));**

**chooseFileButton = new JButton("Выбрать файл");**

**chooseFileButton.setFont(new Font("Arial", Font.BOLD, 14));**

**chooseFileButton.setBounds(140, 30, 150, 40);**

**chooseFileButton.setActionCommand("ChooseFileButton");**

**chooseFileButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**chooseFileButton.addActionListener(this);**

**enterGraphPanel.add(chooseFileButton);**

**filePathTextField = new JTextField("Путь к файлу . . .");**

**filePathTextField.setBounds(310, 35, 260, 30);**

**filePathTextField.setEditable(false);**

**filePathTextField.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT\_CURSOR));**

**enterGraphPanel.add(filePathTextField);**

**// кнопка "создать граф из файла"**

**createGraphFromFileButton = new JButton("Создать граф");**

**createGraphFromFileButton.setFont(new Font("Arial", Font.BOLD, 14));**

**createGraphFromFileButton.setBounds(590, 30, 150, 40);**

**createGraphFromFileButton.setActionCommand("CreateGraphFromFileButton");**

**createGraphFromFileButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**createGraphFromFileButton.addActionListener(this);**

**createGraphFromFileButton.setEnabled(false);**

**enterGraphPanel.add(createGraphFromFileButton);**

**// ввод графа - генерация**

**numberOfVerticesLabel = new JLabel("Количество вершин:");**

**numberOfVerticesLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**numberOfVerticesLabel.setBounds(140, 70, 140, 40);**

**enterGraphPanel.add(numberOfVerticesLabel);**

**// слайдер - количество вершин**

**numberOfVerticesSlider = new JSlider(JSlider.HORIZONTAL, 2, 10, numberOfVertices);**

**numberOfVerticesSlider.setBounds(310, 80, 425, 40);**

**numberOfVerticesSlider.setMajorTickSpacing(1);**

**numberOfVerticesSlider.setMinorTickSpacing(1);**

**numberOfVerticesSlider.setPaintTicks(true);**

**numberOfVerticesSlider.setPaintLabels(true);**

**numberOfVerticesSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**numberOfVerticesSlider.setBorder(BorderFactory.createEmptyBorder(0,0,15,0));**

**numberOfVerticesSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**numberOfVerticesSlider.setName("NumberOfVerticesSlider");**

**numberOfVerticesSlider.addChangeListener(this);**

**enterGraphPanel.add(numberOfVerticesSlider);**

**percentOfEdgesLabel = new JLabel("Процент ребер:");**

**percentOfEdgesLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**percentOfEdgesLabel.setBounds(140, 115, 120, 40);**

**enterGraphPanel.add(percentOfEdgesLabel);**

**// слайдер - процент ребер**

**percentOfEdgesSlider = new JSlider(JSlider.HORIZONTAL, 20, 100, percentOfEdges);**

**percentOfEdgesSlider.setBounds(310, 125, 425, 40);**

**percentOfEdgesSlider.setMajorTickSpacing(10);**

**percentOfEdgesSlider.setMinorTickSpacing(5);**

**percentOfEdgesSlider.setPaintTicks(true);**

**percentOfEdgesSlider.setPaintLabels(true);**

**percentOfEdgesSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**percentOfEdgesSlider.setBorder(BorderFactory.createEmptyBorder(0,0,15,0));**

**percentOfEdgesSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**percentOfEdgesSlider.setName("PercentOfEdgesSlider");**

**percentOfEdgesSlider.addChangeListener(this);**

**enterGraphPanel.add(percentOfEdgesSlider);**

**// ----- форматтер ввода ------**

**NumberFormat nf = NumberFormat.getIntegerInstance();**

**NumberFormatter nff= new NumberFormatter(nf);**

**nff.setMinimum(1);**

**nff.setMaximum(100);**

**// ----------------------------**

**// поля - диапозон весов**

**rangeOfWeightsLabel = new JLabel("Диапозон весов:");**

**rangeOfWeightsLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**rangeOfWeightsLabel.setBounds(140, 170, 120, 40);**

