**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Объектно-ориентированное программирование»**

Тема: разработка приложений на основе принципов объектно-ориентированного подхода

|  |  |  |
| --- | --- | --- |
| Студент гр. 7302 |  | Карманов Д.А. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Карманов Д.А. | | |
| Группа 7302 | | |
| Тема работы: разработка приложений на основе принципов объектно-ориентированного подхода | | |
| Исходные данные:  Поставленное задание (части 3-4, стр. 6-17, 19-32)  Язык программирования С# (.NET 4.7.2) | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Первый раздел», «Второй раздел», «Третий раздел», «Четвертый раздел», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 30 страниц(ы). | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Карманов Д.А. |
| Преподаватель |  | Новакова Н.Е. |

**АННОТАЦИЯ**

Курсовая работа содержит в себе решения таких поставленных задач, как создание иерархии классов, реализация алгоритма нахождения минимального паросочетания и алгоритма нахождения порядковой функции сети, разработка имитационной модели, написание синтаксического анализатора и нахождение минимума функции с помощью метода оптимизации.

На основе разработанных моделей были разработаны приложения, включающие в себя графический пользовательский интерфейс. Полученные результаты приведены в работе.

**SUMMARY**

Course work contains a solution of such tasks as creating a class hierarchy, the implementation of the algorithm for finding the minimum pair and the algorithm for finding the ordinal function of the network, the development of a simulation model, development of a parser and finding the function minimum using the optimization method.

On the basis of the developed model applications have been developed, including a graphical user interface. The results are presented.

Оглавление

[**ВВЕДЕНИЕ** 5](#_Toc45647419)

[**Цель работы** 5](#_Toc45647420)

[**РАЗДЕЛ ТРЕТИЙ. ОСТОВ НАИМЕНЬШЕГО ВЕСА** 6](#_Toc45647421)

[**Формулировка задания** 6](#_Toc45647422)

[**Теоретический аспект задачи** 6](#_Toc45647423)

[**Формализация задачи** 6](#_Toc45647424)

[**Спецификация программы** 8](#_Toc45647425)

[**Руководство пользователя** 10](#_Toc45647426)

[**Руководство программиста** 11](#_Toc45647427)

[**Контрольные примеры** 11](#_Toc45647428)

[**Листинг** 12](#_Toc45647429)

[**Вывод** 17](#_Toc45647430)

[**РАЗДЕЛ ЧЕТВЕРТЫЙ. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ** 19](#_Toc45647431)

[**Формулировка задания** 19](#_Toc45647432)

[**Теоретический аспект задачи** 19](#_Toc45647433)

[**Спецификация программы** 22](#_Toc45647434)

[**Руководство пользователя** 25](#_Toc45647435)

[**Руководство программиста** 25](#_Toc45647436)

[**Контрольные примеры** 26](#_Toc45647437)

[**Листинг** 27](#_Toc45647438)

[**Вывод** 32](#_Toc45647439)

[**ЗАКЛЮЧЕНИЕ** 33](#_Toc45647440)

[**СПИСОК ЛИТЕРАТУРЫ** 34](#_Toc45647441)

# **ВВЕДЕНИЕ**

Был поставлен ряд задач, которые были выполнены в процессе реализации данной работы:

1) Изучить на практике принципы объектно-ориентированного программирования при разработке приложений, используя знания, полученные в рамках дисциплины «объектно-ориентированное программирование»;

2) Закрепить теоретические знания, приобретённые в рамках курсов «объектно-ориентированное программирование», «математическая логика и теория алгоритмов» и «дискретная математика»

Реализация каждой объектной модели проводится с помощью языка программирования C#. Процесс разработки включает в себя создание удобной графической оболочки с помощью системы WindowsForms.

**Цель работы**

Целью курсовой работы является закрепление теоретических знаний и практических навыков разработки программного обеспечения на основе объектно-ориентированного подхода. Выполнение курсовой работы должно базироваться на объектной модели, являющейся концептуальной базой для объектно-ориентированной парадигмы. В курсовой работе должны быть отражены ключевые концепций объектной модели: абстрагирование, передача сообщений, инкапсуляция, модульность, полиморфизм и наследование.

# **РАЗДЕЛ ТРЕТИЙ. ОСТОВ НАИМЕНЬШЕГО ВЕСА**

## **Формулировка задания**

*Вариант Г-46-кр-4*

С помощью алгоритма Краскала определить остов наименьшего веса, а также найти количество остовов графа. Задачу решить в общем виде.

****

Рисунок 3.1. Контрольный пример

В качестве контрольного примера использовать данное задание.

## **Теоретический аспект задачи**

Алгоритм Краскала - эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

## **Формализация задачи**

Поиск всех остовов дерева заключается в проходе графа в ширину и последовательного добавления пройденных вершин с проверкой на появление цикла. Поиск минимального остова заключается в поиске минимальных ребер графа с проверкой на цикл после их добавления в итоговый граф-остов.

