Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ПРОГРАММНОЕ СРЕДСТВО ПОСТРОЕНИЯ И ОТОБРАЖЕНИЯ СЕТЕВЫХ И ДРЕВОВИДНЫХ СТРУКТУР

БГУИР КР 1-40 01 01 119 ПЗ

Студент: гр. 951001 Северин К.М.

Руководитель:

асс. Фадеева Е.Е.

Минск 2020

       Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту группы 951001 Северину Климу Михайловичу

1. Тема работы: Программное средство построения и отображения сетевых и древовидных структур

2. Срок сдачи студентом законченной работы––01.06.2020 г.–––

3. Исходные данные к работе: язык программирования Delphi; структура данных – динамический массив для хранения данных; интерфейс для создания и отображения структур, с возможностью сохранения структур в файл, а также открытия таких файлов.

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Программное средство построения и отображения сетевых и древовидных структур", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Фадеева Е.Е.

7. Дата выдачи задания ––11.02.2020 г.–––

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.03.2020 – 15 % готовности работы;

разделы 2, 3 к 15.03.2020 – 30 % готовности работы;

разделы 4, 5 к 15.04.2020 – 60 % готовности работы;

раздел 6 к 15.05.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 20.05.2020 – 100 % готовности работы.

Защита курсовой работы с 01.06.2020 по 09.06.2020 г.––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Е.Е.Фадеева

(подпись)

Задание принял к исполнению –––\_\_\_\_––

(дата и подпись студента)

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 5](#_Toc42246092)

[**1.** **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ** 6](#_Toc42246093)

[**1.1** **Анализ программных средств построения и отображения сетей и древовидных структур** 6](#_Toc42246094)

[**1.1.1** **Microsoft Visio** 6](#_Toc42246095)

[**1.1.2** **EdrawMax** 7](#_Toc42246096)

[**1.1.3** **GoVisual Diagram Editor** 8](#_Toc42246097)

[**1.2** **Формирование требований к проектируемому программному средству** 9](#_Toc42246098)

[**2.** **АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ** 10](#_Toc42246099)

[**2.1** **Описание функциональности программного средства** 10](#_Toc42246100)

[**3.** **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** 13](#_Toc42246101)

[**3.1** **Обобщенный алгоритм работы программного средства** 13](#_Toc42246102)

[**3.2** **Разработка класса для хранения данных** 13](#_Toc42246103)

[**3.3** **Разработка классов для хранения данных о созданных пользователем блоков** 14](#_Toc42246104)

[**3.3.1** **Описание класса TControlObject** 14](#_Toc42246105)

[**3.3.2** **Описание класса TText** 16](#_Toc42246106)

[**3.3.3** **Описание класса TElement** 17](#_Toc42246107)

[**3.3.4** **Описание класса TLine** 19](#_Toc42246108)

[**3.3.5** **Описание структуры TConnecor** 20](#_Toc42246109)

[**4.** **СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА** 22](#_Toc42246110)

[**4.1** **Разработка главной формы программного средства** 22](#_Toc42246111)

[**4.2** **Разработка модального окна изменения параметров документа** 27](#_Toc42246112)

[**5.** **ТЕСТИРОВАНИЕ ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ** 29](#_Toc42246113)

[**6.** **РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ** 32](#_Toc42246114)

[**ЗАКЛЮЧЕНИЕ** 36](#_Toc42246115)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ** 37](#_Toc42246116)

# **ВВЕДЕНИЕ**

В наше время существует большое количество различных программных средств, в алгоритмах которых используется теория графов. Например, Google maps. Существует так же много известных алгоритмов использующие деревья, как структуру данных

**Граф** — это абстрактное представление множества объектов и связей между ними.

Деревья и сети являются частным случаем графа.

Древовидная структура является одним из способов представления иерархической структуры в графическом виде.

Простейшим примером графа может стать карта метро, где станции будут составлять множество вершин, а дороги между станциями будут составлять множество ребер.

Целью курсовой работу является разработка программного средства, в котором можно строить графы и его подвиды в графическом отображении.

Данная пояснительная записка содержит следующие основные разделы. В первом разделе выполнен анализ прототипов, литературных источников. Во втором разделе сформированы функциональные требования к проектируемому программному средству. Третий раздел посвящен проектированию программного средства. Четвертый раздел описывает этапы создания программного средства. Пятый радел содержит набор тестов и сценарии тестирования. В шестом разделе описано руководство по установке и использованию. Завершающий раздел содержит выводы.

# **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

## **Анализ программных средств построения и отображения сетей и древовидных структур**

В настоящее время большинство программ, в которых можно построить некоторую схему или граф, позиционируют себя как средство для создания различного уровня диаграмм и имеют большое количество шаблонов.

Общепринятого формата файла для таких программ нет, т.е. каждая компания создает свой тип файла.

## **Microsoft Visio**

Microsoft Visio предоставляет возможности для быстрого создания деловой графики различной степени сложности: схем бизнес процесса, технических, инженерных рисунков, презентаций, разнообразных вариантов организационных, маркетинговых и технических диаграмм, электронных схем, систем транспортных коммуникаций и т. д.

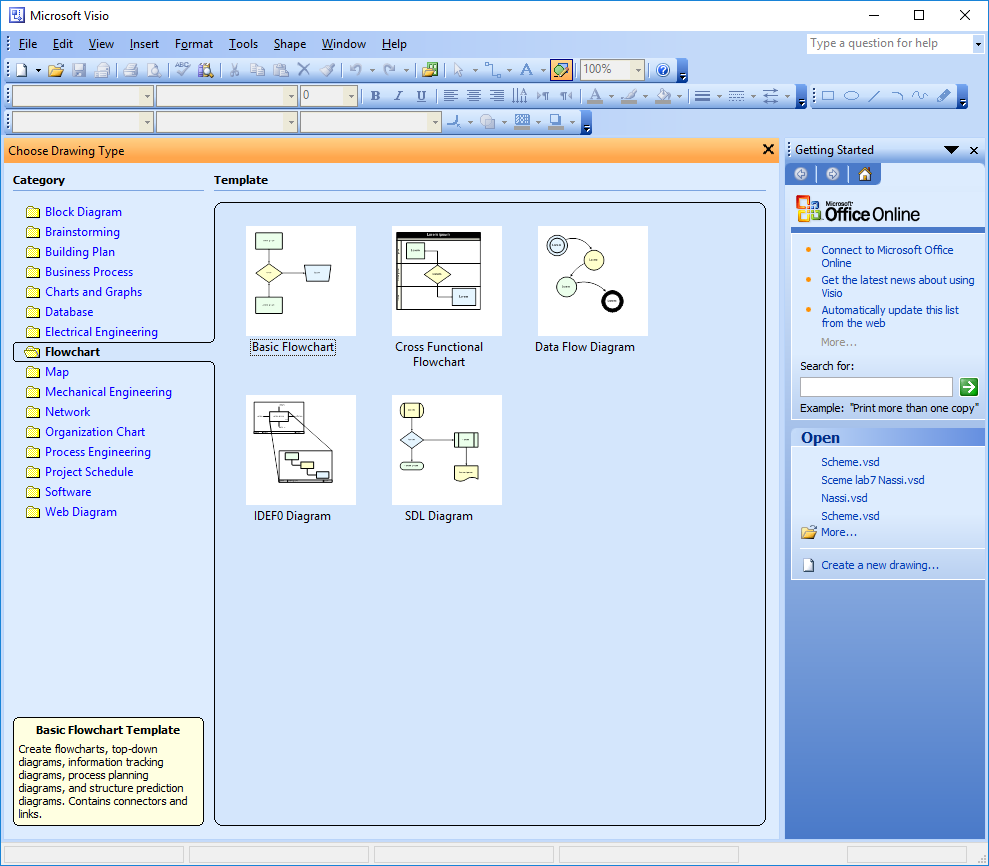


Рисунок 1.1 – Окно Microsoft Visio

Достоинства:

* Векторный графический редактор
* Имеет достаточно большую библиотеку различных шаблонов
* Шаблоны собраны по категориям
* Традиционный для приложений Windows интерфейс
* Поддерживает большое количество форматов. (В основном это форматы поддерживаемые исключительно программой Visio. VSD, VSS, VST, VDX, VSX, VTX, VSL, VSDX, VSDM)
* Возможность сохранение рабочей области в большинство форматов графических изображений (JPEG, PNG, GIF, BMP и др.)
* Возможность изменять вид блоков

Недостатки:

* Не самый удобный способ выбора графического элемента. (Каждый раз приходиться искать нужный блок).
* Жёсткая политика лицензирования. Продукт стоит достаточно дорого, (около $345)
* Доступна только на операционной системе Windows

## **EdrawMax**

Edraw Max - приложение для деловой графики, в котором удобно создавать схемы, диаграммы, инфографику, иллюстрации для презентационных нужд и деловой литературы.

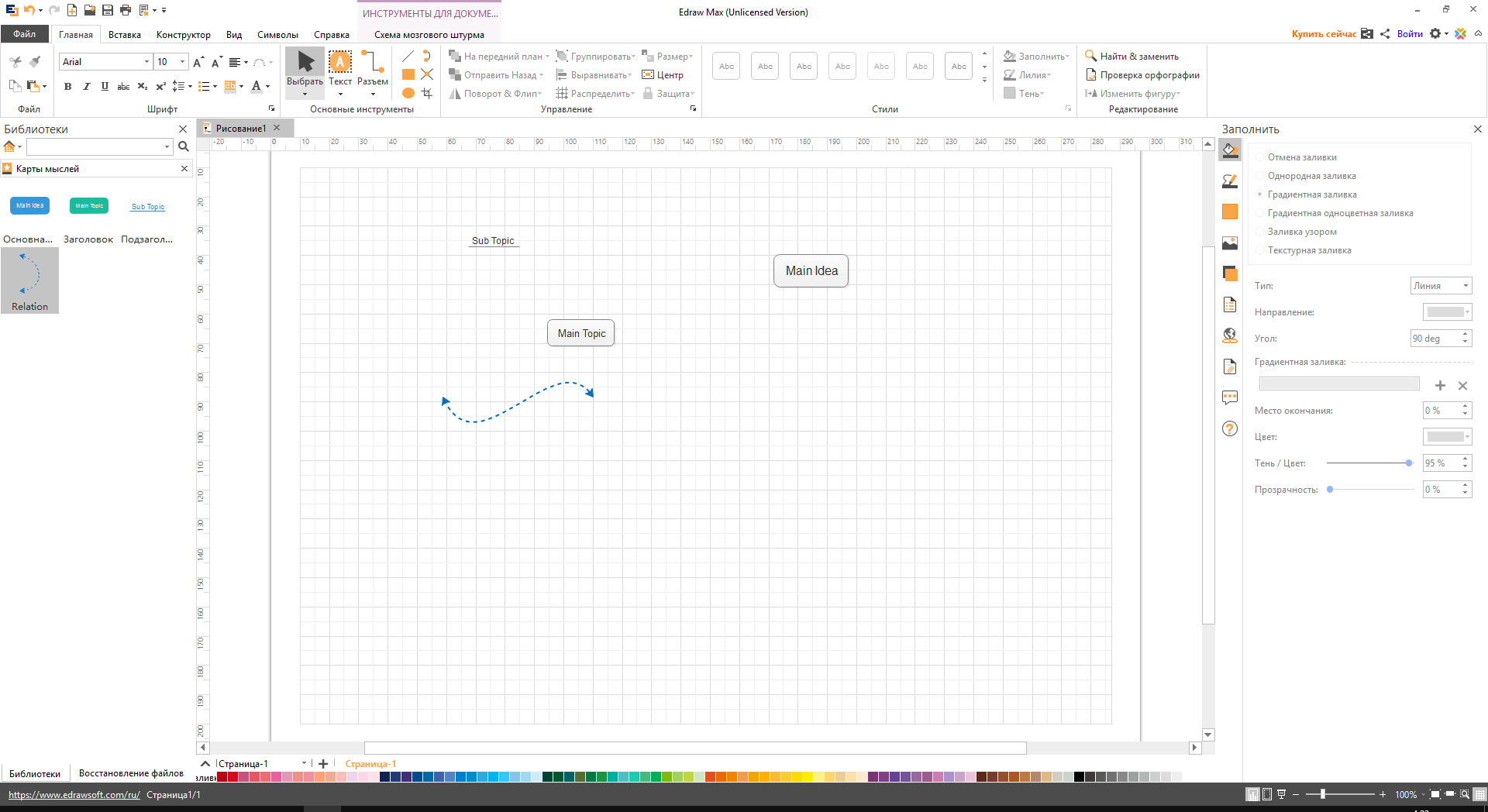


Рисунок 1.2 – Окно программы Edraw Max

Достоинства:

* Доступна на операционных системах Windows, Linux, MacOS
* Векторный графический редактор
* Включает в себя 280 типов диаграмм
* Поддерживает такие форматы Edraw XML файлов .eddx, .edx
* Есть пробная версия
* Возможность изменять вид блоков
* Cтандартный интерфейс приложений из популярного пакета MS Office.
* Возможность сохранение рабочей области в большинство форматов графических изображений (JPEG, PNG, GIF, BMP и др.)

Недостатки:

* Пробная версия доступна в течении 1 месяца. Лицензия же стоит $170
* Из-за большой библиотеки типов диаграмм, много времени уходит на поиск нужного шаблона
* Поиск и выбор графического элемента происходит по большому количеству категорий, что занимает много времени.

## **GoVisual Diagram Editor**

Редактор диаграмм GoVisual (GDE) предоставляет мощные функции для редактирования и автоматического размещения диаграмм. Диаграммы представлены в виде графиков и кластерных графов.