**enterGraphPanel.add(rangeOfWeightsLabel);**

**// от**

**rangeFromLabel = new JLabel("От:");**

**rangeFromLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**rangeFromLabel.setBounds(310, 170, 30, 40);**

**enterGraphPanel.add(rangeFromLabel);**

**rangeFromField = new JFormattedTextField(nff);**

**rangeFromField.setValue(rangeOfWeights.first);**

**rangeFromField.setBounds(340, 175, 50, 30);**

**rangeFromField.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT\_CURSOR));**

**rangeFromField.setName("RangeFromField");**

**rangeFromField.addPropertyChangeListener(this);**

**enterGraphPanel.add(rangeFromField);**

**rangeFromField.addFocusListener(new FocusListener() {**

**public void focusGained(FocusEvent fe) { }**

**public void focusLost(FocusEvent fe) {**

**try {**

**if (Integer.parseInt(rangeFromField.getText()) > Integer.parseInt(rangeToField.getText()))**

**rangeToField.setValue(100);**

**} catch (Exception e1) {}**

**}**

**});**

**// до**

**rangeToLabel = new JLabel("До:");**

**rangeToLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**rangeToLabel.setBounds(420, 170, 30, 40);**

**enterGraphPanel.add(rangeToLabel);**

**rangeToField = new JFormattedTextField(nff);**

**rangeToField.setValue(rangeOfWeights.second);**

**rangeToField.setBounds(450, 175, 50, 30);**

**rangeToField.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT\_CURSOR));**

**rangeToField.setName("RangeToField");**

**rangeToField.addPropertyChangeListener(this);**

**enterGraphPanel.add(rangeToField);**

**rangeToField.addFocusListener(new FocusListener() {**

**public void focusGained(FocusEvent fe) { }**

**public void focusLost(FocusEvent fe) {**

**try {**

**if (Integer.parseInt(rangeFromField.getText()) > Integer.parseInt(rangeToField.getText()))**

**rangeFromField.setValue(1);**

**} catch (Exception e1) {}**

**}**

**});**

**// кнопка "создать граф - генерация"**

**createGraphGenerateButton = new JButton("Создать граф");**

**createGraphGenerateButton.setFont(new Font("Arial", Font.BOLD, 14));**

**createGraphGenerateButton.setBounds(590, 170, 150, 40);**

**createGraphGenerateButton.setActionCommand("CreateGraphGenerateButton");**

**createGraphGenerateButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**createGraphGenerateButton.addActionListener(this);**

**enterGraphPanel.add(createGraphGenerateButton);**

**// ввод графа - ручной ввод**

**handEnterGraphButton = new JButton("Ввести граф");**

**handEnterGraphButton.setFont(new Font("Arial", Font.BOLD, 14));**

**handEnterGraphButton.setBounds(140, 210, 150, 40);**

**handEnterGraphButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**handEnterGraphButton.setActionCommand("EnterGraphButton");**

**handEnterGraphButton.addActionListener(this);**

**enterGraphPanel.add(handEnterGraphButton);**

**// панель "дополнительные параметры"**

**JPanel paramsPanel = new JPanel();**

**paramsPanel.setLayout(null);**

**paramsPanel.setBounds(20, 280, 760, 230);**

**TitledBorder paramsPanelTitle = BorderFactory.createTitledBorder("Дополнительные параметры:");**

**paramsPanel.setBorder(paramsPanelTitle);**

**// --- для шкалы слайдеров ---**

**Hashtable labelTable = new Hashtable();**

**labelTable.put(0, new JLabel("0.0"));**

**labelTable.put(10, new JLabel("0.1"));**

**labelTable.put(20, new JLabel("0.2"));**

**labelTable.put(30, new JLabel("0.3"));**

**labelTable.put(40, new JLabel("0.4"));**

**labelTable.put(50, new JLabel("0.5"));**

**labelTable.put(60, new JLabel("0.6"));**

**labelTable.put(70, new JLabel("0.7"));**

**labelTable.put(80, new JLabel("0.8"));**

**labelTable.put(90, new JLabel("0.9"));**

**labelTable.put(100, new JLabel("1.0"));**

**// ----------------------------**

**JLabel greedOfAlgorithmLabel = new JLabel("\"Жадность\" алгоритма:");**

**greedOfAlgorithmLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**greedOfAlgorithmLabel.setBounds(20, 20, 160, 40);**