Для решения задачи были созданы классы матрицы графа. В ходе решения было решено добавить класс, преобразующий таблицу элемента DataGridView в матрицу смежности графа, чтобы соблюсти принцип единственной ответственности.

Диаграмма используемых классов представлена на рисунке 3.1.



Рисунок 3.2. Диаграмма классов

Классы:

1. Graph - граф, представленный в виде матрицы;

2. MatrixGrid - класс, осуществляющий взаимодействие между матрицами и таблицами в форме;

3. Control - класс, хранящий константы и методы для подсчета остовов графа;

4. Kruskal\_Algorithm - класс, реализующий алгоритм Краскала.

## **Спецификация программы**

*Класс Graph*

*Таблица 3.1.а – описание методов класса Graph*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Update | Обновляет  матрицу | void | public | DataGridView |

*Таблица 3.1.б – описание полей класса Graph*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| Matrix | int | public | Двумерный массив, хранящий граф |

*Класс MatrixGrid*

*Таблица 3.2.а – описание методов класса MatrixGrid*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| WriteMatrix | Записывает табличные значения в матрицу | void | public | Graph, DataGridView |
| WriteTable | Записывает матричные значения в таблицу | void | public | Graph, DataGridView |
| CreateMatrix | Создаёт новую таблицу для графа | void | public | DataGridView |
| Generate | Генерирует  матрицу | void | public | – |

*Таблица 3.2.б – описание полей класса MatrixGrid*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| MatrixGraph | DataGridView | public | Ссылка на таблицу графа |
| MatrixSkeleton | DataGridView | public | Ссылка на таблицу минимального остова |

*Класс Control*

*Таблица 3.3.а – описание методов класса Control*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| GetOstovCount | Возвращает число остовов графа | int | public | Graph |
| FindAlgAdditional | Находит алгебраические дополнения и записывает их в матрицу алгебраических дополнений | void | private | int[,] A, int size, int[,] B |
| GetMinor | Определяет минор матрицы | int | private | int[,] A, int x, int[,] B, int size, int y |
| FindDeterminant | Возвращает значение определителя матрицы | int | private | int[,] A, int size |

*Таблица 3.3.б – описание полей класса Control*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| VertexCount | int | public | Количество вершин |
| MinLength | int | public | Минимальный вес ребра |
| MaxLength | int | public | Максимальный вес ребра |
| Progress | ProgressBar | public | Ссылка на элемент ProgressBar в форме |

*Класс KruskalAlgorithm*

*Таблица 3.4.а – описание методов класса KruskalAlgorithm*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Go | Запускает алгоритм | Graph | public | Graph graph, DataGridView grid |
| CheckLoop | Проверяет граф (остов) на наличие циклов | bool | private | - |
| DFS | Обход матрицы смежности в глубину | void | private | int i, int prev |
| CheckEdge | Проверяет, рассматривалось ли уже данное ребро | bool | private | List<Edge> edges, int i, int j |

*Таблица 3.4.б – описание полей класса KruskalAlgorithm*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| edges | List<Edge> | private | Хранит рассмотренные ребра графа |
| amount | int | private | Хранит количество циклов в графе |
| ostov | Graph | private | Является минимальным остовом графа |
| start | int | private | Стартовая точка для обхода матрицы в глубину |
| visited | bool[,] | private | Метка посещенности определенного ребра |

*Класс Edge*

*Таблица 3.5.б – описание полей класса Edge*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| i | int | public | Индекс строки матрицы |
| j | int | public | Индекс столбца матрицы |

Описания методов и полей классов представлены в таблицах 3.1.а – 3.5.б.

## **Руководство пользователя**

Назначение программы

Программа реализует нахождение минимального остова графа.

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows версии не ниже XP и .NET Framework версии не ниже 4.0.

Выполнение программы

После запуска программы пользователь задаёт количество вершин графа, после чего нажимает на кнопку «Create». Далее вводятся либо собственные значения, либо генерируются случайно (кнопка «Generate» и ползунки для задания диапазона значений). Для построения остова необходимо нажать кнопку «CreateSkeleton». Подсчет количества всех остовов графа происходит отдельно по нажатию кнопки «To count», и, т.к. при больших размерах графа это занимает какое-то время, была введена полоска прогресса, дабы пользователь не подумал, что программа зависла. После выполнения алгоритма можно создать новый граф.

## **Руководство программиста**

Назначение и условие выполнения программы

См. "Руководство пользователя".

Характеристика программы

Программа разработана в среде Microsoft Visual Studio 2019, использовалась версия платформы .NET 4.0. Язык разработки – C#. Программа отвечает требованиям, описанным в задании, и имеет простой и понятный интерфейс, реализованный с помощью платформы Windows Forms.

Выполнение программы

См. "Руководства пользователя".