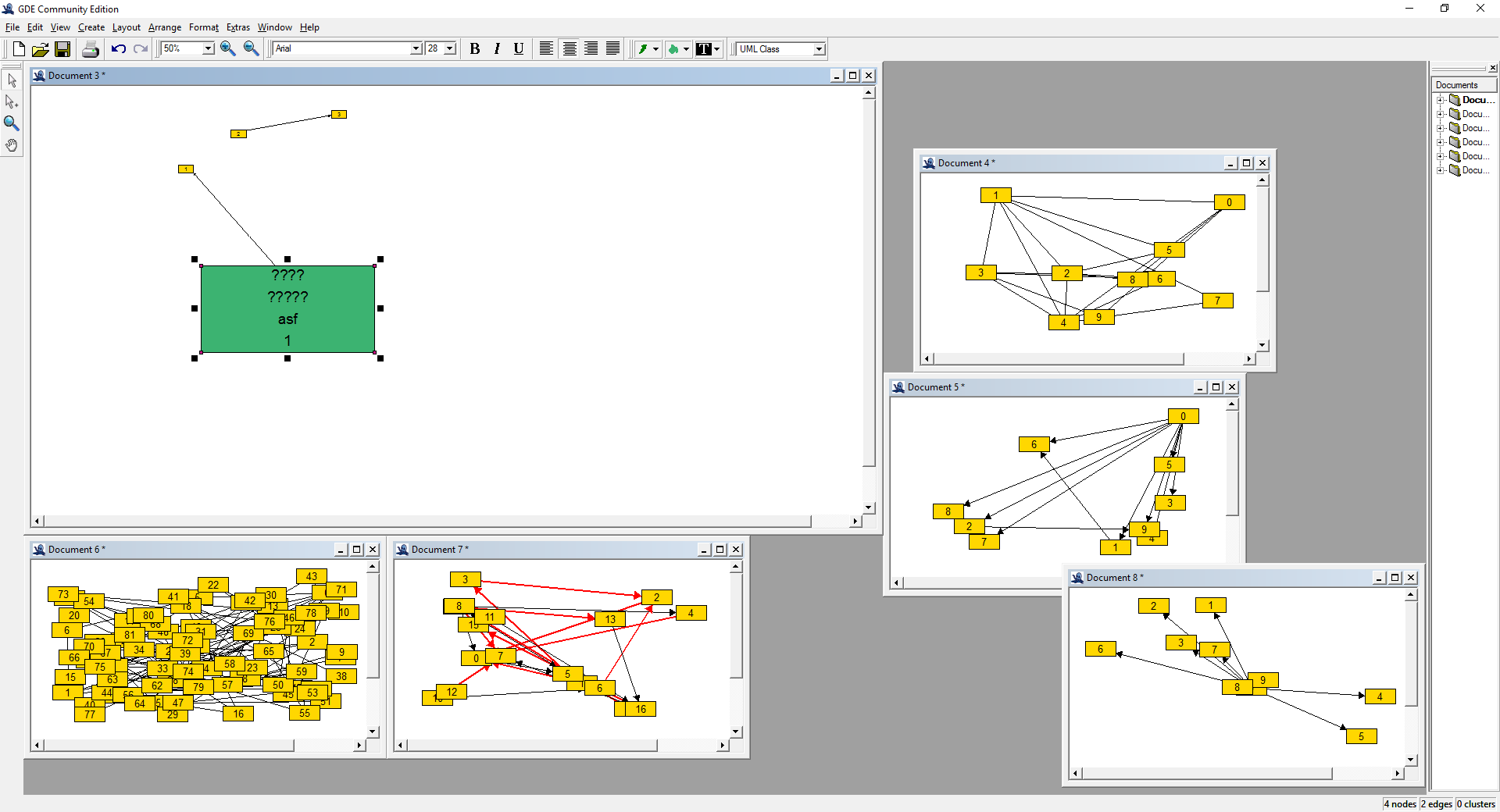


Рисунок 1.3 – Окно программы GoVisual Diagram Editor

Достоинства:

* Является полностью бесплатной программой
* Доступна на таких платформах как Windows, Linux
* Удобный интерфейс
* Быстрое создание диаграмм, графов, деревьев
* Есть функции генерирования случайных графов, деревьев по определенным параметрам
* Возможность работы сразу с несколькими проектам в одном приложении
* Поддерживает единственный формат файла .gml
* Возможность экспорта в JPEG, PNG, BMP SCG, CSV

Недостатки:

* Поддержка программного средства закончилась в 2004 году
* Нет поддержки Unicode. (Отсутствует кириллица)
* Есть лишь один тип графических элементов: прямоугольник
* Ограниченные возможности изменения графических элементов (Цвет блока, текст, размер)

## **Формирование требований к проектируемому программному средству**

Согласно заданию на курсовой проект необходимо разработать программное средство для построения и отображения древовидных структур и сетей. Основные функциональные возможности:

* Выполнять экспорт рабочей области в популярные форматы изображений BMP, JPEG и др.
* Поддерживать специфический формат файла, конкретно данного программного средства
* Иметь удобный графический пользовательский интерфейс
* Поддерживать несколько видов инструментов создания схем
* Изменение параметров рабочей области
* Поддерживать несколько форм графических блоков
* Иметь возможность уменьшать/увеличивать рабочую область

В качестве языка программирования было предложено использовать язык Delphi. Платформой разработки является операционная система Windows, так как большая часть аудитории пользуется именно ей.

# **АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **Описание функциональности программного средства**

Выполнение экспорта рабочей области в форматы изображений я решил объединить с сохранением в специфический формат файла.

Для сохранения в какой-либо поддерживаемый формат файла пользователю необходимо пройти по меню «File –Save as» или воспользоваться горячей клавишей Ctrl+S, в результате чего откроется диалоговое окно c выбором места сохранения, имени и расширения файла.

Для открытия специфичного файла, пользователь должен нажать в главном меню программы «File», а затем «Open», или воспользоваться горячей клавишей Ctrl+O. Далее пользователь увидит диалоговое окно с выбором открываемого файла.

Выбор инструмента создания схемы или вида графического элемента производиться через кнопки на боковой панели.

При выборе некоторого инструмента и клику по рабочей области происходит взаимодействие с блоками.

Для изменения параметров рабочей области можно пройти по следующим пунктам главного меню программы «Image – Resolution»

* 1. Спецификация функциональных требований

Среди функциональных требований есть поддержка нескольких видов инструментов. В программном средстве реализованы следующие инструменты:

* «Мышь»
  + Пользователь должен иметь возможность выделять созданные объекты для дальнейшего взаимодействия
  + Пользователь должен иметь возможность перемещать созданные объекты
* «Прямоугольник»
  + Пользователь должен иметь возможность создать элемент с формой прямоугольника
* «Эллипс»
  + Пользователь должен иметь возможность создать элемент с формой эллипса
* «Круг»
  + Пользователь должен иметь возможность создать элемент с формой круга
* «Линия»
  + Пользователь должен иметь возможность создавать линии на рабочем пространстве
* «Текст»
  + Пользователь должен иметь возможность напечатать текст на рабочем пространстве

В программном средстве должны быть реализованы следующие формы графических блоков:

* Прямоугольник
* Эллипс
* Круг
* Линия
* Текст

Для графических элементов «Прямоугольник», «Эллипс», «Круг»:

* Пользователь должен иметь возможность изменять внешний вид блоков:
  + Изменять размер блока
  + Изменять точно положение блока
  + Менять цвет фона блока
  + Писать текст внутри блока
  + Менять шрифт текста
  + Менять цвет текста
  + Менять кегль текста
  + Менять цвет линии обводки
  + Менять толщину линии обводки

Для графического элемента «Линия»:

* Пользователь должен иметь возможность менять внешний вид линии
  + Менять цвет линии
  + Менять толщину линии
* Должна быть реализована привязки линии к блокам «Прямоугольник», «Эллипс», «Круг»

Для графического элемента «Текст»

* Пользователь должен иметь возможность менять внешний вид текста
  + Изменять размер поля с текста
  + Изменять точно положение текста
  + Менять шрифт текста
  + Менять цвет текста
  + Менять кегль текста

Спецификация функции экспорта рабочей области в форматы изображений:

* Поддерживаемые форматы:
  + JPEG
  + PNG
  + BMP
* Пользователь должен видеть привычное диалоговое окно сохранения файла в операционной системе Windows

Сохранение и открытие специфичного формата файла:

* Расширение файла .tmf (TreeMakerFile)
* Должна производиться запись/чтение из файла:
  + Размера рабочей области
  + Информации о количестве блоков, о параметрах каждого блока (к параметрам относятся: размер; положение; цвет фона; цвет и толщина линии обводки; шрифт, цвет, кегль и размер текста)
  + Информации о положении линии и о возможной привязке линии к какому-нибудь блоку.

Удаление блоков, расположенных на рабочей области происходит через выбор пункта меню «Edit - Delete» или по горячей клавише Del.

Изменение параметров рабочей области производиться через модальное окно. В модальном окне можно изменить два параметра: ширину и высоту рабочей области. Открытие модального окна происходит через пункт меню «Image - Resolution» Блоки, расположенные на рабочей области, при уменьшении/увеличении ее размеров, изменяются пропорционально ней.

Приближение/отдаление рабочей области в нужном направлении производиться наведением курсора мыши в нужную и прокруткой колесика мыши с зажатой кнопкой Ctrl.

# **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **Обобщенный алгоритм работы программного средства**

Схема алгоритма программы представлена в приложении А

## **Разработка класса для хранения данных**

Для хранения созданных пользователем блоков мною было решено использование динамических массивов в качестве структуры данных.

Класс TVector<T> отвечает за реализацию этой структуры данных.

Таблица 3.1 - Поля класса TVector<T>

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| FData | Динамический массив для хранения данных | Array of T; |
| FDataSize | Размер(количество) используемых ячеек массива | Integer |
| FReservedSize | Фактический размер динамического массива | Integer |

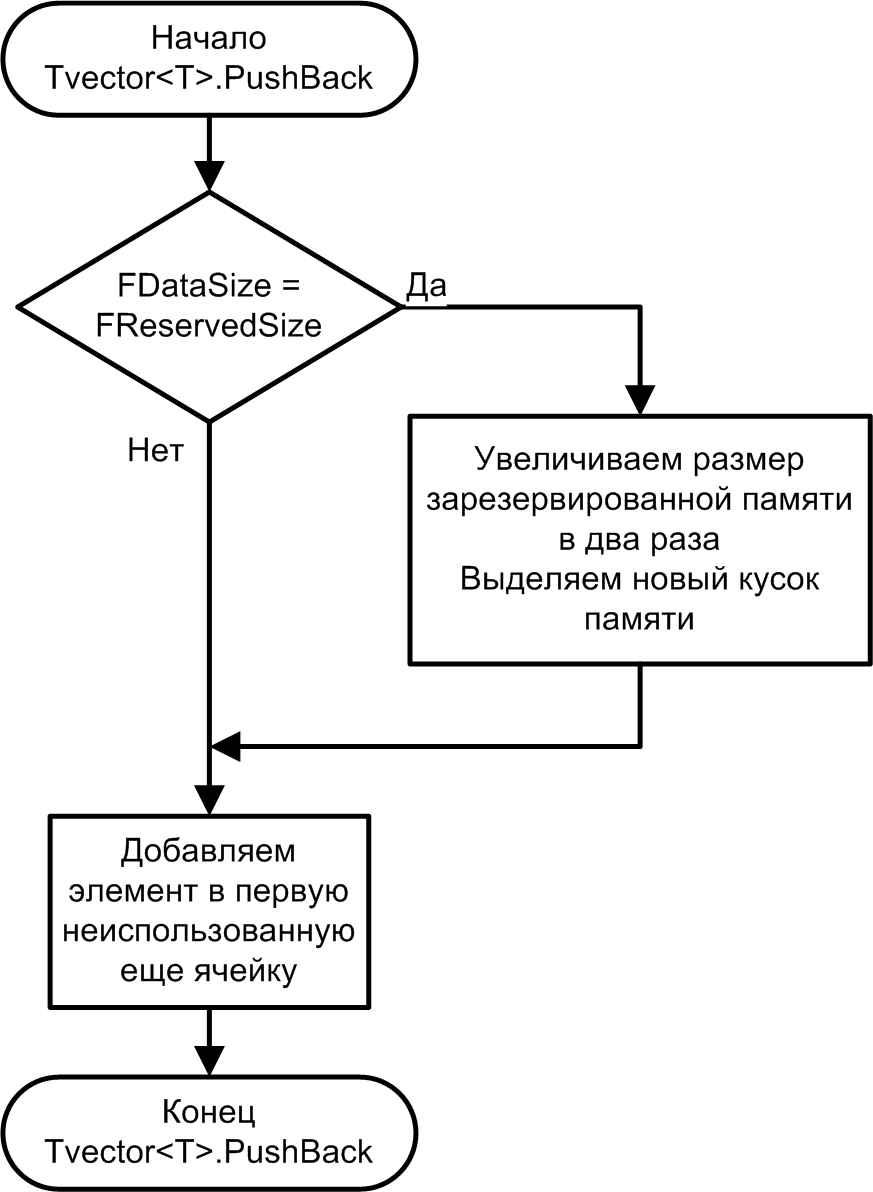


Рисунок 3.1 – Блок-схема добавления элемента в конец динамического массива

Для возможности увеличения в два зарезервированной памяти мною было решено хранить два значения, объем зарезервированной памяти и заполненной мною памяти. Это нужно для того чтобы улучшить производительность программного средства. Если использовать обычный алгоритм добавления элемента в конец массива, то есть увеличивать размер динамического массива на единицу, то мы получим на N запросов добавления элемента в конец асимптотике O(N^2), так как каждый раз будет производиться поиск нового куска памяти нужного нам размера, а затем поэлементно копироваться в новую память. В свою очередь, если мы будем увеличивать размер динамического массива вдвое, то это позволит нам достичь асимптотики в O(N logN), что значительно ускоряет работу с программой при больших объемах данных. Схема алгоритма добавления элемента в массив представлена на рисунке 3.1.