**paramsPanel.add(greedOfAlgorithmLabel);**

**// слайдер - жадность алгоритма**

**Double tmp = params.first \* 100;**

**Integer greedInt = tmp.intValue();**

**JSlider greedOfAlgorithmSlider = new JSlider(JSlider.HORIZONTAL, 0, 100, greedInt);**

**greedOfAlgorithmSlider.setBounds(310, 30, 425, 40);**

**greedOfAlgorithmSlider.setLabelTable(labelTable);**

**greedOfAlgorithmSlider.setMajorTickSpacing(10);**

**greedOfAlgorithmSlider.setMinorTickSpacing(5);**

**greedOfAlgorithmSlider.setPaintTicks(true);**

**greedOfAlgorithmSlider.setPaintLabels(true);**

**greedOfAlgorithmSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**greedOfAlgorithmSlider.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));**

**greedOfAlgorithmSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**greedOfAlgorithmSlider.setName("GreedOfAlgorithmSlider");**

**greedOfAlgorithmSlider.addChangeListener(this);**

**paramsPanel.add(greedOfAlgorithmSlider);**

**JLabel rateOfEvaporationLabel = new JLabel("Скорость испарения феромона:");**

**rateOfEvaporationLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**rateOfEvaporationLabel.setBounds(20, 65, 220, 40);**

**paramsPanel.add(rateOfEvaporationLabel);**

**// слайдер - скорость испарения феромона**

**Double tmp1 = params.second \* 100;**

**Integer evaporationSpeedInt = tmp1.intValue();**

**JSlider rateOfEvaporationSlider = new JSlider(JSlider.HORIZONTAL, 0, 100, evaporationSpeedInt);**

**rateOfEvaporationSlider.setBounds(310, 75, 425, 40);**

**rateOfEvaporationSlider.setLabelTable(labelTable);**

**rateOfEvaporationSlider.setMajorTickSpacing(10);**

**rateOfEvaporationSlider.setMinorTickSpacing(5);**

**rateOfEvaporationSlider.setPaintTicks(true);**

**rateOfEvaporationSlider.setPaintLabels(true);**

**rateOfEvaporationSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**rateOfEvaporationSlider.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));**

**rateOfEvaporationSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**rateOfEvaporationSlider.setName("RateOfEvaporationSlider");**

**rateOfEvaporationSlider.addChangeListener(this);**

**paramsPanel.add(rateOfEvaporationSlider);**

**JLabel numberOfRandomAntsLabel = new JLabel("Количество \"Блиц\" муравьев: ");**

**numberOfRandomAntsLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**numberOfRandomAntsLabel.setBounds(20, 115, 210, 40);**

**paramsPanel.add(numberOfRandomAntsLabel);**

**// слайдер - количество блиц муравьев**

**JSlider numberOfRandomAntsSlider = new JSlider(JSlider.HORIZONTAL, 1, 101, antParams.first+1);**

**numberOfRandomAntsSlider.setBounds(310, 120, 425, 40);**

**Hashtable<Integer, JLabel> table = new Hashtable<Integer, JLabel>();**

**table.put (1, new JLabel("1"));**

**table.put (11, new JLabel("10"));**

**table.put (21, new JLabel("20"));**

**table.put (31, new JLabel("30"));**

**table.put (41, new JLabel("40"));**

**table.put (51, new JLabel("50"));**

**table.put (61, new JLabel("60"));**

**table.put (71, new JLabel("70"));**

**table.put (81, new JLabel("80"));**

**table.put (91, new JLabel("90"));**

**table.put (101, new JLabel("100"));**

**numberOfRandomAntsSlider.setLabelTable(table);**

**numberOfRandomAntsSlider.setMajorTickSpacing(10);**

**numberOfRandomAntsSlider.setMinorTickSpacing(5);**

**numberOfRandomAntsSlider.setPaintTicks(true);**

**numberOfRandomAntsSlider.setPaintLabels(true);**

**numberOfRandomAntsSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**numberOfRandomAntsSlider.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));**

**numberOfRandomAntsSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**numberOfRandomAntsSlider.setName("NumberOfRandomAntsSlider");**