Входные и выходные данные

В качестве входных данных выступает матрица смежности, реализующая граф. У матрицы необходимо задать размер. В качестве выходных данных выступает наименьший остов и (по желанию пользователя) количество остовов всего.

## **Контрольные примеры**

Результат обработки контрольного примера, представленного на рисунке 3.1, представлен на рисунке 3.4.

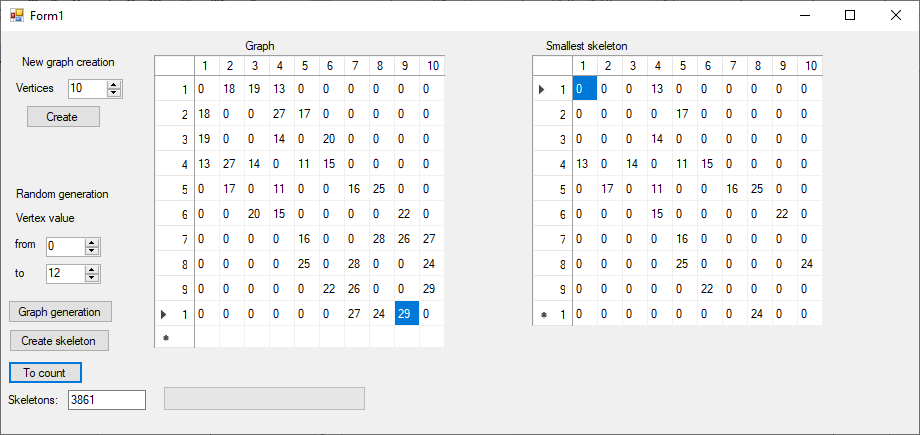


Рисунок 3.4. Результат контрольного примера

## **Листинг**

Graph.cs

class Graph

{

public int?[,] Matrix { get; set; }

public void Update(DataGridView table)

{

Matrix = new int?[Control.VertexCount, Control.VertexCount];

for (int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < Control.VertexCount; j++)

{

if (Convert.ToString(table.Rows[i].Cells[j].Value) == "0" ||

table.Rows[i].Cells[j].Value == null)

Matrix[i, j] = 0;

else if (Convert.ToString(table.Rows[i].Cells[j].Value) != "0" &&

table.Rows[i].Cells[j].Value != null)

Matrix[i, j] = Convert.ToInt32(table.Rows[i].Cells[j].Value);

}

}

}

}

GraphGrid.cs

static class GraphGrid

{

public static DataGridView MatrixGraph { get; set; }

public static DataGridView MatrixSkeleton { get; set; }

public static Random rnd;

//Write data into matrix from table

public static void WriteMatrix(Graph graph, DataGridView grid)

{

graph.Update(grid);

}

//Write data into table from matrix

public static void WriteTable(Graph graph, DataGridView grid)

{

for (int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < Control.VertexCount; j++)

{

if (graph.Matrix[i, j] == 0)

grid.Rows[i].Cells[j].Value = '0';

if (i != j)

{

grid.Rows[i].Cells[j].Value = graph.Matrix[i, j];

if (graph.Matrix[i,j] == 0)

grid.Rows[i].Cells[j].Value = '0';

}

}

}

}

public static void CreateMatrix(DataGridView grid)

{

try

{

grid.Width = 360;

grid.Height = 360;

grid.ColumnCount = Control.VertexCount;

grid.RowCount = Control.VertexCount;

for (int i = 0; i < grid.Columns.Count; i++)

{

grid.Columns[i].Width = 25;

grid.Rows[i].Height = 25;

}

for (int i = 0; i < grid.Rows.Count; i++)

{

grid.Rows[i].HeaderCell.Value = (i + 1).ToString();

grid.Columns[i].HeaderCell.Value = (i + 1).ToString();

}

}

catch(Exception)

{

if (Control.VertexCount == 0)

MessageBox.Show("Размер матрицы не может быть меньше 1");

}

}

public static void Generate()

{

rnd = new Random();

for(int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < Control.VertexCount; j++)

{

if (i == j)

MatrixGraph.Rows[i].Cells[j].Value = '0';

else

{

MatrixGraph.Rows[i].Cells[j].Value = rnd.Next(Control.MinLength, Control.MaxLength + 1);

MatrixGraph.Rows[j].Cells[i].Value = MatrixGraph.Rows[i].Cells[j].Value;

}

}

}

}

public static void FixGraph(DataGridView grid)

{

for (int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < Control.VertexCount; j++)

{

if (i == j)

MatrixGraph.Rows[i].Cells[j].Value = '0';

else

{

MatrixGraph.Rows[i].Cells[j].Value = MatrixGraph.Rows[j].Cells[i].Value;

}

}

}

}

}

Control.cs

class Control

{

public static int VertexCount { get; set; }

public static int MinLength { get; set; }

public static int MaxLength { get; set; }

public static ProgressBar Progress { get; set; }

public static int GetOstovCount(Graph graph)