## **Разработка классов для хранения данных о созданных пользователем блоков**

Для хранения информации о блоках, расположенных на рабочем пространстве было решено создать следующие классы:

1. TControlObject
2. TText наследующий TControlObject
3. TElement наследущий TControlObject
4. TLine

Также была создана структура TConnector для описания позиции конца линии

### **Описание класса TControlObject**

В таблице 3.2 приведены поля класса TControlObject

В таблице 3.3 приведено описание основных методов класса TControlObject

Таблица 3.2 - Поля класса TControlObject

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| Fid | Уникальный номер блока | Integer |
| FIsVisible | Флаг, означающий является ли элемент видимым | Boolean |
| FIsSelected | Флаг, означающий выделен ли сейчас элемент | Boolean |

Таблица 3.2 - Поля класса TControlObject (Продолжение)

|  |  |  |
| --- | --- | --- |
| FCanvas | Объект класса TCanvas. Нужен для отрисовки блока | TCanvas |
| FLeft | Положение левой границы блока относительно левой границы документа | Integer |
| FTop | Положение верхней границы блока относительно верхней границы документа | Integer |
| FWidth | Ширина блока | Integer |
| FHeigth | Высота блока | Integer |

Таблица 3.3 – Основные методы класса TControlObject

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| SetPosition | Устанавливает блок в позицию X, Y | procedure SetPosition(const X, Y: Integer); | X | Новое значение положения левой границы блока |
| Y | Новое значение положения верхней границы блока |
| Move | Перемещает блок на DeltaX единиц по горизонтали и на DeltaY единиц по вертикали | procedure Move(const DeltaX, DeltaY: Integer); | DeltaX | Перемещение по горизонтали |
| DeltaY | Перемещение по вертикали |

Таблица 3.3 – Основные методы класса TControlObject (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SetSize | Устанавливает ширину блока в AWidth единиц и высоту блока в AHeigth единиц | procedure SetSize(const AWidth, AHeigth: Integer); | AWidth | Новое значение ширины блока |
| AHeigth | Новое значение высоты блока |

### **Описание класса TText**

В таблице 3.4 приведены поля класса TText

В таблице 3.5 приведены основные методы класса TText

Таблица 3.4 - Поля класса TText

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| Унаследованные поля от класса TControlObject (смотри таблицу 3.2) | | |
| FCaption | Текст, отображаемый на блоке | String |
| FBrush | Кисть, которой отрисовывается фон блока | TBrush |
| FFont | Шрифт текста | TFont |
| FTextFormat | Положение текста | TTextFormat |

Таблица 3.5 – Основные методы класса TText

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| Draw | Отрисовка текста | procedure Draw; | - | - |

Таблица 3.5 – Основные методы класса TText (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IsInside | Выдает истину, если точка c координатами X, Y находиться внутри блока, иначе ложь | function IsInside(const X, Y: Integer): Boolean; | X | Положение точки по горизонтали |
| Y | Положение точки по вертикали |

### **Описание класса TElement**

В таблице 3.6 приведены поля класса TElement

В таблице 3.7 приведены основные методы класса TElement

На рисунке 3.2 приведен алгоритм отрисовки объекта класса TElement

Таблица 3.6 – Поля класса TElement

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| Унаследованные поля от класса TControlObject (смотри таблицу 3.2) | | |
| FShape | Форма текущего блока (Эллипс, прямоугольник и др.) | TShapeType |
| FBrush | Кисть, которой отрисовывается фон блока | TBrush |
| FPen | Ручка, которой отрисовываются линии блока | TPen |
| FCaption | Текст, отображаемый на блоке | String |
| FFont | Шрифт текста | TFont |
| FTextFormat | Положение текста | TTextFormat |

Таблица 3.7 – Основные методы класса TElement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| Draw | Отрисовка текста | procedure Draw; | - | - |
| IsInside | Выдает истину, если точка c координатами X, Y находиться внутри блока, иначе ложь | function IsInside(const X, Y: Integer): Boolean; | X | Положение точки по горизонтали |
| Y | Положение точки по вертикали |

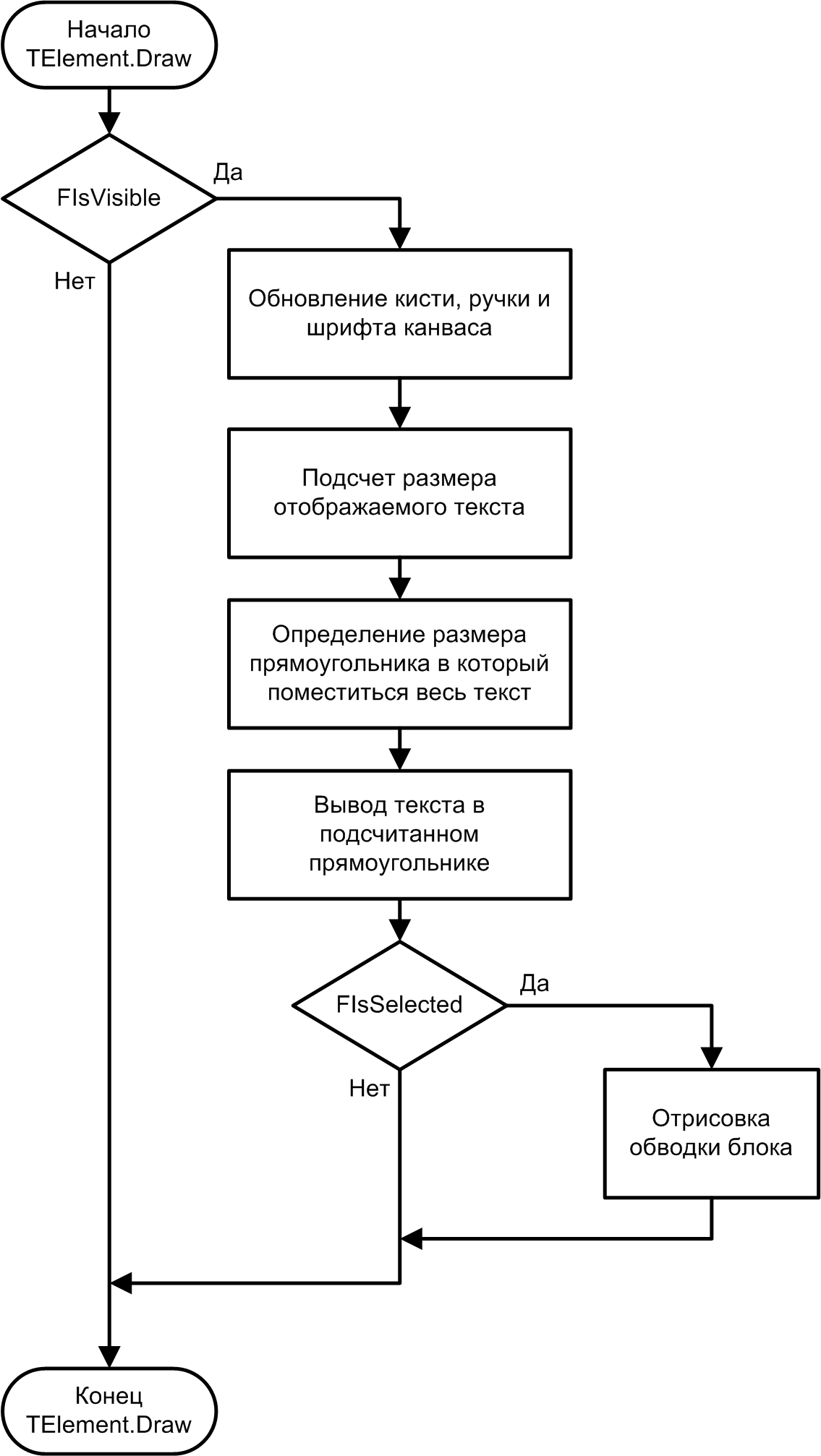


Рисунок 3.2 – Блок-схема отрисовки объекта класса TElement

### **Описание класса TLine**

В таблице 3.8 приведены поля класса TLine

В таблице 3.9 приведены основные методы класса TLine

Таблица 3.8 – Поля класса TLine

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| FText | Текст, отображаемый на линии | String |
| FIsSelected | Флаг, означающий выбрана ли линия | Boolean |
| FPen | Ручка, которой отрисовывается линия | TPen |
| FCanvas | Объект класса TCanvas. Нужен для отрисовки блока | TCanvas |
| FStart | Позиция начала линии | TConnector |
| FFinish | Позиция конца линии | TConnector |

Таблица 3.9 – Основные методы класса TLine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| Draw | Отрисовка текста | procedure Draw; | - | - |
| IsInside | Выдает истину, если точка c координатами X, Y находиться внутри блока, иначе ложь | function IsInside(const X, Y: Integer): Boolean; | X | Положение точки по горизонтали |
| Y | Положение точки по вертикали |

### **Описание структуры TConnecor**

В таблице 3.10 приведены поля структуры TConnector

Таблица 3.10 – Поля структуры TConnector

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| BindToElement | Флаг, означающий привязан ли конец линии к блоку | Boolean |
| Element | Если конец линии привязан к блоку, то задает объект класса TElement, к которому привязан | TElement |
| Pos | Если конец линии не привязан к блоку, то задает точку | TPoint |

# **СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА**

Разработка программы выполнялась на основе спецификации функциональных требований.

В программном средстве используются три формы:

1. Главная форма
2. Форма изменения размеров документа
3. Форма окна «About»

## **Разработка главной формы программного средства**

В таблице 4.1 представлены основные поля главной формы.

В таблице 4.2 представлены основные методы, вызываемые при обработке событий.

В таблице 4.3 представлены основные методы главной формы

Таблица 4.1 – Основные поля главной формы

|  |  |  |
| --- | --- | --- |
| **Идентификатор поля** | **Назначение** | **Тип поля** |
| ImageWidth | Ширина документа | Integer |
| ImageHeigth | Высота документа | Integer |
| TextTmp | Временная переменная объекта класса TText. Текст, с которым взаимодействует пользователь | TText |
| ElementTmp | Временная переменная объекта класса TElement. Элемент, с которым взаимодействует пользователь | TElement |
| LineTmp | Временная переменная объекта класса TLine | TLine |
| ConnectorTmp | Временная переменная указатель на структуру TConnector. Указывает на тип соединения конца линии, с которой взаимодействует пользователь | ^TConnector |
| SelectionState | Флаг, указывающий состояние выделения блока, с которым пользователь ведет взаимодействие | Boolean |

Таблица 4.1 – Основные поля главной формы (Продолжение)

|  |  |  |
| --- | --- | --- |
| CurrentTool | Текущий выбранный инструмент | TTools |
| StartPoint | Точка начала нажатия | TPoint |
| Elements | Динамический массив элементов | TVector<TElement> |
| Texts | Динамический массив текстов | TVector<TText> |
| Lines | Динамический массив линий | TVector<TLine> |

Таблица 4.2 – Основные методы-обработчики событий

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| WorkspaceMouseDown | Реакция программы на нажатие клавиши мыши над документом | procedure WorkspaceMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer); | Sender | Объект, который сгенерировал событие |
| Button | Параметр, определяющий какая кнопка мыши была нажата |
| Shift | Параметр определяющий нажатия управляющих клавиш |
| X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |

Таблица 4.2 – Основные методы-обработчики событий (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WorkspaceMouseMove | Реакция программы на движение мыши над документом | procedure WorkspaceMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer); | Sender | Объект, который сгенерировал событие |
| Shift | Параметр определяющий нажатия управляющих клавиш |
| X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |
| WorkspaceMouseUp | Реакция программы на отпуск клавиши мыши над документоом | procedure WorkspaceMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer); | Sender | Объект, который сгенерировал событие |
| Button | Параметр, определяющий какая кнопка мыши была нажата |
| Shift | Параметр определяющий нажатия управляющих клавиш |
| X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |

Таблица 4.2 – Основные методы-обработчики событий (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FormCreate | Реакция программы на создание формы | procedure FormCreate(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| toolButtonClick | Реакция программы на нажатие на инструмент | procedure toolButtonClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| ImageResolutionExecute | Реакция программы на нажатие пункта меню «Image-Resolution» | procedure ImageResolutionExecute(Sender: TObject); | Sender | Объект, который сгенерировал событие |

Таблица 4.3 – Основные методы формы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| IsClickedElements | Возвращает элемент, на который было произведено нажатие, если такого нет, то возвращает nil | function IsClickedElements(const X, Y: Integer): TElement; | X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |

Таблица 4.3 – Основные методы формы (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IsClickedLines | Возвращает линию, на которую было произведено нажатие, если такого нет, то возвращает nil | function IsClickedLines(const X, Y: Integer): TLine; | X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |
| IsClickedTexts | Возвращает текст, на который было произведено нажатие, если такого нет, то возвращает nil | function IsClickedTexts(const X, Y: Integer): TText; | X | Положение курсора по горизонтали |
| Y | Положение курсора по вертикали |
| ReDraw | Процедура перерисовки документа | procedure ReDraw; | - | - |
| ClearWorkspace | Процедура очистки документа | procedure ClearWorkspace; | - | - |
| ShowPanel | Процедура, обновляющая значения боковой панели с параметрами выделенных элементов | procedure ShowPanel; | - | - |
| UpdateResolution | Процедура, обновляющая размер документа | procedure UpdateResolution; | - | - |
| SelectAll | Процедура выделения всех объектов рабочей области | procedure SelectAll; | - | - |

Таблица 4.3 – Основные методы формы (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DeselectAll | Процедура снятия выделения со всех объектов рабочей области | procedure DeselectAll; | - | - |

## **Разработка модального окна изменения параметров документа**

В таблице 4.4 приведены основные методы, вызываемые при обработке событий.

В таблице 4.5 приведены основные методы формы модального окна.