**numberOfRandomAntsSlider.addChangeListener(this);**

**paramsPanel.add(numberOfRandomAntsSlider);**

**JLabel numberOfAntsLabel = new JLabel("Общее количество муравьев: ");**

**numberOfAntsLabel.setFont(new Font("Arial", Font.PLAIN, 14));**

**numberOfAntsLabel.setBounds(20, 165, 210, 40);**

**paramsPanel.add(numberOfAntsLabel);**

**// слайдер - общее количество муравьев**

**JSlider numberOfAntsSlider = new JSlider(JSlider.HORIZONTAL, 10, 10010, antParams.second+10);**

**numberOfAntsSlider.setBounds(305, 165, 435, 40);**

**Hashtable<Integer, JLabel> table1 = new Hashtable<Integer, JLabel>();**

**table1.put (10, new JLabel("10"));**

**table1.put (1010, new JLabel("1000"));**

**table1.put (2010, new JLabel("2000"));**

**table1.put (3010, new JLabel("3000"));**

**table1.put (4010, new JLabel("4000"));**

**table1.put (5010, new JLabel("5000"));**

**table1.put (6010, new JLabel("6000"));**

**table1.put (7010, new JLabel("7000"));**

**table1.put (8010, new JLabel("8000"));**

**table1.put (9010, new JLabel("9000"));**

**table1.put (10010, new JLabel("10000"));**

**numberOfAntsSlider.setLabelTable(table1);**

**numberOfAntsSlider.setMajorTickSpacing(1000);**

**numberOfAntsSlider.setMinorTickSpacing(500);**

**numberOfAntsSlider.setPaintTicks(true);**

**numberOfAntsSlider.setPaintLabels(true);**

**numberOfAntsSlider.setFont(new Font("Arial", Font.PLAIN, 10));**

**numberOfAntsSlider.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));**

**numberOfAntsSlider.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**numberOfAntsSlider.setName("NumberOfAntsSlider");**