{

if (VertexCount == 1 || VertexCount == 0)

return 0;

int[,] KirchhoffMatrix = new int[VertexCount, VertexCount];

for (int i = 0; i < VertexCount; i++)

for (int j = 0; j < VertexCount; j++)

KirchhoffMatrix[i, j] = 0;

int row = 0;

int column = 0;

int neighbors = 0;

for (int i = 0; i < VertexCount; i++)

{

for (int j = 0; j < VertexCount; j++)

{

if (graph.Matrix[i, j] != 0 && graph.Matrix != null)

{

KirchhoffMatrix[i, j] = -1;

neighbors++;

}

}

row = i;

column = row;

KirchhoffMatrix[row, column] = neighbors;

neighbors = 0;

}

int[,] B = new int[VertexCount, VertexCount];

int determinant = FindDeterminant(KirchhoffMatrix, VertexCount);

FindAlgAddition(KirchhoffMatrix, VertexCount, B);

return B[0, 0] < 0 ? B[0, 0] \* (-1) : B[0, 0];

}

static void FindAlgAddition(int[,] A, int size, int[,] B)

{

int i, j;

// находим определитель матрицы A

int det = FindDeterminant(A, size);

if (det > 0) // это для знака алгебраического дополнения

det = -1;

else

det = 1;

int[,] minor = new int[size - 1, size - 1];

for (j = 0; j < size; j++)

{

int progress = Progress.Maximum / size;

Progress.Value = Progress.Value + progress;

for (i = 0; i < size; i++)

{

// получаем алгебраическое дополнение

GetMinor(A, minor, j, i, size);

if ((i + j) % 2 == 0)

B[j, i] = -det \* FindDeterminant(minor, size - 1);

else

B[j, i] = det \* FindDeterminant(minor, size - 1);

}

}

}

static int GetMinor(int[,] A, int[,] B, int x, int y, int size)

{

int xCount = 0, yCount = 0;

int i, j;

for (i = 0; i < size; i++)

{

if (i != x)

{

yCount = 0;

for (j = 0; j < size; j++)

{

if (j != y)

{

B[xCount, yCount] = A[i, j];

yCount++;

}

}

xCount++;

}

}

return 0;

}

static int FindDeterminant(int[,] A, int size)

{

// останавливаем рекурсию, если матрица

// состоит из одного элемента

if (size == 1)

{

return A[0, 0];

}

else

{

int det = 0;

int i;

int[,] Minor = new int[size - 1, size - 1];

for (i = 0; i < size; i++)

{

GetMinor(A, Minor, 0, i, size);

// Рекурсия

det += (int)Math.Pow(-1, i) \* A[0, i] \* FindDeterminant(Minor, size - 1);

}

return det;

}

}

}

KruskalAlgorithm.cs

class KruskalAlgorithm

{

class Edge

{

public int i = 0;

public int j = 0;

}

static List<Edge> edges = new List<Edge>();

static int amount; //loops amount

static Graph ostov;

static int start;

static bool[] visited;

public static Graph Go(Graph graph, DataGridView grid)

{

visited = new bool[Control.VertexCount];

ostov = new Graph();

GraphGrid.WriteMatrix(ostov, grid);

int edgeCount = 0;

for (int i = 0; i < Control.VertexCount; i++)

for (int j = 0; j < Control.VertexCount; j++)

{

ostov.Matrix[i, j] = 0;

if (graph.Matrix[i, j] > 0)

edgeCount++;

}

edgeCount /= 2;

int row = 0;

int column = 0;

//Algorithm

while (edges.Count / 2 < edgeCount)

{

Edge edge1 = new Edge();

Edge edge2 = new Edge();

int min = Int32.MaxValue;

for (int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < Control.VertexCount; j++)

{

if (graph.Matrix[i, j] > 0)

{

if (graph.Matrix[i, j] <= min && !CheckEdge(edges, i, j))

{

min = Convert.ToInt32(graph.Matrix[i, j]);

row = i; column = j;

}

}

}

}

edge1.i = row;

edge1.j = column;

edge2.j = row;

edge2.i = column;

edges.Add(edge1);

edges.Add(edge2);

ostov.Matrix[row, column] = graph.Matrix[row, column];

ostov.Matrix[column, row] = graph.Matrix[column, row];

if (CheckLoop())

{

ostov.Matrix[row, column] = 0;

ostov.Matrix[column, row] = 0;

}

}

edges.Clear();

return ostov;

}

private static bool CheckLoop()

{

amount = 0;

for (int i = 0; i < Control.VertexCount; i++)

{

for (int j = 0; j < visited.Length; j++)

visited[j] = false;

start = i;

if (visited[i] == false)

DFS(i, i);

}

return amount > 0;

}

private static void DFS(int i, int prev)

{

visited[i] = true;

for (int r = 0; r < Control.VertexCount; r++)

{

if (r != i && ostov.Matrix[i, r] != 0)