Таблица 4.4 – Основные методы-обработчики событий модального окна

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| btnCancelClick | Реакция программы на нажатие клавиши мыши над документом | procedure btnOkClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| btnCancelClick | Реакция программы на движение мыши над документом | procedure btnCancelClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |

Таблица 4.5 – Основные методы формы модального окна

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор метода** | **Назначение** | **Заголовок метода** | **Идентификатор параметра** | **Назначение параметра** |
| ChangeResolution | Возвращает результат выполнения модального окна | function ChangeResolution(const AWidth, AHeight: Integer): TModalResult; | AWidth | Текущее значение ширины документа |
| AHeigth | Текущее значение высоты документа |

# **ТЕСТИРОВАНИЕ ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Функциональные тесты, проведенные над программой представлены в таблице 5.1

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Создание блока с формой прямоугольник | 1. Выбрать инструмент на панели 2. Нажать на свободную область документа | В месте нажатия, появляется блок с формой прямоугольника | Тест пройден |
| 2 | Изменение текста блока | 1. Выделить блок, нажав правой кнопкой мыши по нему инструментом мышь 2. Изменить текст на панели справа | В центре выделенного блока появляется печатаемый текст | Тест пройден |
| 3 | Изменение цвета фона, ширины линии и цвета линии | 1. Выделить блок, нажав правой кнопкой мыши по нему инструментом мышь 2. Изменить необходимые параметры на панели справа | Все параметры изменяются у выделенного блока в соответствии с выбранными значениями | Тест пройден |

Таблица 5.1 – Результаты функционального тестирования (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Изменение шрифта у блока | 1. Выделить блок, нажав правой кнопкой мыши по нему инструментом мышь 2. Вызвать диалоговое окно справа на панели | Шрифт у выделенных блоков изменился в соответствии с выбранным | Тест пройден |
| 5 | Привязка линии к блокам | 1. Создать еще блок, с любой формой 2. Взять инструмент линия 3. Зажать левую кнопку мыши на первом блоке 4. Протянуть курсор до второго блока и отпустить | Между блоками образовалась линия связи. | Тест пройден |
| 6 | Проверка на привязку линии при передвижении блока | 1. Выбрать инструмент мышь 2. Перемещать блок, к которому привязана линия | Линия передвигается вместе с блоком | Тест пройден |

Таблица 5.1 – Результаты функционального тестирования (Продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | Проверка на правильность сохранения данных в специфичный файл | 1. Выбрать пункт меню «File – Save as» 2. Ввести имя файла для сохранения 3. Закрыть программу 4. Открыть программу 5. Открыть файл | Вся проделанная нами работа ранее сохранена и ничего не было потеряно | Тест пройден |
| 8 | Проверка на увеличение элементов, при увеличении рабочей области | 1. Добавить несколько блоков на рабочее пространство 2. Выбрать пункт меню «Image - Resolution» 3. Изменить параметры |  |  |

# **РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

Для того, чтобы начать использовать программное средство, необходимо запустить файл TreeMaker.exe. После открытия программы появится окно показанное на рисунке 6.1

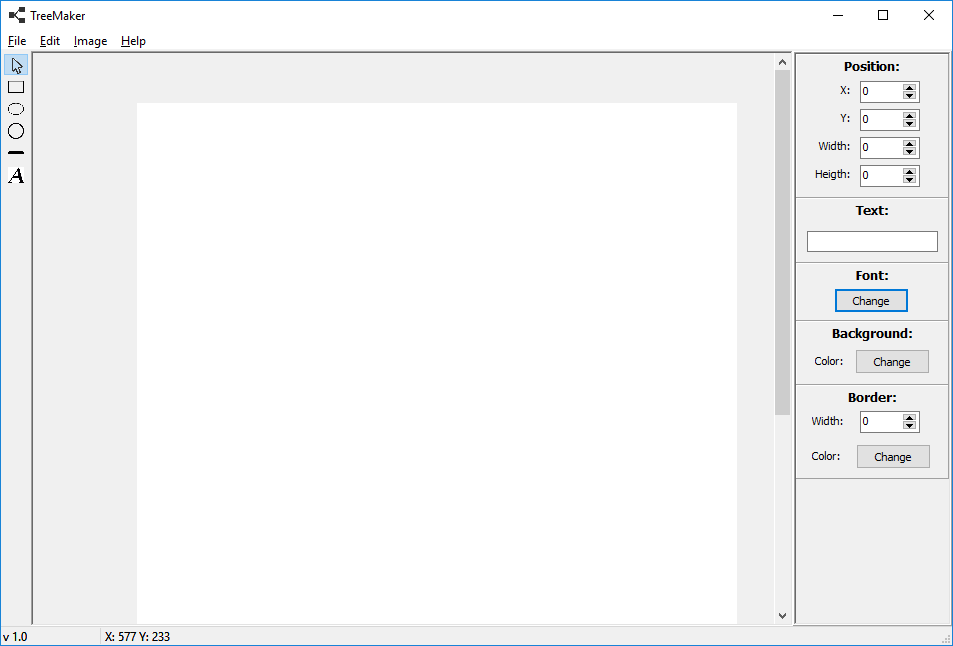


Рисунок 6.1 – Главное окно программы

Для открытия файла с расширением .tmf пользователю необходимо выбрать пункт меню «File - Open» или же нажать сочетание клавиш Ctrl + O, после чего откроется диалоговое окно, выбора файла. (рисунок 6.2)

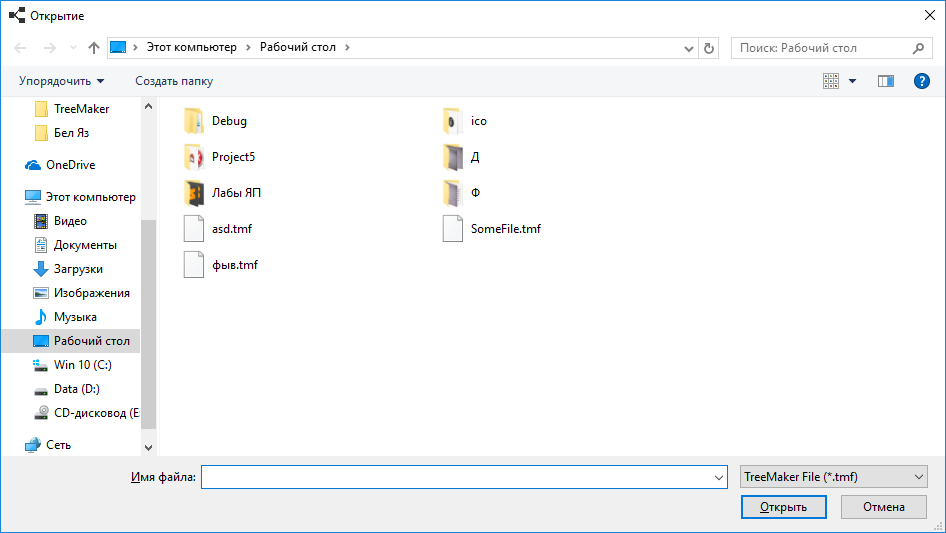


Рисунок 6.2 – Диалоговое окно открытия файла

После выбора файла, у нас отобразится его содержимое в рабочей области (рисунок 6.3).

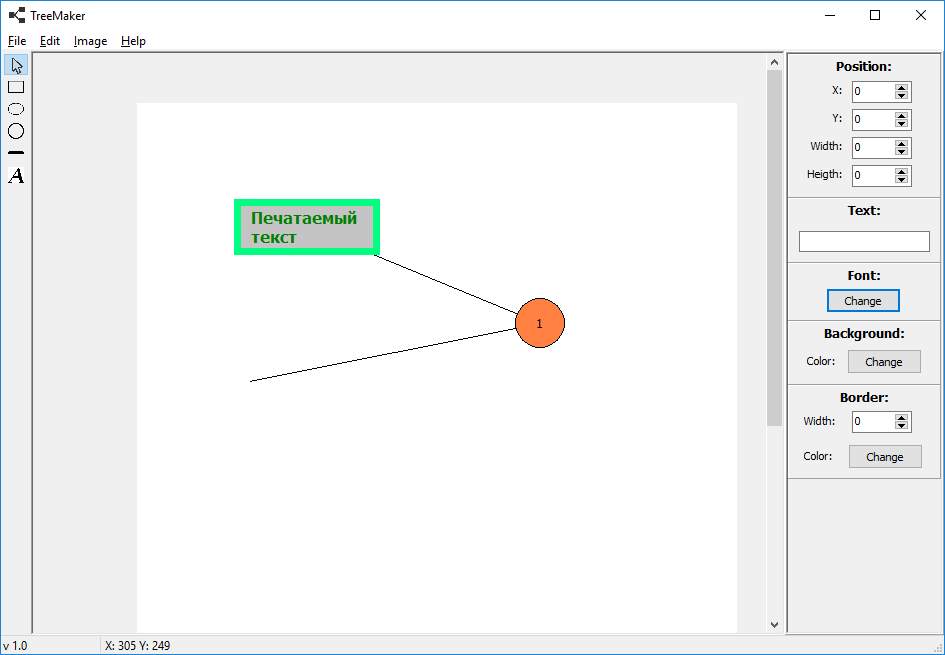


Рисунок 6.3 – Главное окно программы после открытия файла

Слева находиться панель инструментов (рисунок 6.4)

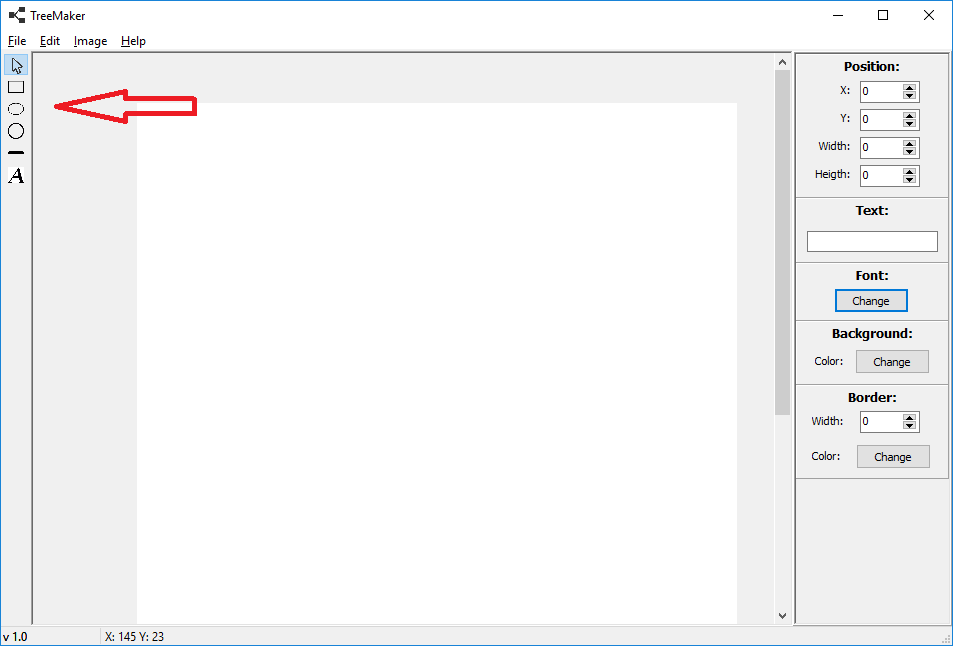


Рисунок 6.4 – Панель инструментов

Справа находиться все свойства блоков и линий (рисунок 6.5)

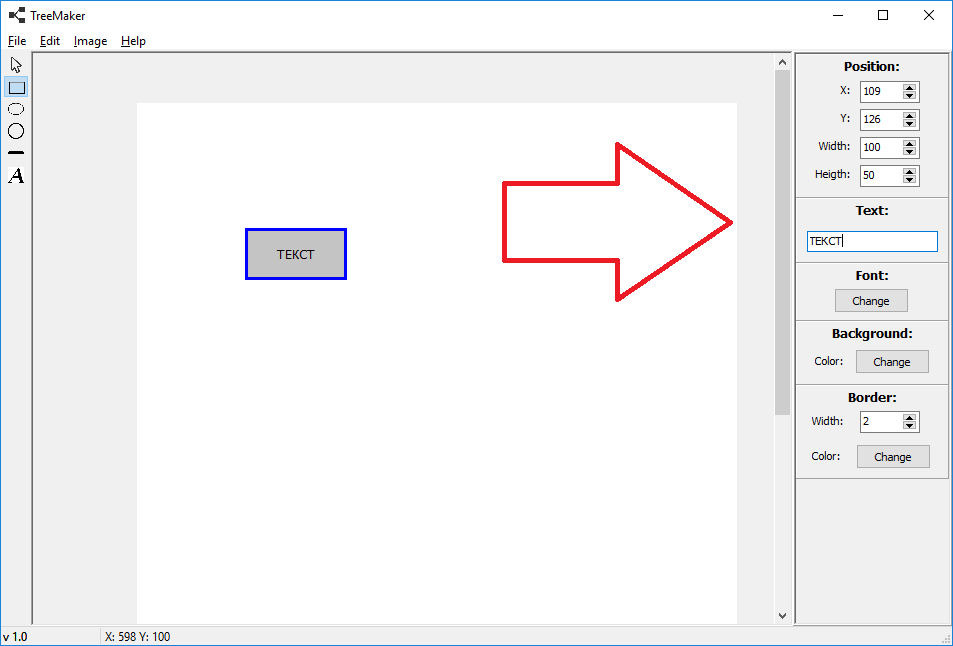


Рисунок 6.5 – Панель свойств

Для изменения размера документа нужно выбрать пункт меню Image – Resolution, после чего откроется модальное окно с выбором размеров документа (рисунок 6.6).

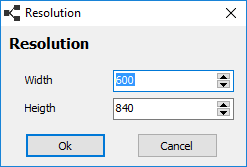


Рисунок 6.6 – Окно изменения размеров документа

После того, как мы поработали, надо сохранить результат. Для этого следует перейти по пункту меню File – Save as, в результате чего откроется окно с выбором расширения и имени сохраняемого файла (рисунок 6.7).

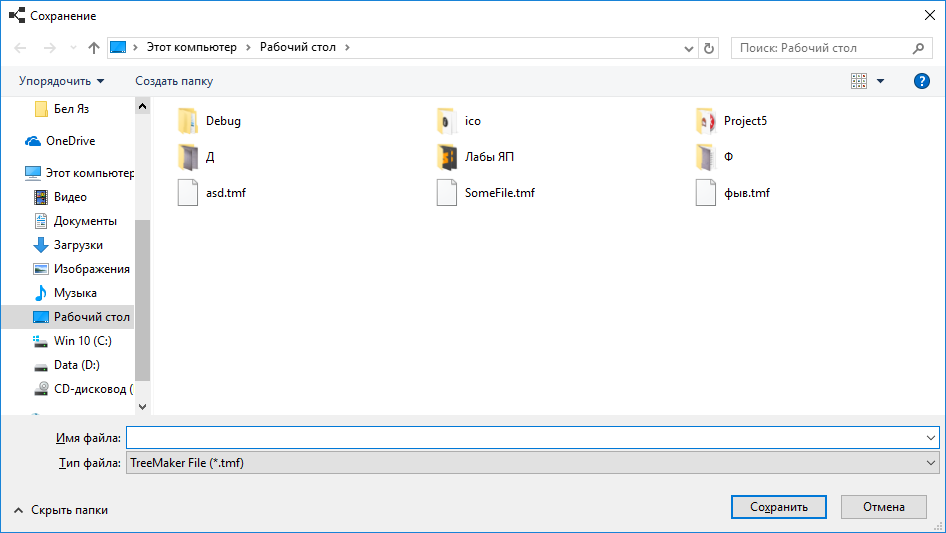


Рисунок 6.7 – Окно сохранения файла

# **ЗАКЛЮЧЕНИЕ**

В результате работы над курсовым проектом было создано исправно работающее программное средство для построения и отображения древовидных структур и сете, которое может пригодиться людям, чья жизнь связана с информатикой.