**numberOfAntsSlider.addChangeListener(this);**

**paramsPanel.add(numberOfAntsSlider);**

**// кнопки**

**JButton backButton = new JButton("В главное меню");**

**backButton.setFont(new Font("Arial", Font.BOLD, 14));**

**backButton.setBounds(100, 520, 230, 40);**

**backButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**backButton.setActionCommand("BackButton");**

**backButton.addActionListener(this);**

**add(backButton);**

**showGraphButton = new JButton("Показать граф");**

**showGraphButton.setFont(new Font("Arial", Font.BOLD, 14));**

**showGraphButton.setBounds(460, 520, 230, 40);**

**if (!graph.isCreated()) {**

**showGraphButton.setText("Показать граф (нет графа)");**

**showGraphButton.setEnabled(false);**

**}**

**showGraphButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));**

**showGraphButton.setActionCommand("ShowGraphButton");**

**showGraphButton.addActionListener(this);**

**add(showGraphButton);**

**add(enterGraphPanel);**

**add(paramsPanel);**

**pack();**

**setLocationRelativeTo(null); // центрирование окна**

**setVisible(true);**

**// по-умолчанию - выбор из файла**

**enableEnterFromFileGraph();**

**disableGenerationGraph();**

**disableHandEnterGraph();**

**}**

**// заблокировать компоненты (радио кнопки)**

**private void disableEnterFromFileGraph() {**

**chooseFileButton.setEnabled(false);**

**filePathTextField.setEnabled(false);**

**createGraphFromFileButton.setEnabled(false);**

**}**

**private void disableGenerationGraph() {**

**numberOfVerticesLabel.setEnabled(false);**

**numberOfVerticesSlider.setEnabled(false);**

**percentOfEdgesLabel.setEnabled(false);**

**percentOfEdgesSlider.setEnabled(false);**

**rangeOfWeightsLabel.setEnabled(false);**

**rangeFromLabel.setEnabled(false);**

**rangeToLabel.setEnabled(false);**

**rangeFromField.setEnabled(false);**

**rangeToField.setEnabled(false);**

**createGraphGenerateButton.setEnabled(false);**

**}**

**private void disableHandEnterGraph() {**

**handEnterGraphButton.setEnabled(false);**

**}**

**// разблокировать компоненты (радио кнопки)**

**private void enableEnterFromFileGraph() {**

**chooseFileButton.setEnabled(true);**

**filePathTextField.setEnabled(true);**

**if (file != null) createGraphFromFileButton.setEnabled(true);**

**}**

**private void enableGenerationGraph() {**

**numberOfVerticesLabel.setEnabled(true);**

**numberOfVerticesSlider.setEnabled(true);**

**percentOfEdgesLabel.setEnabled(true);**

**percentOfEdgesSlider.setEnabled(true);**

**rangeOfWeightsLabel.setEnabled(true);**

**rangeFromLabel.setEnabled(true);**

**rangeToLabel.setEnabled(true);**

**rangeFromField.setEnabled(true);**

**rangeToField.setEnabled(true);**

**createGraphGenerateButton.setEnabled(true);**

**}**

**private void enableHandEnterGraph() {**

**handEnterGraphButton.setEnabled(true);**

**}**

**// слайдеры**

**public void stateChanged(ChangeEvent e) {**

**Object source = e.getSource();**

**JSlider slider = (JSlider)source;**

**switch (slider.getName()) {**

**case"NumberOfVerticesSlider":**

**numberOfVertices = slider.getValue();**

**break;**

**case "PercentOfEdgesSlider":**

**percentOfEdges = slider.getValue();**

**break;**

**case "GreedOfAlgorithmSlider":**

**// делить на 100, т. к. целочисленная шкла (1 == 0.1, 100 == 1)**

**params.first = new Integer(slider.getValue()).doubleValue() / 100;**

**break;**

**case "RateOfEvaporationSlider":**

**// делить на 100, т. к. целочисленная шкла (1 == 0.1, 100 == 1)**

**params.second = new Integer(slider.getValue()).doubleValue() / 100;**

**break;**

**case "NumberOfRandomAntsSlider":**

**if (slider.getValue() == 1)**

**antParams.first = slider.getValue();**

**else**

**antParams.first = slider.getValue()-1;**

**break;**

**case "NumberOfAntsSlider":**

**antParams.second = slider.getValue();**

**break;**

**default:**

**break;**

**}**

**}**

**// поля "от" и "до"**

**public void propertyChange(PropertyChangeEvent e) {**

**Object source = e.getSource();**

**JFormattedTextField field = (JFormattedTextField)source;**

**switch (field.getName()) {**

**case "RangeFromField":**

**try {**

**rangeOfWeights.first = Integer.parseInt(rangeFromField.getText());**

**} catch (Exception e1) { }**

**break;**

**case "RangeToField":**

**try {**

**rangeOfWeights.second = Integer.parseInt(rangeToField.getText());**

**} catch (Exception e1) { }**

**break;**

**}**

**}**

**// нажатие кнопок**

**public void actionPerformed(ActionEvent e) {**

**String cmd = e.getActionCommand();**

**switch(cmd) {**

**// радио кнопки**

**case "FromFileRadioButton":**