{

if (visited[r] == false)

DFS(r, i);

else if (visited[r] == true && r == start && r != prev)

amount++;

}

}

}

private static bool CheckEdge(List<Edge> edges, int i, int j)

{

if (edges.Count > 0)

{

Edge edge;

for (int r = 0; r < edges.Count; r++)

{

edge = edges.ElementAt(r);

if (edge.i == i && edge.j == j)

return true;

}

}

return false;

}

}

## **Вывод**

Была разработана программа, реализующая нахождение минимального остова и количества всех остовов графа с помощью алгоритма Краскала. Была создана объектная модель для удобного взаимодействия таких элементов программы, как таблицы, предназначенные для отображения необходимых для работы алгоритма матриц, а также сам алгоритм.

Также было закреплено на практике использование таких особенностей языка C#, как свойства и null-типы. Для удобного взаимодействия программы с пользователем был разработан графический пользовательский интерфейс с помощью платформы Windows Forms.

# **РАЗДЕЛ ЧЕТВЕРТЫЙ. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ**

## **Формулировка задания**

*Вариант 28*

*Автомобильная пробка*

1. Смоделировать движение автомобилей в пробке. Для простоты

дорогу считать однополосной. Практика показывает, что многие водители

склонны подъезжать вплотную к идущей впереди машине, сначала

увеличивая свою скорость, чтобы догнать ее, а затем, снижая, чтобы не

врезаться (вплоть до остановки). В результате плотный поток движется не

равномерно, а с пульсациями (каждый, наверное, знает это по своему опыту;

при желании можно найти видео эксперимента, подтверждающего это, в

Интернете).

2. Насколько уменьшается средняя скорость пульсирующего потока

автомобилей по сравнению с равномерным движением?

3. Настроить параметры имитационной модели так, чтобы часть

водителей соблюдала достаточную дистанцию. При какой их доле поток

движется без остановок? Без пульсаций? Обосновать, можно ли частично

разрешить проблему пробок, если водители будут соблюдать большую

дистанцию и двигаться как можно равномернее.

Пользовательский интерфейс должен обеспечивать ввод параметров и

протоколирование результатов имитационного эксперимента. Необходимо

сформировать протокол результатов моделирования. Обеспечить вывод

данных 3-мя способами: в текстовый файл, файл XML и таблицу БД.

Программа должна быть реализована с помощью технологии Windows

Presentation Foundation. Формализация задачи должна быть представлена с

помощью диаграмм UML (диаграммы состояний, последовательностей,

деятельностей, классов и др.)

## **Теоретический аспект задачи**

Предметная область системы

Движение автомобилей в автомобильной пробке.

Описание объекта моделирования

Главный объект моделирования - автомобиль. Объект обладает таким основным свойством, как скорость. В зависимости от поведения других машин определяется поведение отдельно взятого автомобиля. Изменение поведения выражается в изменении скорости.

Цель моделирования

Изучение изменения состояния автомобильной пробки в зависимости от поведения автомобилистов - кто-то старается держать дистанцию, другие пытаются подъехать ближе к машине впереди (иногда настолько близко, что вплоть до остановки).

Характер реализации модели

Выбран машинный характер реализации модели: модель представляет собой программу, позволяющую с помощью последовательных вычислений имитировать процесс функционирования объекта.

Управление модельным временем

В качестве способа изменения времени в модели выбран метод фиксированного шага, т.е. отчет системного времени идет через интервалы времени определенной длины (в реализованной модели это 0,05 с). Таким образом, разработанная модель является дискретной (переменные изменяются дискретно в определенные моменты имитационного времени).

**Формализация задачи**

Для сохранения статистики в XML файл необходимо, чтобы класс, который мы сохраняем, мог быть сериализуем, но также необходимо, чтобы экземпляр класса статистики был одним и только одним. Первоначальная идея сделать класс и его методы статическим противоречит принципам сериализуемости - класс не должен быть статическим. Это можно было бы обойти, но длинным и сложным путем. Тогда был выбран другой способ - класс будет обычным, но реализовывать паттерн одиночки. Тогда экземпляр класса будет всегда один и тот же, и также мы добьемся возможности сериализовать его для записи в XML файл.

Диаграмма используемых классов представлена на рисунке 4.1.



Рисунок 4.1. Диаграмма классов

Классы:

Imitation - основной класс имитационной модели, хранит переменную времени и запускает обновление состояния объектов;

Road - описывает дорогу, хранит очередь из автомобилей и организовывает их передвижение;

Car - описывает автомобиль;

Settings - хранит задаваемые пользователем параметры моделирования;

Statistics - хранит статистические данные о результатах моделирования. Реализует паттерн Singleton.



Рисунок 4.2. Диаграмма деятельности

Диаграмма деятельности представлена на рисунке 4.2.