Разработка приложения включала в себя решение множества задач и проблем, как итог было изучено большое количество приложений для построения различных схем, проведен их анализ, и сформированы оптимальные требования для приложений подобного рода.

Далее были изучены некоторые возможности создания приложений в Delphi и формирование конкретных функциональных требований к программе на основе возможностей языка

Затем были разработаны структуры данных, разработана примерная архитектура приложения. Далее были детализированы все функции.

Программа была отлажен и протестирована сначала разработчиком, а затем несколько раз обычным пользователем. После испытаний были внесены корректировки в интерфейсе, работе некоторых функций.

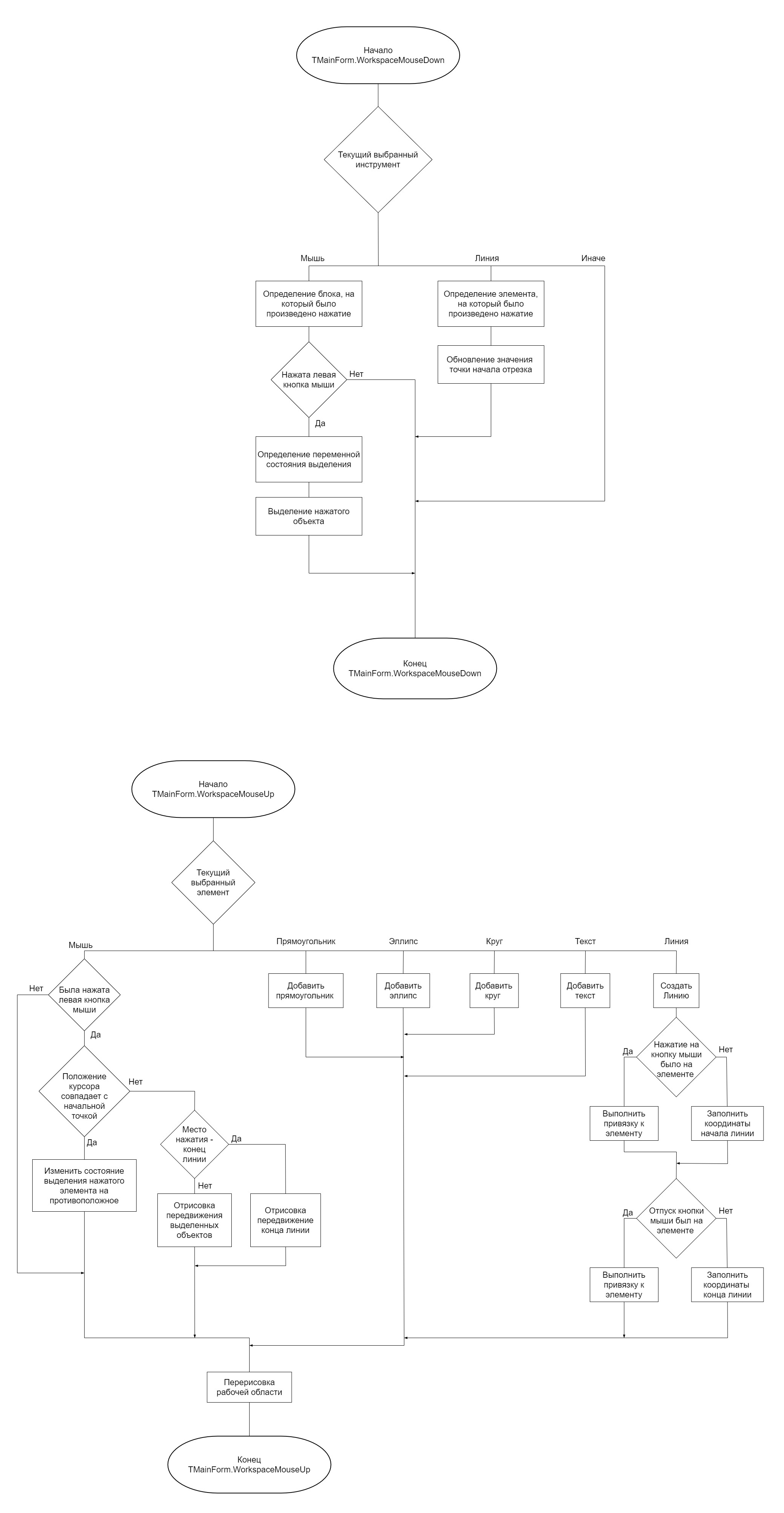
Пройдя все вышеперечисленные этапы на выходе получилось корректно работающее программное средство для построения и отображения древовидных структур и сетей.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Древовидная структура [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Древовидная_структура>
2. Microsoft Visio [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Microsoft_Visio>
3. Microsoft Visio [Электронный ресурс] – Режим доступа: <http://visio.microsoft.com/>
4. EDraw [Электронный ресурс] – Режим доступа: <https://www.edrawsoft.com/>
5. GDE GoVisual Diagram Editor[Электронный ресурс] – Режим доступа: <https://download.cnet.com/GDE-GoVisual-Diagram-Editor/3000-2075_4-10305236.html>
6. Основы Delphi [Электронный ресурс] – Режим доступа: <http://www.delphibasics.ru/>
7. Embarcadero/IDERA Product Documentation [Электронный ресурс] – Режим доступа: <http://docs.embarcadero.com/>
8. Delphi Russian Knowledge Base [Электронный ресурс] – Режим доступа: <https://drkb.ru/>
9. ГОСТ 19.701–90 (ИСО 5807–85) [Текст]. – Единая система программной документации: Сб. ГОСТов. - М.: Стандартинформ, 2005 с.

**ПРИЛОЖЕНИЕ А**

**Обобщенная схема программы**



**ПРИЛОЖЕНИЕ Б  
Текст главного программного модуля**

unit MainWindow;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Vcl.ExtCtrls, Vcl.StdCtrls,

Database, Vcl.ToolWin, Vcl.ComCtrls, System.Actions, Vcl.ActnList,

Vcl.StdActns, System.ImageList, Vcl.ImgList, Vector, Vcl.ActnMan,

Vcl.ActnColorMaps, Vcl.Samples.Spin;

const

// Константы состояний контролов

ST\_ALL\_OK = 0;

ST\_DIFF\_VALUES = 1;

ST\_UNDEFINED = 2;

ST\_ERROR = 3;

ST\_UPDATING = 4;

type

// Инструменты

TTools = (toolMouse, toolRectangle, toolEllipse, toolCircle, toolLine,

toolText);

TMainForm = class(TForm)

Menu: TMainMenu;

miFile: TMenuItem;

MiOpen: TMenuItem;

miSave: TMenuItem;

miEdit: TMenuItem;

miHelp: TMenuItem;

miAbout: TMenuItem;

miUndo: TMenuItem;

ToolbarPanel: TPanel;

miView: TMenuItem;

Properties: TPanel;

StatusBar: TStatusBar;

ImageList: TImageList;

ActionList: TActionList;

actFileOpen: TFileOpen;

actEditCut: TEditCut;

actEditCopy: TEditCopy;

actEditPaste: TEditPaste;

actEditSelectAll: TEditSelectAll;

actEditUndo: TEditUndo;

actEditDelete: TEditDelete;

actFileSaveAs: TFileSaveAs;

actFileExit: TFileExit;

miExit: TMenuItem;

miCut: TMenuItem;

miCopy: TMenuItem;

miPaste: TMenuItem;

miSelectAll: TMenuItem;

miDelete: TMenuItem;

ToolBar: TToolBar;

tlMouse: TToolButton;

tlRectangle: TToolButton;

tlEllipse: TToolButton;

tlCircle: TToolButton;

tlLine: TToolButton;

MainProperties: TPanel;

lblPosition: TLabel;

lblX: TLabel;

lblY: TLabel;

ScrollBox: TScrollBox;

Workspace: TImage;

ScrollBoxProperties: TScrollBox;

lblWidth: TLabel;

lblHeigth: TLabel;

TextProperties: TPanel;

FontDialog: TFontDialog;

lblFont: TLabel;

btnFont: TButton;

BorderPanel: TPanel;

lblBorder: TLabel;

lblColor: TLabel;

lblBorderWidth: TLabel;

BackGroundPanel: TPanel;

lblBackground: TLabel;

btnBackgroundColor: TButton;

lblBackgroundColor: TLabel;

ColorDialog: TColorDialog;

spinX: TSpinEdit;

spinY: TSpinEdit;

spinHeigth: TSpinEdit;

spinWidth: TSpinEdit;

PositionPanelCaptions: TPanel;

PositionSpinPanel: TPanel;

pnlText: TPanel;

lblText: TLabel;

edtText: TEdit;

HelpAbout: TAction;

miImage: TMenuItem;

miResolution: TMenuItem;

ImageResolution: TAction;

tlText: TToolButton;

btnBorderColor: TButton;

spinBorderWidth: TSpinEdit;

tbScale: TTrackBar;

pnlBackground: TPanel;

Zoom: TPanel;

lblScale: TLabel;

procedure FormCreate(Sender: TObject);

procedure toolButtonClick(Sender: TObject);

procedure WorkspaceMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

procedure WorkspaceMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure WorkspaceMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure actEditSelectAllExecute(Sender: TObject);

procedure actEditDeleteExecute(Sender: TObject);

procedure spinMainPropertiesChange(Sender: TObject);

procedure ImageResolutionExecute(Sender: TObject);

procedure actFileSaveAsAccept(Sender: TObject);

procedure actFileOpenAccept(Sender: TObject);

procedure edtTextChange(Sender: TObject);

procedure btnBackgroundColorClick(Sender: TObject);

procedure btnFontClick(Sender: TObject);

procedure btnBorderColorClick(Sender: TObject);

procedure HelpAboutExecute(Sender: TObject);

procedure spinBorderWidthChange(Sender: TObject);

procedure ActionListUpdate(Action: TBasicAction; var Handled: Boolean);

procedure actEditDeleteUpdate(Sender: TObject);

procedure tbScaleChange(Sender: TObject);

procedure ScrollBoxMouseWheel(Sender: TObject; Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);

procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);

private

ImageWidth, ImageHeigth: Integer; // Размеры документа

FScale: Real;

FChanges: Boolean;

TextTmp: TText;

ElementTmp: TElement;

LineTmp: TLine;

ConnectorTmp: ^TConnector;

SelectionState: Boolean;

CurrentTool: TTools; // Текущий инструмент

StartPoint: TPoint; // Начальная точка (Точка нажима клавиши мыши)

Elements: TVector<TElement>; // Динамический массив элементов

Texts: TVector<TText>; // Динамеческий массив текстов

Lines: TVector<TLine>; // Динамический массив линий

function IsClickedElements(const X, Y: Integer): TElement;

function IsClickedLines(const X, Y: Integer): TLine;

function IsClickedTexts(const X, Y: Integer): TText;

procedure SetDefaultsElement(Element: TElement);

procedure UpdateScale(Delta: Integer);

// Добавление элементов

procedure AddRectangle(const X, Y: Integer);

procedure AddEllipse(const X, Y: Integer);

procedure AddCircle(const X, Y: Integer);

procedure AddText(const X, Y: Integer);

// Выделить все

procedure SelectAll;

// Снять выделение

procedure DeselectAll;

procedure ReDraw; // Перерисовка

procedure ClearWorkspace; // Очистка рабочей области

procedure ShowPanel; // Обновление значений контролов

procedure ClearMainPropertiesPanel;

procedure FillMainPropertiesPanel(const X, Y, Width, Heigth: Integer);

procedure FillTextPanel(const Text: String);

procedure FillBorderPanel(const Value: Integer);

procedure UpdateResolution(PosX: Integer = -1; PosY: Integer = -1);

procedure ElementOnMove(Sender: TObject; const X, Y: Integer);

procedure TextOnMove(Sender: TObject; const X, Y: Integer);

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

{$R \*.dfm}

uses

uModalResolution,

Vcl.Imaging.jpeg,

Vcl.Imaging.pngimage, uAbout;

// Передвижение элемента

procedure TMainForm.ElementOnMove(Sender: TObject; const X, Y: Integer);

var

Tmp: TElement;

begin

Tmp := TElement.Create(Sender as TElement);

// Tmp.Assign((Sender as TElement), FScale);

Tmp.Selected := False;

Tmp.SetPosition(Tmp.Left + X, Tmp.Top + Y);

Tmp.Draw(FScale);

Tmp.Free;

end;

// Передвижение текста

procedure TMainForm.TextOnMove(Sender: TObject; const X, Y: Integer);

var

Tmp: TText;

begin

Tmp := TText.Create(Sender as TText);

Tmp.SetPosition(Tmp.Left + X, Tmp.Top + Y);

// Tmp.Selected := False;

Tmp.Draw(FScale);

Tmp.Free;

end;

// Обновление функций.

procedure TMainForm.ActionListUpdate(Action: TBasicAction;

var Handled: Boolean);

begin

actEditSelectAll.Enabled := True;

actEditDelete.Enabled := True;

end;

// Добавление элементов

procedure TMainForm.AddCircle(const X, Y: Integer);

var

Element: TElement;

begin

DeselectAll;

Element := TElement.Create;

Element.Shape := stCircle;

SetDefaultsElement(Element);

Element.SetSize(50, 50);

Element.SetPosition(X - Element.Width shr 1, Y - Element.Heigth shr 1);

Element.Draw;

Elements.PushBack(Element);

end;

procedure TMainForm.AddEllipse(const X, Y: Integer);

var

Element: TElement;

begin

DeselectAll;

Element := TElement.Create;

Element.Shape := stEllipse;

SetDefaultsElement(Element);

Element.SetPosition(X - Element.Width shr 1, Y - Element.Heigth shr 1);

Element.Draw;

Elements.PushBack(Element);

end;

procedure TMainForm.AddRectangle(const X, Y: Integer);

var

Element: TElement;

begin

DeselectAll;

Element := TElement.Create;

Element.Shape := stRectangle;

SetDefaultsElement(Element);

Element.SetPosition(X - Element.Width shr 1, Y - Element.Heigth shr 1);

Element.Draw;

Elements.PushBack(Element);

end;

procedure TMainForm.SetDefaultsElement(Element: TElement);

begin

Element.Canvas := Workspace.Canvas;

Element.SetSize(100, 50);

Element.Font.Size := 10;

Element.Caption := '';

Element.Brush.Color := $C4C4C4;

Element.Pen.Width := 2;