**enableEnterFromFileGraph();**

**disableGenerationGraph();**

**disableHandEnterGraph();**

**break;**

**case "GenerateRadioButton":**

**enableGenerationGraph();**

**disableEnterFromFileGraph();**

**disableHandEnterGraph();**

**break;**

**case "HandEnterRadioButton":**

**enableHandEnterGraph();**

**disableGenerationGraph();**

**disableEnterFromFileGraph();**

**break;**

**// кнопки**

**case "ChooseFileButton":**

**int isFileOpen = fileChooser.showOpenDialog(this);**

**if (isFileOpen == JFileChooser.APPROVE\_OPTION) {**

**file = fileChooser.getSelectedFile();**

**filePathTextField.setText(file.getPath());**

**createGraphFromFileButton.setEnabled(true);**

**}**

**break;**

**case "CreateGraphFromFileButton":**

**try {**

**if (graph.createGraphFromFile(new Scanner(file))) {**

**showGraphButton.setText("Показать граф");**

**showGraphButton.setEnabled(true);**

**if (graph.isNull())**

**JOptionPane.showMessageDialog(null,**

**"Граф успешно создан. Ребер нет!", "Внимание", JOptionPane.WARNING\_MESSAGE);**

**else**

**JOptionPane.showMessageDialog(null,**

**"Граф успешно создан!", "Граф создан", JOptionPane.PLAIN\_MESSAGE);**

**} else {**

**JOptionPane.showMessageDialog(null,**

**"Невозможно считать граф из файла!", "Ошибка", JOptionPane.ERROR\_MESSAGE);**

**}**

**} catch (FileNotFoundException e1) {**

**e1.printStackTrace();**

**}**

**break;**

**case "CreateGraphGenerateButton":**

**rangeOfWeights.first = Integer.parseInt(rangeFromField.getText());**

**rangeOfWeights.second = Integer.parseInt(rangeToField.getText());**

**graph.generateGraph(numberOfVertices, percentOfEdges,**

**rangeOfWeights.first, rangeOfWeights.second);**

**showGraphButton.setText("Показать граф");**

**showGraphButton.setEnabled(true);**

**if (graph.isNull())**

**JOptionPane.showMessageDialog(null,**

**"Граф успешно создан. Ребер нет!", "Внимание", JOptionPane.WARNING\_MESSAGE);**

**else**

**JOptionPane.showMessageDialog(null,**

**"Граф успешно создан!", "Граф создан", JOptionPane.PLAIN\_MESSAGE);**

**break;**

**case "EnterGraphButton":**

**graph.deleteGraph();**

**new GraphEnterFrame(this, graph);**

**setVisible(false);**

**break;**

**case "ShowGraphButton":**

**new ShowGraphFrame(this, graph);**

**setVisible(false);**

**break;**

**case "BackButton":**

**setVisible(false);**

**dispose();**

**mainMenuFrame.setVisible(true);**

**break;**

**default:**

**break;**

**}**

**}**

**}**

**ShowGraphFrame.java**

**package** GUI;  
  
**import** java.util.List;  
**import** javax.swing.\*;  
**import** javax.swing.border.TitledBorder;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.WindowListener;  
**import** java.awt.event.WindowEvent;  
  
**import** Algorithm.AntAlgorithm;  
**import** Graph.Graph;  
**import** Staff.Pair;  
  
*// окно "алгоритм"***public class** ShowGraphFrame **extends** JFrame **implements** ActionListener {  
  
 **private final** ParametersFrame **parametersFrame**; *// ссылка на окно-родитель* **private** Graph **graph**; *// ссылка на граф  
  
 // панель с графом* **class** GraphPanel **extends** JPanel {  
  
 **public** GraphPanel() {  
 setBounds(300, 10, 685, 660);  
 setLayout(**null**);  
 TitledBorder graphPanelTitle = BorderFactory.*createTitledBorder*(**""**);  
 setBorder(graphPanelTitle);  
 }  
  
 **final** BasicStroke **stroke** = **new** BasicStroke(2.0f);  
 **final** BasicStroke **wideStroke** = **new** BasicStroke(8.0f);  
  
 **public void** paintComponent(Graphics g) {  
 **super**.paintComponent(g);  
  
 Graphics2D g2 = (Graphics2D)g;  
  
 g2.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);  
 g2.setStroke( **stroke** );  
  
  
 *// вершины* **for** (**int** i = 0; i < **graph**.**numberOfVertices**; i++) {  
 **int** x = **graph**.**vertices**.get(i).**coordX**;  
 **int** y = **graph**.**vertices**.get(i).**coordY**;  
 String name = String.*valueOf*(**graph**.**vertices**.get(i).**letter**);  
  
 g.drawOval(x, y, 50, 50);  
 g.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 20));  
 g.drawString(name, x + 20, y + 30);  
 }  
  
 *// ребра* **for** (**int** i = 0; i < **graph**.**edges**.size(); i++) {  
 **int** x1 = **graph**.**vertices**.get(**graph**.**edges**.get(i).**firstNode**).**coordX** + 25;  
 **int** y1 = **graph**.**vertices**.get(**graph**.**edges**.get(i).**firstNode**).**coordY** + 25;  
 **int** x2 = **graph**.**vertices**.get(**graph**.**edges**.get(i).**secondNode**).**coordX** + 25;  
 **int** y2 = **graph**.**vertices**.get(**graph**.**edges**.get(i).**secondNode**).**coordY** + 25;  
  