## **Спецификация программы**

*Класс Imitation*

*Таблица 4.1.а - описание методов класса Imitation*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| StartImitation | Запускает моделирование | void | public | - |
| StopImitation | Останавливает моделирование | void | public | - |
| model | Запускает следующее событие | void | private | object  EventArgs |

*Таблица 4.1.б - описание полей класса Imitation*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| t | int | private | Время моделирования |
| r | Road | private | Автострада |
| timer | DispatcherTimer | private | Обеспечивает изменения состояния через определенный промежуток времени |
| isPlaying | bool | private | Определяет, запущено ли уже моделирование |
| canvas | Canvas | private | Дорога на графическом интерфейсе |
| statistics | TextBox | private | Текстовое поле для статистики на графическом интерфейсе |

*Класс Road*

*Таблица 4.2.а - описание методов класса Road*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| AddCar | Добавляет автомобиль | void | private | - |
| RemoveCar | Удаляет автомобиль | void | private | - |
| NextTime | Переход к следующему состоянию | void | public | t |
| Refresh | Обновление трассы (в GUI) | void | private | - |

*Таблица 4.2.б - описание полей класса Road*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| Cars | Queue<Car> | public | Автомобили |
| RCars | Queue<Rectangle> | public | Автомобили (GUI) |
| TimeToCar | int | private | Время до следующего автомобиля |
| canv | Canvas | private | Дорога на графическом интерфейсе |
| stat | TextBox | private | Текстовое поле для статистики на графическом интерфейсе |

*Класс Car*

*Таблица 4.3.а - описание методов класса Car*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Move | Передвижение | void | public | - |
| SpeedDown | Замедление | void | private | - |
| SpeedUp | Ускорение | void | private | - |

*Таблица 4.3.б - описание полей класса Car*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| StartSpeed | int | public | Начальная скорость |
| CurrentSpeed | int | public | Текущая скорость |
| ExpectedSpeed | int | public | Ожидаемая скорость |
| Coordinate | int | public | Координаты |
| StopTime | int | public | Время остановки |
| Status | char | public | Статус автомобиля (двигается нормально, ускоренно, замедленно, остановился) |
| KeepDistance | bool | public | Шанс того, что этот автомобиль будет стараться соблюдать дистанцию |

## **Руководство пользователя**

Назначение программы

Программа реализует имитационное моделирование движения автомобилей в автомобильной пробке.

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows версии не ниже XP и .NET Framework версии не ниже 4.0.

Выполнение программы

После запуска программы пользователь задаёт интервалы начальной скорости автомобилей и их появления на трассе, а также время процесса моделирования. Нажатие на кнопки «Запуск» запускает процесс моделирования, «Пауза» – останавливает. Кнопка «Сброс» возвращает состояние программы в начальную точку, а «Сохранение статистики» записывает статистику моделирования в файл.

## **Руководство программиста**

Назначение программы и условия выполнения программы

См. "Руководство пользователя".

Характеристика программы

Программа разработана в среде Microsoft Visual Studio 2015, использовалась версия платформы .NET 4.0. Язык разработки – C#. Программа отвечает требованиям, описанным в задании, и имеет простой и понятный интерфейс, реализованный с помощью платформы WPF.

Выполнение программы

См. "Руководство пользователя".

Входные и выходные данные

В качестве входных данных выступают такие параметры моделирования, как интервалы начальной скорости автомобилей и их появления на трассе, время процесса моделирования, а также доля водителей, соблюдающих дистанцию.

В качестве выходные данных выступают отображения изменяющихся состояний автомобилей на трассе, а также статистические данные о ходе моделирования, представленные в специальном окне.

## **Контрольные примеры**

На рисунке 4.5 продемонстрировано состояние процесса моделирования в определённый момент времени. Заданные параметры: начальная скорость – от 8 до 15 км/ч, интервал появления автомобилей – от 1 с, время имитации – не ограничено. Текущее время моделирования – 11 с, максимальная зафиксированная скорость – 38 км/ч, минимальная - 1 км/ч, количество автомобилей – 11, равномерный поток быстрее пульсирующего в 2 раза.

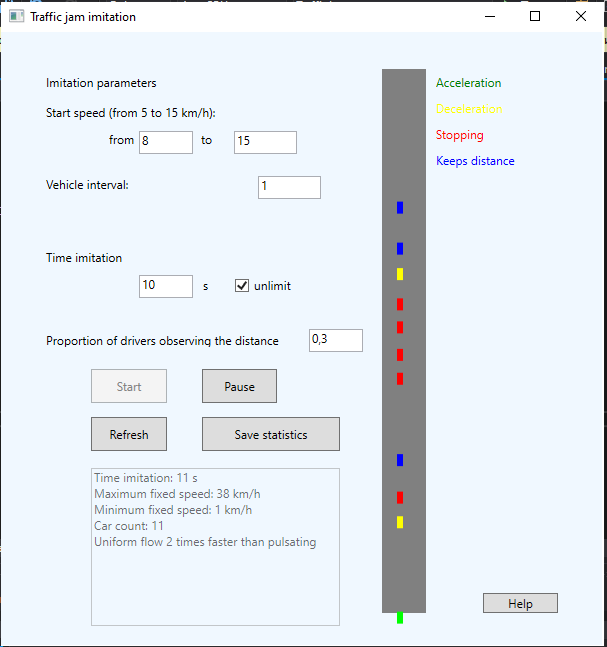


Рисунок 4.5. Пример работы программы

При нажатии "Save statistics" статистика сохраняется в файлы statistics.txt и statistics.xml.