Element.Selected := True;

end;

procedure TMainForm.AddText(const X, Y: Integer);

var

Tmp: TText;

begin

DeselectAll;

Tmp := TText.Create;

Tmp.Selected := True;

Tmp.SetSize(100, 50);

Tmp.SetPosition(X - Tmp.Width shr 1, Y - Tmp.Heigth shr 1);

Tmp.Canvas := Workspace.Canvas;

Tmp.Caption := '';

Texts.PushBack(Tmp);

end;

procedure TMainForm.btnBackgroundColorClick(Sender: TObject);

var

I: Integer;

begin

if ColorDialog.Execute then

begin

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

Brush.Color := ColorDialog.Color;

ClearWorkspace;

ReDraw;

end;

end;

procedure TMainForm.btnBorderColorClick(Sender: TObject);

var

I: Integer;

begin

if ColorDialog.Execute then

begin

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

Pen.Color := ColorDialog.Color;

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

Pen.Color := ColorDialog.Color;

end;

end;

procedure TMainForm.btnFontClick(Sender: TObject);

var

I: Integer;

TmpFont: TFont;

State: Integer;

begin

State := ST\_UNDEFINED;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

if State = ST\_UNDEFINED then

TmpFont := Font

else if not TmpFont.Equals(Font) then

begin

State := ST\_DIFF\_VALUES;

Break;

end;

if State <> ST\_DIFF\_VALUES then

for I := 0 to Texts.Size - 1 do

with Texts.At[I] do

if State = ST\_UNDEFINED then

TmpFont := Font

else if not TmpFont.Equals(Font) then

begin

State := ST\_DIFF\_VALUES;

Break;

end;

if State <> ST\_DIFF\_VALUES then

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if State = ST\_UNDEFINED then

TmpFont := Font

else if not TmpFont.Equals(Font) then

begin

State := ST\_DIFF\_VALUES;

Break;

end;

if Assigned(TmpFont) then

FontDialog.Font.Assign(TmpFont);

if FontDialog.Execute then

begin

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

Font.Assign(FontDialog.Font);

for I := 0 to Texts.Size - 1 do

with Texts.At[I] do

if Selected then

Font.Assign(FontDialog.Font);

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

Text.Font.Assign(FontDialog.Font);

ClearWorkspace;

ReDraw;

end;

end;

procedure TMainForm.ClearMainPropertiesPanel;

begin

spinX.Value := -2;

spinY.Value := -2;

spinWidth.Value := -2;

spinHeigth.Value := -2;

end;

procedure TMainForm.ClearWorkspace;

begin

Workspace.Canvas.Brush.Style := bsSolid;

Workspace.Canvas.Brush.Color := clWhite;

Workspace.Canvas.FillRect(TRect.Create(0, 0, Workspace.Width,

Workspace.Height));

end;

procedure TMainForm.DeselectAll;

var

I: Integer;

begin

for I := 0 to Elements.Size - 1 do

Elements.At[I].Selected := False;

for I := 0 to Lines.Size - 1 do

Lines.At[I].Selected := False;

for I := 0 to Texts.Size - 1 do

Texts.At[I].Selected := False;

end;

procedure TMainForm.actEditDeleteExecute(Sender: TObject);

var

I: Integer;

begin

I := 0;

while I < Lines.Size do

begin

if Lines.At[I].Selected then

begin

Lines.At[I].Free;

Lines.Erase(I);

FChanges := True;

end

else

begin

with Lines.At[I] do

begin

if FStart.BindToElement and FStart.Element.Selected then

FStart.OffBind;

if FFinish.BindToElement and FFinish.Element.Selected then

FFinish.OffBind;

end;

Inc(I);

end;

end;

I := 0;

while I < Elements.Size do

begin

if Elements.At[I].Selected then

begin

Elements.At[I].Free;

Elements.Erase(I);

FChanges := True;

end

else

Inc(I);

end;

I := 0;

while I < Texts.Size do

begin

if Texts.At[I].Selected then

begin

Texts.At[I].Free;

Texts.Erase(I);

FChanges := True;

end

else

Inc(I);

end;

ClearWorkspace;

ReDraw;

end;

procedure TMainForm.actEditDeleteUpdate(Sender: TObject);

begin

actEditDelete.Enabled := True;

end;

procedure TMainForm.actEditSelectAllExecute(Sender: TObject);

begin

SelectAll;

ClearWorkspace;

ReDraw;

end;

procedure TMainForm.edtTextChange(Sender: TObject);

var

Tmp: TEdit;

I: Integer;

begin

Tmp := Sender as TEdit;

if Tmp.Tag and ST\_UPDATING > 0 then

exit;

if Tmp.Tag = ST\_ERROR then

exit;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

Caption := Tmp.Text;

for I := 0 to Texts.Size - 1 do

with Texts.At[I] do

if Selected then

Caption := Tmp.Text;

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

Text.Caption := Tmp.Text;

ClearWorkspace;

ReDraw;

end;

function ExtendWithExt(const FileName, Extension: TFileName): TFileName;

begin

if Length(FileName) < Length(Extension) then

Result := FileName + Extension

else

begin

if Copy(FileName, Length(FileName) - Length(Extension) + 1,

Length(Extension)) <> Extension then

Result := FileName + Extension

else

Result := FileName;

end;

end;

procedure TMainForm.actFileOpenAccept(Sender: TObject);

procedure OpenBMP(const FileName: TFileName);

begin

Workspace.Picture.LoadFromFile(FileName);

end;

procedure OpenJPEG(const FileName: TFileName);

begin

Workspace.Picture.Graphic.LoadFromFile(FileName);

end;

procedure OpenPNG(const FileName: TFileName);

begin

Workspace.Picture.Graphic.LoadFromFile(FileName);

end;

procedure OpenTMF(const FileName: TFileName);

function ElementToInt(Element: TElement): Integer;

var

A: Integer absolute Element;

begin

Result := A;

end;

var

fsFile: TFileStream;

N, PredSize, I: Integer;

begin

fsFile := TFileStream.Create(FileName, fmOpenRead);

// ImageSize

fsFile.Read(ImageWidth, SizeOf(ImageWidth));

fsFile.Read(ImageHeigth, SizeOf(ImageHeigth));

UpdateResolution;

// Elements

fsFile.Read(N, SizeOf(N));

PredSize := Elements.Size;

Elements.Reserve(N);

for I := PredSize to N - 1 do

begin

Elements.At[I] := TElement.Create;

Elements.At[I].Canvas := Workspace.Canvas;

end;

for I := 0 to N - 1 do

Elements.At[I].ReadFromFileStream(fsFile);

// Lines

fsFile.Read(N, SizeOf(N));

PredSize := Lines.Size;

Lines.Reserve(N);

for I := PredSize to N - 1 do

begin

Lines.At[I] := TLine.Create;

Lines.At[I].Canvas := Workspace.Canvas;

end;

for I := 0 to N - 1 do

with Lines.At[I] do

begin

ReadFromFileStream(fsFile);

if Start.BindToElement then

FStart.Element := Elements.At[ElementToInt(Start.Element)];

if Finish.BindToElement then

FFinish.Element := Elements.At[ElementToInt(Finish.Element)];

end;

// Texts

fsFile.Read(N, SizeOf(N));

PredSize := Texts.Size;

Texts.Reserve(N);

for I := PredSize to N - 1 do

begin

Texts.At[I] := TText.Create;

Texts.At[I].Canvas := Workspace.Canvas;

end;

for I := 0 to N - 1 do

Texts.At[I].ReadFromFileStream(fsFile);

fsFile.Free;

end;

var

OpenDialog: TOpenDialog;

Res: Integer;

Flag: Boolean;

begin

Flag := True;

if FChanges then

begin

Res := MessageDlg('Есть несохраненные изменения. Вы хотите сохранить?',

mtConfirmation, mbYesNoCancel, 0);

case Res of

mrYes:

begin

actFileSaveAs.Execute;

Flag := actFileSaveAs.ExecuteResult;

end;

mrNo:

Flag := True;

mrCancel:

Flag := False;

end;

end;

if Flag then

begin

OpenDialog := (Sender as TFileOpen).Dialog;

case OpenDialog.FilterIndex of

// 1: OpenBMP(ExtendWithExt(OpenDialog.FileName, '.bmp'));

// 2: OpenJPEG(ExtendWithExt(OpenDialog.FileName, '.jpg'));

// 3: OpenPNG(ExtendWithExt(OpenDialog.FileName, '.png'));

1:

OpenTMF(ExtendWithExt(OpenDialog.FileName, '.tmf'));

end;

FChanges := False;

ClearWorkspace;

ReDraw;

end;

end;

procedure TMainForm.actFileSaveAsAccept(Sender: TObject);

procedure SaveBMP(const FileName: TFileName);

var

Res: Integer;

begin

Res := mrOk;

if FileExists(FileName) then

Res := MessageDlg('Файл существует. Перезаписать?', mtConfirmation,

mbYesNoCancel, 0);

if Res = mrOk then

Workspace.Picture.SaveToFile(FileName);

end;

procedure SaveJPEG(const FileName: TFileName);

var

imJPEG: TJpegImage;

Res: Integer;

begin

Res := mrYes;

if FileExists(FileName) then

Res := MessageDlg('Файл существует. Перезаписать?', mtConfirmation,

mbYesNoCancel, 0);

if Res = mrYes then

begin

imJPEG := TJpegImage.Create;

imJPEG.Assign(Workspace.Picture.Graphic);

imJPEG.SaveToFile(FileName);

end;

end;

procedure SavePNG(const FileName: TFileName);

var

imPNG: TPngImage;

Res: Integer;

begin

Res := mrYes;

if FileExists(FileName) then

Res := MessageDlg('Файл существует. Перезаписать?', mtConfirmation,

mbYesNoCancel, 0);

if Res = mrYes then

begin

imPNG := TPngImage.Create;

imPNG.Assign(Workspace.Picture.Graphic);

imPNG.SaveToFile(FileName);

end;

end;

procedure SaveTMF(const FileName: TFileName);

var

fsFile: TFileStream;

N, I: Integer;

Res: Integer;

begin

Res := mrYes;

if FileExists(FileName) then

Res := MessageDlg('Файл существует. Перезаписать?', mtConfirmation,

mbYesNoCancel, 0);

if Res = mrYes then

begin

fsFile := TFileStream.Create(FileName, fmCreate);

// ImageSize

fsFile.Write(ImageWidth, SizeOf(ImageWidth));

fsFile.Write(ImageHeigth, SizeOf(ImageHeigth));

// Elements

N := Elements.Size;

fsFile.Write(N, SizeOf(N));

for I := 0 to Elements.Size - 1 do

begin

Elements.At[I].Id := I;

Elements.At[I].WriteToFileStream(fsFile);

end;

// Lines

N := Lines.Size;

fsFile.Write(N, SizeOf(N));

for I := 0 to Lines.Size - 1 do

Lines.At[I].WriteToFileStream(fsFile);

// Texts

N := Texts.Size;

fsFile.Write(N, SizeOf(N));

for I := 0 to Texts.Size - 1 do

Texts.At[I].WriteToFileStream(fsFile);

fsFile.Free;

end;

end;

var

SaveDialog: TSaveDialog;

begin

SaveDialog := (Sender as TFileSaveAs).Dialog;

case SaveDialog.FilterIndex of

1:

SaveBMP(ExtendWithExt(SaveDialog.FileName, '.bmp'));

2:

SaveJPEG(ExtendWithExt(SaveDialog.FileName, '.jpg'));

3:

SavePNG(ExtendWithExt(SaveDialog.FileName, '.png'));

4:

SaveTMF(ExtendWithExt(SaveDialog.FileName, '.tmf'));

end;

FChanges := False;

end;

procedure TMainForm.FillBorderPanel(const Value: Integer);

begin

if spinBorderWidth.Tag = ST\_DIFF\_VALUES or ST\_UPDATING then

exit;

if spinBorderWidth.Tag = ST\_ERROR or ST\_UPDATING then

exit;

if (spinBorderWidth.Tag = ST\_UNDEFINED or ST\_UPDATING) then

begin

spinBorderWidth.Value := Value;

spinBorderWidth.Tag := ST\_ALL\_OK or ST\_UPDATING;

end

else if (spinBorderWidth.Value <> Value) then

begin

spinBorderWidth.Tag := ST\_DIFF\_VALUES or ST\_UPDATING;

spinBorderWidth.Value := 0;

end;

end;

procedure TMainForm.FillMainPropertiesPanel(const X, Y, Width, Heigth: Integer);

procedure UpdateSpin(Spin: TSpinEdit; Value: Integer);

begin

if Spin.Tag = ST\_DIFF\_VALUES or ST\_UPDATING then

exit;

if Spin.Tag = ST\_ERROR or ST\_UPDATING then

exit;

if (Spin.Tag = ST\_UNDEFINED or ST\_UPDATING) then

begin

Spin.Value := Value;

Spin.Tag := ST\_ALL\_OK or ST\_UPDATING;

end

else if (Spin.Value <> Value) then

begin

Spin.Tag := ST\_DIFF\_VALUES or ST\_UPDATING;

Spin.Value := 0;

end;

end;

begin

UpdateSpin(spinX, X);

UpdateSpin(spinY, Y);

UpdateSpin(spinWidth, Width);

UpdateSpin(spinHeigth, Heigth);

end;

procedure TMainForm.FillTextPanel(const Text: String);

begin

if edtText.Tag = ST\_ERROR or ST\_UPDATING then

exit;

if edtText.Tag = ST\_DIFF\_VALUES or ST\_UPDATING then

exit;

if (edtText.Tag and ST\_UNDEFINED > 0) or (edtText.Text = Text) then

begin

edtText.Text := Text;

edtText.Tag := ST\_ALL\_OK or ST\_UPDATING;

end

else

begin

edtText.Tag := ST\_DIFF\_VALUES or ST\_UPDATING;

edtText.Text := '';

end;

end;

procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);

var

Res: Integer;

begin

if FChanges then

begin

Res := MessageDlg('Есть несохраненные изменения. Вы хотите сохранить?',

mtConfirmation, mbYesNoCancel, 0);

case Res of

mrYes:

begin

actFileSaveAs.Execute;

CanClose := actFileSaveAs.ExecuteResult;

end;

mrNo:

CanClose := True;

mrCancel:

CanClose := False;

end;

end;

end;

procedure TMainForm.FormCreate(Sender: TObject);

begin

CurrentTool := toolMouse;

Elements := TVector<TElement>.Create;