 **int** x11= (**int**)(x1 + 25 \* (x2 - x1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** y11 = (**int**)(y1 + 25 \* (y2 - y1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** x22 = (**int**)(x2 + 25 \* (x1 - x2) /  
 Math.*sqrt*(Math.*pow*((**double**)(x1 - x2), 2.0)  
 + Math.*pow*((**double**)(y1 - y2), 2.0)));  
  
 **int** y22 = (**int**)(y2 + 25 \* (y1 - y2) /  
 Math.*sqrt*(Math.*pow*((**double**)(x1 - x2), 2.0)  
 + Math.*pow*((**double**)(y1 - y2), 2.0)));  
  
  
 g2.drawLine(x11, y11, x22, y22);  
  
 **int** x= (**int**)(x1 + 70 \* (x2 - x1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 **int** y = (**int**)(y1 + 70 \* (y2 - y1) /  
 Math.*sqrt*(Math.*pow*((**double**)(x2 - x1), 2.0)  
 + Math.*pow*((**double**)(y2 - y1), 2.0)));  
  
 *// вес* g2.setColor(Color.***RED***);  
 g2.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 15));  
 g2.drawString(String.*valueOf*(**graph**.**edges**.get(i).**weight**), x - 8, y + 8);  
 g2.setColor(Color.***BLACK***);  
 }  
 }  
 }  
  
 *// конструктор* **public** ShowGraphFrame(ParametersFrame parametersFrame\_, Graph graph\_) {  
 **super**(**"Алгоритм"**);  
  
 **parametersFrame** = parametersFrame\_;  
 **graph** = graph\_;  
  
 *// окно* setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
 setPreferredSize(**new** Dimension(1000, 700));  
 setResizable(**false**);  
 setLayout(**null**);  
  
 *// действие при закрытии окна* addWindowListener(**new** WindowListener() {  
 **public void** windowClosing(WindowEvent event) {  
 event.getWindow().setVisible(**false**);  
 event.getWindow().dispose();  
 **parametersFrame**.setVisible(**true**);  
 }  
  
 **public void** windowActivated(WindowEvent event) {  
 }  
  
 **public void** windowClosed(WindowEvent event) {  
 }  
  
 **public void** windowDeactivated(WindowEvent event) {  
 }  
  
 **public void** windowDeiconified(WindowEvent event) {  
 }  
  
 **public void** windowIconified(WindowEvent event) {  
 }  
  
 **public void** windowOpened(WindowEvent event) { }  
 });  
  
 *// иконка* setIconImage(Toolkit.*getDefaultToolkit*().getImage(getClass().getResource(**"/GUI/images/icon.png"**)));  
  
 *// боковая панель* JPanel sidePanel = **new** JPanel();  
 sidePanel.setBounds(10, 10, 280, 660);  
 sidePanel.setLayout(**null**);  
 TitledBorder sidePanelTitle = BorderFactory.*createTitledBorder*(**""**);  
 sidePanel.setBorder(sidePanelTitle);  
  
  
  
 JButton backButton = **new** JButton(**"В главное меню"**);  
 backButton.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 14));  
 backButton.setBounds(20, 270, 240, 60);  
 backButton.setCursor(Cursor.*getPredefinedCursor*(Cursor.***HAND\_CURSOR***));  
 backButton.setActionCommand(**"BackButton"**);  
 backButton.addActionListener(**this**);  
 sidePanel.add(backButton);  
  
 *// панель для графа* GraphPanel graphPanel = **new** GraphPanel();  
  
 add(sidePanel);  
 add(graphPanel);  
 pack();  
 setLocationRelativeTo(**null**); *// центрирование окна* setVisible(**true**);  
 }  
  
 *// нажатие кнопки* **public void** actionPerformed(ActionEvent e) {  
 Object source = e.getSource();  
  
 *// кнопки* **if** (source **instanceof** JButton) {  
 JButton btn = (JButton) e.getSource();  
 String cmd = btn.getActionCommand();  
  