## **Листинг**

Car.cs

class Car

{

public double StartSpeed { get; set; }

public double CurrentSpeed { get; set; }

public double ExpectedSpeed { get; set; }

public double Coordinate { get; set; }

public bool KeepDistance { get; set; } //indicates whether the driver is keeping a distance

static Random rnd = new Random();

/\*

\* Status - vehicle condition

\* n - normal, keeps speed

\* d - deceleration, speed decreases every second

\* u - acceleration, speed increases every second

\* s - the car is in stop state

\* \*/

public char Status { get; set; }

public int StopTime { get; set; }

public Car()

{

StartSpeed = rnd.Next(Settings.MinSpeed, Settings.MaxSpeed);

CurrentSpeed = StartSpeed;

Coordinate = 0;

Status = 'n';

StopTime = 0;

if (Settings.values[rnd.Next(0, 100)] == 1) //imitation of the chance that the driver keeps a distance

KeepDistance = true;

else

KeepDistance = false;

}

public void Move()

{

switch (Status)

{

case 'd':

SpeedDown();

Coordinate += CurrentSpeed / 20;

break;

case 'u':

SpeedUp();

Coordinate += CurrentSpeed / 20;

break;

case 's':

StopTime -= 1;

if (StopTime == 0)

{

Status = 'u';

ExpectedSpeed = StartSpeed;

}

break;

case 'n':

Coordinate += CurrentSpeed / 20;

break;

}

}

private void SpeedDown() // deceleraton

{

if (CurrentSpeed - ExpectedSpeed <= 5)

{

CurrentSpeed = ExpectedSpeed;

Status = 'n';

}

else

CurrentSpeed -= 5;

}

private void SpeedUp() // acceleration

{

if (ExpectedSpeed - CurrentSpeed <= 5)

{

CurrentSpeed = ExpectedSpeed;

Status = 'n';

}

else

CurrentSpeed += 5;

}

}

Road.cs

class Road

{

public Queue<Car> Cars { get; set; }

public Queue<Rectangle> RCars { get; set; }

double TimeToCar; //time to next car

Random rnd;

Canvas canv;

TextBox stat;

public Road(Canvas canvas, TextBox statistic)

{

Cars = new Queue<Car>();

RCars = new Queue<Rectangle>();

TimeToCar = 0;

rnd = new Random();

canv = canvas;

stat = statistic;

}

void AddCar()

{

Cars.Enqueue(new Car());

}

void RemoveCar()

{

Cars.Dequeue();

}

public void NextTime(ref int t) //next road condition

{

Car PrevCar = null;

if (TimeToCar <= 0)

{

AddCar();

Statistics.GetInstance.CarCount++;

TimeToCar = Settings.Interval \* 20;

}

else TimeToCar--;

foreach (var x in Cars)

{

double min = Double.MaxValue;

double max = 0;

x.Move();

foreach (var car in Cars)

{

if (car.CurrentSpeed > max)

max = car.CurrentSpeed;

if (min > car.CurrentSpeed && car.CurrentSpeed != 0)

min = car.CurrentSpeed;

}

if (Statistics.GetInstance.MinSpeed > min)

Statistics.GetInstance.MinSpeed = (int)min;

if (Statistics.GetInstance.ImitationTime % 20 == 0)

Statistics.GetInstance.MaxDiff = (int)(max / min);

stat.Text = "Time imitation: " + (Statistics.GetInstance.ImitationTime / 20 + 1) + " s\n"

+ "Maximum fixed speed: " + Statistics.GetInstance.MaxSpeed + " km/h\n"

+ "Minimum fixed speed: " + Statistics.GetInstance.MinSpeed + " km/h\n"

+ "Car count: " + Statistics.GetInstance.CarCount;

if (Convert.ToInt32(x.CurrentSpeed) > Statistics.GetInstance.MaxSpeed)

Statistics.GetInstance.MaxSpeed = Convert.ToInt32(x.CurrentSpeed);

stat.Text += $"\nUniform flow {Statistics.GetInstance.MaxDiff} times faster than pulsating";

if (PrevCar != null)

{

if (x.Status != 's') //if doesn't standing

{

if (x.CurrentSpeed > 30) //speed limit for free road

{

x.Status = 'd';

x.ExpectedSpeed = 30;

}

else

{

if (x.KeepDistance) //if this car keeps a distance (it just keep more distance to the next car)