Texts := TVector<TText>.Create;

Lines := TVector<TLine>.Create;

StatusBar.Panels[0].Text := 'v 2.0';

ImageWidth := 600;

ImageHeigth := 840;

tbScale.Position := 100;

FScale := 1.0;

UpdateResolution;

ClearWorkspace;

ShowPanel;

FChanges := False;

with ScrollBox.HorzScrollBar do

Position := Round(0.5 \* Range) - ScrollBox.Width shr 1;

with ScrollBox.VertScrollBar do

Position := Round(0.5 \* Range) - ScrollBox.Height shr 1;

end;

procedure TMainForm.HelpAboutExecute(Sender: TObject);

begin

frmAbout.Show;

end;

procedure TMainForm.ImageResolutionExecute(Sender: TObject);

var

Res: TModalResult;

I: Integer;

ScaleX, ScaleY: Real;

begin

Res := formResolution.ChangeResolution(ImageWidth, ImageHeigth);

if Res = mrOk then

begin

ScaleX := ImageWidth;

ImageWidth := formResolution.Width;

ScaleX := ImageWidth / ScaleX;

ScaleY := ImageHeigth;

ImageHeigth := formResolution.Heigth;

ScaleY := ImageHeigth / ScaleY;

UpdateResolution;

for I := 0 to Elements.Size - 1 do

Elements.At[I].Assign(Elements.At[I], ScaleX, ScaleY);

for I := 0 to Texts.Size - 1 do

Texts.At[I].Assign(Texts.At[I], ScaleX, ScaleY);

for I := 0 to Lines.Size - 1 do

Lines.At[I].Assign(Lines.At[I], ScaleX, ScaleY);

ClearWorkspace;

ReDraw;

FChanges := True;

end;

end;

function TMainForm.IsClickedElements(const X, Y: Integer): TElement;

var

I: Integer;

begin

Result := Nil;

for I := Elements.Size - 1 downto 0 do

if Elements.At[I].IsInside(X, Y) then

begin

Result := Elements.At[I];

Break;

end;

end;

function TMainForm.IsClickedLines(const X, Y: Integer): TLine;

var

I: Integer;

begin

Result := Nil;

for I := Lines.Size - 1 downto 0 do

if Lines.At[I].IsInside(X, Y) then

begin

Result := Lines.At[I];

Break;

end;

end;

function TMainForm.IsClickedTexts(const X, Y: Integer): TText;

var

I: Integer;

begin

Result := Nil;

for I := Texts.Size - 1 downto 0 do

if Texts.At[I].IsInside(X, Y) then

begin

Result := Texts.At[I];

Break;

end;

end;

procedure TMainForm.ReDraw;

var

I: Integer;

begin

for I := 0 to Lines.Size - 1 do

if not Lines.At[I].Selected then

Lines.At[I].Draw(FScale);

for I := 0 to Elements.Size - 1 do

Elements.At[I].Draw(FScale);

for I := 0 to Texts.Size - 1 do

Texts.At[I].Draw(FScale);

for I := 0 to Lines.Size - 1 do

if Lines.At[I].Selected then

Lines.At[I].Draw(FScale);

end;

procedure TMainForm.ScrollBoxMouseWheel(Sender: TObject; Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);

begin

MousePos := ScreenToClient(Mouse.CursorPos);

MousePos.X := MousePos.X - (Sender as TScrollBox).Left;

MousePos.Y := MousePos.Y - (Sender as TScrollBox).Top;

if ssCtrl in Shift then

begin

UpdateScale(WheelDelta);

UpdateResolution(MousePos.X, MousePos.Y);

ClearWorkspace;

ReDraw;

Handled := True;

end;

end;

procedure TMainForm.SelectAll;

var

I: Integer;

begin

for I := 0 to Elements.Size - 1 do

Elements.At[I].Selected := True;;

for I := 0 to Lines.Size - 1 do

Lines.At[I].Selected := True;

for I := 0 to Texts.Size - 1 do

Texts.At[I].Selected := True;

end;

procedure TMainForm.ShowPanel;

procedure UpdateTWinControl(Sender: TWinControl);

begin

Sender.Tag := Sender.Tag xor ST\_UPDATING;

Sender.Enabled := Sender.Tag <> ST\_ERROR;

end;

procedure UpdateSpins;

begin

UpdateTWinControl(spinX);

UpdateTWinControl(spinY);

UpdateTWinControl(spinWidth);

UpdateTWinControl(spinHeigth);

end;

var

I: Integer;

begin

// UPDATE SPINS

spinX.Tag := ST\_UNDEFINED or ST\_UPDATING;

spinY.Tag := ST\_UNDEFINED or ST\_UPDATING;

spinWidth.Tag := ST\_UNDEFINED or ST\_UPDATING;

spinHeigth.Tag := ST\_UNDEFINED or ST\_UPDATING;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

FillMainPropertiesPanel(Left, Top, Width, Heigth);

for I := 0 to Texts.Size - 1 do

with Texts.At[I] do

if Selected then

FillMainPropertiesPanel(Left, Top, Width, Heigth);

for I := 0 to Lines.Size - 1 do

if Lines.At[I].Selected then

begin

spinX.Tag := ST\_ERROR or ST\_UPDATING;

spinY.Tag := ST\_ERROR or ST\_UPDATING;

spinWidth.Tag := ST\_ERROR or ST\_UPDATING;

spinHeigth.Tag := ST\_ERROR or ST\_UPDATING;

end;

UpdateSpins;

// UPDATE TEXT

edtText.Tag := ST\_UNDEFINED or ST\_UPDATING;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

FillTextPanel(Caption);

for I := 0 to Texts.Size - 1 do

with Texts.At[I] do

if Selected then

FillTextPanel(Caption);

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

FillTextPanel(Text.Caption);

UpdateTWinControl(edtText);

spinBorderWidth.Tag := ST\_UNDEFINED or ST\_UPDATING;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

FillBorderPanel(Pen.Width);

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

FillBorderPanel(Pen.Width);

UpdateTWinControl(spinBorderWidth);

end;

procedure TMainForm.spinBorderWidthChange(Sender: TObject);

var

I: Integer;

Tmp: TSpinEdit;

begin

Tmp := Sender as TSpinEdit;

if Tmp.Tag and ST\_UPDATING > 0 then

exit;

if Tmp.Tag = ST\_ERROR then

exit;

for I := 0 to Elements.Size - 1 do

with Elements.At[I] do

if Selected then

Pen.Width := Tmp.Value;

for I := 0 to Lines.Size - 1 do

with Lines.At[I] do

if Selected then

Pen.Width := Tmp.Value;

ClearWorkspace;

ReDraw;

end;

procedure TMainForm.spinMainPropertiesChange(Sender: TObject);

var

I: Integer;

Tmp: TSpinEdit;

begin

if (Sender as TSpinEdit).Tag and ST\_UPDATING > 0 then

exit;

if (Sender as TSpinEdit).Tag = ST\_ERROR then

exit;

Tmp := Sender as TSpinEdit;

if Tmp.Name = spinX.Name then // spinX

begin

for I := 0 to Elements.Size - 1 do

if Elements.At[I].Selected then

with Elements.At[I] do

SetPosition(spinX.Value, Top);

for I := 0 to Texts.Size - 1 do

if Texts.At[I].Selected then

with Texts.At[I] do

SetPosition(spinX.Value, Top);

end

else if Tmp.Name = spinY.Name then // spinY

begin

for I := 0 to Elements.Size - 1 do

if Elements.At[I].Selected then

with Elements.At[I] do

SetPosition(Left, spinY.Value);

for I := 0 to Texts.Size - 1 do

if Texts.At[I].Selected then

with Texts.At[I] do

SetPosition(Left, spinY.Value);

end

else if Tmp.Name = spinWidth.Name then // spinWidth

begin

for I := 0 to Elements.Size - 1 do

if Elements.At[I].Selected then

with Elements.At[I] do

SetSize(spinWidth.Value, Heigth);

for I := 0 to Texts.Size - 1 do

if Texts.At[I].Selected then

with Texts.At[I] do

SetSize(spinWidth.Value, Heigth);

end

else if Tmp.Name = spinHeigth.Name then // spinHeigth

begin

for I := 0 to Elements.Size - 1 do

if Elements.At[I].Selected then

with Elements.At[I] do

SetSize(Width, spinHeigth.Value);

for I := 0 to Texts.Size - 1 do

if Texts.At[I].Selected then

with Texts.At[I] do

SetSize(Width, spinHeigth.Value);

end;

ClearWorkspace;

ReDraw;

end;

procedure TMainForm.tbScaleChange(Sender: TObject);

begin

if tbScale.Tag <> ST\_UPDATING then

begin

FScale := tbScale.Position / 100;

UpdateResolution;

ClearWorkspace;

ReDraw;

end;

end;

procedure TMainForm.toolButtonClick(Sender: TObject);

begin

CurrentTool := TTools((Sender as TToolButton).Tag);

end;

procedure TMainForm.UpdateResolution(PosX: Integer = -1; PosY: Integer = -1);

procedure SetWorkspaceSize(NewWidth, NewHeigth: Integer);

begin

Workspace.Width := NewWidth;

Workspace.Height := NewHeigth;

Workspace.Picture.Bitmap.Width := NewWidth;

Workspace.Picture.Bitmap.Height := NewHeigth;

end;

var

NewWidth, NewHeigth: Integer;

OldWidth, OldHeigth: Integer;

FreezeP, ShiftP: TPoint;

begin

OldWidth := Workspace.Width;

OldHeigth := Workspace.Height;

NewWidth := Round(ImageWidth \* FScale);

NewHeigth := Round(ImageHeigth \* FScale);

if PosX = -1 then

ShiftP.X := ScrollBox.ClientWidth shr 1

else

ShiftP.X := PosX;

if PosY = -1 then

ShiftP.Y := ScrollBox.ClientHeight shr 1

else

ShiftP.Y := PosY;

FreezeP.X := ScrollBox.HorzScrollBar.Position - ScrollBox.ClientWidth + 100;

FreezeP.Y := ScrollBox.VertScrollBar.Position - ScrollBox.ClientHeight + 100;

FreezeP := FreezeP + ShiftP;

FreezeP.X := Round(FreezeP.X \* NewWidth / OldWidth);

FreezeP.Y := Round(FreezeP.Y \* NewHeigth / OldHeigth);

SetWorkspaceSize(NewWidth, NewHeigth);

pnlBackground.Width := Workspace.Width + ScrollBox.ClientWidth \* 2 - 200;

pnlBackground.Height := Workspace.Height + ScrollBox.ClientHeight \* 2 - 200;

Workspace.Top := ScrollBox.ClientHeight - 100;

Workspace.Left := ScrollBox.ClientWidth - 100;

with ScrollBox.HorzScrollBar do

begin

Range := pnlBackground.Width;

Position := FreezeP.X + ScrollBox.ClientWidth - 100 - ShiftP.X;

end;

with ScrollBox.VertScrollBar do

begin

Range := pnlBackground.Height;

Position := FreezeP.Y + ScrollBox.ClientHeight - 100 - ShiftP.Y;

end;

end;

procedure TMainForm.UpdateScale(Delta: Integer);

begin

FScale := FScale + Delta / 500;

if FScale < 0.1 then

FScale := 0.1;

if FScale > 5 then

FScale := 5;

tbScale.Tag := ST\_UPDATING;

tbScale.Position := Trunc(FScale \* 100);

tbScale.Tag := 0;

end;

function GetPoint(Sender: TConnector): TPoint;

begin

if Sender.BindToElement then

Result := Sender.Element.GetCenter

else

Result := Sender.Pos;

end;

procedure TMainForm.WorkspaceMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var

StP, FnP: TPoint;

begin

X := Round(X / FScale);

Y := Round(Y / FScale);

case CurrentTool of

toolMouse:

begin

ElementTmp := IsClickedElements(X, Y);

LineTmp := IsClickedLines(X, Y);

TextTmp := IsClickedTexts(X, Y);

ConnectorTmp := nil;

if (Button = mbLeft) then

begin

StartPoint.Create(X, Y);

if Assigned(LineTmp) then

begin

SelectionState := LineTmp.Selected;

end

else if Assigned(ElementTmp) then

begin

SelectionState := ElementTmp.Selected;

end

else if Assigned(TextTmp) then

begin

SelectionState := TextTmp.Selected;

end;

if not(ssCtrl in Shift) and not(SelectionState) then

DeselectAll;

if Assigned(LineTmp) then

begin

StP := GetPoint(LineTmp.Start);

FnP := GetPoint(LineTmp.Finish);

if (Abs(X - StP.X) < 3 + LineTmp.Pen.Width) and

(Abs(Y - StP.Y) < 3 + LineTmp.Pen.Width) then

ConnectorTmp := @LineTmp.Start

else if (Abs(X - FnP.X) < 3 + LineTmp.Pen.Width) and

(Abs(Y - FnP.Y) < 3 + LineTmp.Pen.Width) then

ConnectorTmp := @LineTmp.Finish

else

ConnectorTmp := nil;

if Assigned(ConnectorTmp) then

DeselectAll;

LineTmp.Selected := True;

end

else if Assigned(ElementTmp) then

begin

ElementTmp.Selected := True;

end

else if Assigned(TextTmp) then

begin

TextTmp.Selected := True;

end

else

DeselectAll;

end;

end;

toolLine:

begin

ElementTmp := IsClickedElements(X, Y);

StartPoint.Create(X, Y);

end;

end;

end;

procedure TMainForm.WorkspaceMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

var

I: Integer;

begin

X := Round(X / FScale);

Y := Round(Y / FScale);

StatusBar.Panels[1].Text := 'X: ' + IntToStr(X) + ' Y: ' + IntToStr(Y);

case CurrentTool of

toolMouse:

begin

if (ssLeft in Shift) and not Assigned(ConnectorTmp) then

begin

ClearWorkspace;

ReDraw;

end;

if ssLeft in Shift then

begin

if Assigned(ConnectorTmp) then

begin

ConnectorTmp^.BindToElement := False;

ConnectorTmp^.Pos.Create(X, Y);

ClearWorkspace;

ReDraw;

end

else

begin

for I := Elements.Size - 1 downto 0 do

begin

if Elements.At[I].Selected then

ElementOnMove(Elements.At[I], X - StartPoint.X,

Y - StartPoint.Y);

end;

for I := Texts.Size - 1 downto 0 do

begin

if Texts.At[I].Selected then

TextOnMove(Texts.At[I], X - StartPoint.X, Y - StartPoint.Y);

end;

end;

end;

end;

end;

end;

procedure TMainForm.WorkspaceMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var