 **switch**(cmd) {  
  
 **case "BackButton"**:  
 setVisible(**false**);  
 dispose();  
 **parametersFrame**.setVisible(**true**);  
 **break**;  
 }  
 }  
 }  
}

**Тестирование**

Тестирование алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Жадность алгоритма | Скорость испарения  феромона | Количество «блиц» муравьев | Общее количество муравьев | Исходная матрица | Выбранный муравьями путь |
| 0.9 | 0.5 | 30 | 10000 | 0 5 2 10  5 0 0 15  2 0 0 13  10 15 13 0 | A-C-D 100% \* |
| 0.1 | 0.5 | 30 | 10000 | 0 5 2 10  5 0 0 15  2 0 0 13  10 15 13 0 | A-C-D 60% A-D 30% A-B-D  10% |
| 0.1 | 0.1 | 30 | 10000 | 0 5 2 10  5 0 0 15  2 0 0 13  10 15 13 0 | A-D 80%  A-C-D 20% |
| 0.9 | 0.5 | 40 | 10000 | 0 1 0 0 0 0 0  1 0 1 1 0 0 0  0 1 0 1 0 0 0  0 1 1 0 1 1 0  0 0 0 1 0 1 0  0 0 0 1 1 0 1  0 0 0 0 0 1 0 | A-B-D-F-G 35%\*\*  A-B-D-E-F-G 30%  A-B-C-D-F-G 20% A-B-C-D-E-F-G 15% |
| 0.1 | 0.5 | 40 | 10000 | 0 1 0 0 0 0 0  1 0 1 1 0 0 0  0 1 0 1 0 0 0  0 1 1 0 1 1 0  0 0 0 1 0 1 0  0 0 0 1 1 0 1  0 0 0 0 0 1 0 | A-B-D-F-G 35%  A-B-C-D-E-F-G 25%  A-B-D-E-F-G 20%  A-B-C-D-F-G 20% |
| 0.1 | 0.1 | 40 | 10000 | 0 1 0 0 0 0 0  1 0 1 1 0 0 0  0 1 0 1 0 0 0  0 1 1 0 1 1 0  0 0 0 1 0 1 0  0 0 0 1 1 0 1  0 0 0 0 0 1 0 | A-B-D-F-G 90%  A-B-D-E-F-G 5%  A-B-C-D-F-G 5% A-B-C-D-E-F-G 0% |
| 0.9 | 0.5 | 20 | 10000 | 0 10 0 0 20  10 0 10 10 0  0 10 0 10 0  0 10 10 0 10  20 0 0 10 0 | A-B-D-E 45%  A-B-C-D-E  40%  A-E  15% |
| 0.1 | 0.5 | 20 | 10000 | 0 10 0 0 20  10 0 10 10 0  0 10 0 10 0  0 10 10 0 10  20 0 0 10 0 | A-E  60% A-B-D-E 30%  A-B-C-D-E  10% |
| 0.1 | 0.1 | 20 | 10000 | 0 10 0 0 20  10 0 10 10 0  0 10 0 10 0  0 10 10 0 10  20 0 0 10 0 | A-E  100% A-B-D-E 0%  A-B-C-D-E  0% |
| 0.9 | 0.5 | 40 | 10000 | 0 10 0 0 0 0  10 0 40 10 0 0  0 40 0 70 60 20  0 10 70 0 20 0  0 0 60 20 0 30  0 0 20 0 30 0 | A-B-D-E-F-C 90% A-B-D-E-C 10% |
| 0.1 | 0.5 | 40 | 10000 | 0 10 0 0 0 0  10 0 40 10 0 0  0 40 0 70 60 20  0 10 70 0 20 0  0 0 60 20 0 30  0 0 20 0 30 0 | A-B-C  55% A-B-D-C  25%  A-B-D-E-C  20% |
| 0.1 | 0.1 | 40 | 10000 | 0 10 0 0 0 0  10 0 40 10 0 0  0 40 0 70 60 20  0 10 70 0 20 0  0 0 60 20 0 30  0 0 20 0 30 0 | A-B-C  100% |

\* - здесь и далее за 10 запусков

\*\* - здесь и далее за 20 запусков

**Заключение об успешности выполнения проекта**

Проект был выполнен успешно.