{

if (Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) <= 10)

{

x.Status = 's';

x.StopTime = 6;

x.CurrentSpeed = 0;

}

else

if (Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) <= 25)

{

x.Status = 'd';

x.ExpectedSpeed = PrevCar.CurrentSpeed;

}

else

if ((Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) > 25) && (x.Status != 'u') && (x.CurrentSpeed < x.StartSpeed + 5))

{

x.Status = 'u';

x.ExpectedSpeed = x.CurrentSpeed + 3;

}

if (x.Status == 'd' && Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) > 20)

{

x.Status = 'u';

x.ExpectedSpeed = x.StartSpeed;

}

}

else //this car is trying to drive close

{

if (Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) <= 5)

{

x.Status = 's';

x.StopTime = 6;

x.CurrentSpeed = 0;

}

else if (Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) < 6)

{

x.Status = 'd';

x.ExpectedSpeed = PrevCar.CurrentSpeed + 1;

}

else if (Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) <= 10)

{

x.Status = 'd';

x.ExpectedSpeed = PrevCar.CurrentSpeed + 5;

}

else

if ((Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) > 10) && (x.Status != 'u'))

{

x.Status = 'u';

x.ExpectedSpeed = x.CurrentSpeed + 5;

}

if (x.Status == 'd' && Math.Abs(PrevCar.Coordinate - 3 - x.Coordinate) > 10)

{

x.Status = 'u';

x.ExpectedSpeed = x.StartSpeed;

}

}

}

}

}

PrevCar = x;

}

if (Cars.Count != 0 && Cars.Peek().Coordinate >= 185) //if road ended

{

RemoveCar();

}

Refresh();

}

private void Refresh() // refresh the road (visually)

{

foreach (var x in RCars)

{

canv.Children.Remove(x);

}

foreach (var x in Cars)

{

Rectangle r;

SolidColorBrush scb = null;

switch (x.Status)

{

case 's':

scb = new SolidColorBrush(Color.FromRgb(255, 0, 0));

break;

case 'd':

scb = new SolidColorBrush(Color.FromRgb(255, 255, 0));

break;

case 'u':

scb = new SolidColorBrush(Color.FromRgb(0, 255, 0));

break;

case 'n':

scb = new SolidColorBrush(Color.FromRgb(255, 255, 255));

break;

}

if (x.KeepDistance == true)

scb.Color = Color.FromRgb(0, 0, 255);

r = new Rectangle()

{

Fill = scb,

Width = 6,

Height = 12

};

Canvas.SetTop(r, canv.Height - (x.Coordinate) \* 3);

Canvas.SetLeft(r, 15);

RCars.Enqueue(r);

canv.Children.Add(r);

}

}

}

Imitation.cs

class Imitation

{

static int t;

static Road r;

static DispatcherTimer timer = null;

static bool isPlaying;

static Canvas canvas;

static TextBox statistic;

public static void SetParameters(Canvas canv, TextBox stat)

{

canvas = canv;

statistic = stat;

t = 0;

r = new Road(canvas, statistic);

isPlaying = false;

}

public static void StartImitation()

{

if(isPlaying == false)

{

isPlaying = true;

timer = new DispatcherTimer();

timer.Tick += new EventHandler(model);

timer.Interval = new TimeSpan(0, 0, 0, 0, 50);

timer.Start();

}

}

public static void StopImitation()

{

timer?.Stop();

isPlaying = false;

}

static void model(object sender, EventArgs e)

{

r.NextTime(ref t);

t++;

Statistics.GetInstance.ImitationTime++;

if(t >= (Settings.ImitationTime \* 20))

{

StopImitation();

if (Settings.Unlimit == true)

StartImitation();

else t = 0;

}

}

}

## **Вывод**

Была разработана имитационная модель, описывающая поведение автомобилей при движении в автомобильной пробке на однополосной автостраде.

Также разработана программа, реализующая созданную модель. Для удобного взаимодействия программы с пользователем был разработан графический пользовательский интерфейс с помощью платформы Windows Presentation Foundation.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были закреплены навыки работы с языком объектно-ориентированного программирования С#, освоен процесс создания программ с использованием Windows Forms в среде Microsoft Visual Studio. Разработаны четыре приложения с графическим пользовательским интерфейсом. Приложения работают корректно и соответствуют поставленным задачам и корректному графическому отображению в окне программы при помощи соответствующих инструментов.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Горячев А.В., Кравчук Д.К., Новакова Н.Е. Объектно-ориентированное моделирование. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 2010.

2. Язык UML: руководство пользователя,Г. Буч, Д. Рамбо, А. Джекобсон.СПб.: Питер, 2004. 429 с.

3. Шилдт Г. С#. Полный справочник, 2004.

4. Сайт разработчиков Microsoft. // Microsoft Developer Network. URL: <https://msdn.microsoft.com/en-us> (дата обращения: 15.12.2019)