I: Integer;

begin

X := Round(X / FScale);

Y := Round(Y / FScale);

case CurrentTool of

toolMouse:

begin

if (Button = mbLeft) then

begin

if (StartPoint.X = X) and (StartPoint.Y = Y) then

begin

if not(ssCtrl in Shift) then

DeselectAll;

if Assigned(ElementTmp) then

ElementTmp.Selected := not SelectionState

else if Assigned(LineTmp) then

LineTmp.Selected := not SelectionState

else if Assigned(TextTmp) then

TextTmp.Selected := not SelectionState;

end

else

begin

if Assigned(ConnectorTmp) then

begin

ElementTmp := IsClickedElements(X, Y);

if Assigned(ElementTmp) and not(ssAlt in Shift) then

begin

ConnectorTmp^.BindToElement := True;

ConnectorTmp^.Element := ElementTmp;

end

else

begin

ConnectorTmp^.BindToElement := False;

ConnectorTmp^.Pos.Create(X, Y);

end;

end

else

begin

for I := Elements.Size - 1 downto 0 do

if Elements.At[I].Selected then

Elements.At[I].Move(X - StartPoint.X, Y - StartPoint.Y);

for I := Texts.Size - 1 downto 0 do

if Texts.At[I].Selected then

Texts.At[I].Move(X - StartPoint.X, Y - StartPoint.Y);

end;

FChanges := True;

end;

end;

end;

toolRectangle:

begin

AddRectangle(X, Y);

FChanges := True;

end;

toolEllipse:

begin

AddEllipse(X, Y);

FChanges := True;

end;

toolCircle:

begin

AddCircle(X, Y);

FChanges := True;

end;

toolText:

begin

AddText(X, Y);

FChanges := True;

end;

toolLine:

begin

DeselectAll;

LineTmp := TLine.Create;

LineTmp.Canvas := Workspace.Canvas;

LineTmp.Selected := True;

if Assigned(ElementTmp) then

begin

LineTmp.FStart.BindToElement := True;

LineTmp.FStart.Element := ElementTmp;

end

else

begin

LineTmp.FStart.BindToElement := False;

LineTmp.FStart.Pos := StartPoint;

end;

ElementTmp := IsClickedElements(X, Y);

if Assigned(ElementTmp) then

begin

LineTmp.FFinish.BindToElement := True;

LineTmp.FFinish.Element := ElementTmp;

end

else

begin

LineTmp.FFinish.BindToElement := False;

LineTmp.FFinish.Pos.Create(X, Y);

end;

Lines.PushBack(LineTmp);

FChanges := True;

end;

end;

ClearWorkspace;

ReDraw;

ShowPanel;

end;

initialization

end.

**ПРИЛОЖЕНИЕ В**

**Текст программного модуля реализации динамического массива**

unit Vector;

interface

type

TVector<T> = class

private

// Массив данных

FData: array of T;

// Используемый размер

FDataSize: Integer;

// Зарезервированный размер

FReservedSize: Integer;

function ReadAt(const Ind: Integer): T;

procedure WriteAt(const Ind: Integer; const Value: T);

public

constructor Create;

destructor Destroy;

// Получение первого элемента

function Front: T;

// Получение последнего элемента

function Back: T;

// Удаление из массива

procedure Erase(const Id: Integer);

// Добавление в конец массива

procedure PushBack(const X: T);

// Получение размера массива

function Size: Integer;

// Пустой ли массив?

function Empty: Boolean;

// Очистка массива

procedure Clear;

// Зарезервировать X ячеек массива

procedure Reserve(const X: Integer);

// Доступ к элементу массива

property At[const Index: Integer]: T read ReadAt write WriteAt;

end;

implementation

{ TVector<T> }

function TVector<T>.Back: T;

begin

Result := FData[FDataSize - 1];

end;

procedure TVector<T>.Clear;

begin

FDataSize := 0;

FReservedSize := 1;

SetLength(FData, FReservedSize);

end;

constructor TVector<T>.Create;

begin

FDataSize := 0;

FReservedSize := 1;

SetLength(FData, FReservedSize);

end;

destructor TVector<T>.Destroy;

begin

FReservedSize := 0;

SetLength(FData, FReservedSize);

inherited;

end;

function TVector<T>.Empty: Boolean;

begin

Result := FDataSize = 0;

end;

procedure TVector<T>.Erase(const Id: Integer);

var

I: Integer;

begin

Dec(FDataSize);

for I := Id to FDataSize - 1 do

FData[I] := FData[I + 1];

end;

function TVector<T>.Front: T;

begin

Result := FData[0];

end;

procedure TVector<T>.PushBack(const X: T);

begin

if FDataSize = FReservedSize then

begin

FReservedSize := FReservedSize shl 1;

SetLength(FData, FReservedSize);

end;

FData[FDataSize] := X;

Inc(FDataSize);

end;

function TVector<T>.ReadAt(const Ind: Integer): T;

begin

Result := FData[Ind];

end;

procedure TVector<T>.Reserve(const X: Integer);

begin

FDataSize := X;

FReservedSize := X;

SetLength(FData, FReservedSize);

end;

function TVector<T>.Size: Integer;

begin

Result := FDataSize;

end;

procedure TVector<T>.WriteAt(const Ind: Integer; const Value: T);

begin

FData[Ind] := Value;

end;

end.

**ПРИЛОЖЕНИЕ Г**

**Текст программного модуля, реализующего окно с выбором размеров рабочей области**

unit uModalResolution;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Samples.Spin;

type

TformResolution = class(TForm)

spinWidth: TSpinEdit;

spinHeigth: TSpinEdit;

lblHeader: TLabel;

lblWidth: TLabel;

lblHeigth: TLabel;

btnOk: TButton;

btnCancel: TButton;

procedure btnOkClick(Sender: TObject);

procedure btnCancelClick(Sender: TObject);

private

function GetHeight: Integer;

function GetWidth: Integer;

public

property Width: Integer read GetWidth;

property Heigth: Integer read GetHeight;

function ChangeResolution(const AWidth, AHeight: Integer): TModalResult;

end;

var

formResolution: TformResolution;

implementation

{$R \*.dfm}

procedure TformResolution.btnCancelClick(Sender: TObject);

begin

ModalResult := mrCancel;

end;

procedure TformResolution.btnOkClick(Sender: TObject);

begin

ModalResult := mrOk;

end;

function TformResolution.ChangeResolution(const AWidth, AHeight: Integer)

: TModalResult;

begin

spinWidth.Value := AWidth;

spinHeigth.Value := AHeight;

Result := ShowModal;

end;

function TformResolution.GetHeight: Integer;

begin

Result := spinHeigth.Value;

end;

function TformResolution.GetWidth: Integer;

begin

Result := spinWidth.Value;

end;

end.

**ПРИЛОЖЕНИЕ Д**

**Текст программного модуля, реализующего основные классы блоков**

unit Database;

interface

uses

Vcl.Graphics,

Vcl.ExtCtrls,

System.Types,

System.Classes,

SysUtils;

type

TControlObject = class

protected

FId: Integer;

FIsVisible, FIsSelected: Boolean;

FCanvas: TCanvas;

FLeft, FTop: Integer;

FWidth, FHeigth: Integer;

procedure SetCanvas(ACanvas: TCanvas);

public

constructor Create; overLoad;

constructor Create(AObject: TControlObject); overload;

destructor Destroy;

procedure Assign(AObject: TControlObject;

const AScale: Real = 1.0); overload;

procedure Assign(AObject: TControlObject; const XScale: Real;

const YScale: Real); overload;

procedure SetPosition(const X, Y: Integer);

procedure SetSize(const AWidth, AHeigth: Integer);

procedure Move(const DeltaX, DeltaY: Integer);

procedure DrawControls;

function GetCenter(AScale: Real = 1.0): TPoint;

procedure ReadFromFileStream(FileStream: TFileStream);

procedure WriteToFileStream(FileStream: TFileStream);

property Id: Integer read FId write FId;

property Left: Integer read FLeft;

property Top: Integer read FTop;

property Width: Integer read FWidth;

property Heigth: Integer read FHeigth;

property Canvas: TCanvas read FCanvas write SetCanvas;

property Visible: Boolean read FIsVisible write FIsVisible;

property Selected: Boolean read FIsSelected write FIsSelected;

end;

TText = class(TControlObject)

private

FCaption: String;

FBrush: TBrush;

FFont: TFont;

FTextFormat: TTextFormat;

public

constructor Create; overload;

constructor Create(AObject: TText); overload;

destructor Destroy;

procedure Assign(AObject: TText; AScale: Real = 1.0); overload;

procedure Assign(AObject: TText; XScale: Real; YScale: Real); overload;

procedure Draw(AScale: Real = 1.0);

procedure ReadFromFileStream(FileStream: TFileStream);

procedure WriteToFileStream(FileStream: TFileStream);

function IsInside(const X, Y: Integer): Boolean;

property Brush: TBrush read FBrush write FBrush;

property Caption: String read FCaption write FCaption;

property Font: TFont read FFont write FFont;

property TextFormat: TTextFormat read FTextFormat write FTextFormat;

end;

TElement = class(TControlObject)

private

FShape: TShapeType;

FBrush: TBrush;

FPen: TPen;

FCaption: String;

FFont: TFont;

FTextFormat: TTextFormat;

public

constructor Create; overload;

constructor Create(AObject: TElement); overload;

destructor Destroy;

procedure Assign(AObject: TElement; AScale: Real = 1.0); overload;

procedure Assign(AObject: TElement; XScale: Real; YScale: Real); overload;

function IsInside(const X, Y: Integer): Boolean;

procedure Draw(const AScale: Real = 1.0);

procedure ReadFromFileStream(FileStream: TFileStream);

procedure WriteToFileStream(FileStream: TFileStream);

property Brush: TBrush read FBrush write FBrush;

property Caption: String read FCaption write FCaption;

property Font: TFont read FFont write FFont;

property TextFormat: TTextFormat read FTextFormat write FTextFormat;

property Pen: TPen read FPen write FPen;

property Shape: TShapeType read FShape write FShape;

end;

TConnector = record

procedure OffBind;

case BindToElement: Boolean of

True:

(Element: TElement);

False:

(Pos: TPoint);

end;

TLine = class

private

FText: TText;

FIsSelected: Boolean;

FPen: TPen;

FCanvas: TCanvas;

procedure SetCanvas(const ACanvas: TCanvas);

public

FStart, FFinish: TConnector;

constructor Create;

destructor Destroy;

procedure Assign(AObject: TLine; AScale: Real = 1.0); overload;

procedure Assign(AObject: TLine; XScale: Real; YScale: Real); overload;

procedure Draw(AScale: Real = 1.0);

procedure ReadFromFileStream(FileStream: TFileStream);

procedure WriteToFileStream(FileStream: TFileStream);

function IsInside(const X, Y: Integer): Boolean;

property Text: TText read FText write FText;

property Selected: Boolean read FIsSelected write FIsSelected;

property Pen: TPen read FPen write FPen;

property Canvas: TCanvas read FCanvas write SetCanvas;

property Start: TConnector read FStart write FStart;

property Finish: TConnector read FFinish write FFinish;

end;

implementation

uses

MainWindow;

var

TmpElement: TElement;

TmpLine: TLine;

TmpText: TText;

const

DefaultTextFormat: TTextFormat = [tfNoClip, tfWordBreak];

{ TLine }

function Min(const a, b: Real): Real; overload;

begin

Result := a;

if a > b then

Result := b;

end;

procedure TLine.Assign(AObject: TLine; XScale, YScale: Real);

begin

FText.Assign(AObject.Text);

FPen.Assign(AObject.Pen);

FPen.Width := Round(FPen.Width \* Min(XScale, YScale));

FIsSelected := AObject.Selected;

FStart := AObject.Start;

FFinish := AObject.Finish;

FCanvas := AObject.Canvas;

if not FStart.BindToElement then

begin

FStart.Pos.X := Round(FStart.Pos.X \* XScale);

FStart.Pos.Y := Round(FStart.Pos.Y \* YScale);

end;

if not FFinish.BindToElement then

begin

FFinish.Pos.X := Round(FFinish.Pos.X \* XScale);

FFinish.Pos.Y := Round(FFinish.Pos.Y \* YScale);

end;

end;

procedure TLine.Assign(AObject: TLine; AScale: Real);

begin

FText.Assign(AObject.Text, AScale);

FPen.Assign(AObject.Pen);

FPen.Width := Round(FPen.Width \* AScale);

FIsSelected := AObject.Selected;

FStart := AObject.Start;

FFinish := AObject.Finish;

FCanvas := AObject.Canvas;

if not FStart.BindToElement then

begin

FStart.Pos.X := Round(FStart.Pos.X \* AScale);

FStart.Pos.Y := Round(FStart.Pos.Y \* AScale);

end;

if not FFinish.BindToElement then

begin

FFinish.Pos.X := Round(FFinish.Pos.X \* AScale);

FFinish.Pos.Y := Round(FFinish.Pos.Y \* AScale);

end;

end;

constructor TLine.Create;

begin

FText := TText.Create;

FText.FCaption := '';

FPen := TPen.Create;

end;

destructor TLine.Destroy;

begin

FText.Free;

FPen.Free;

Inherited;

end;

procedure TLine.Draw;

const

DotSize = 3;

procedure DrawDot(const P: TPoint);

begin

FCanvas.Brush.Color := clBlue;

FCanvas.Rectangle(P.X - DotSize, P.Y - DotSize, P.X + DotSize,

P.Y + DotSize);

end;

var

StartPoint, FinishPoint: TPoint;

Size: Integer;

begin

TmpLine.Assign(Self, AScale);

with TmpLine do

begin

case FStart.BindToElement of

True:

StartPoint := FStart.Element.GetCenter(AScale);

False:

StartPoint := FStart.Pos;

end;

FCanvas.MoveTo(StartPoint.X, StartPoint.Y);

FCanvas.Pen := FPen;

case FFinish.BindToElement of

True:

FinishPoint := FFinish.Element.GetCenter(AScale);

False:

FinishPoint := FFinish.Pos;

end;

FCanvas.LineTo(FinishPoint.X, FinishPoint.Y);

if FIsSelected then

begin

FCanvas.Pen.Color := clBlue;

FCanvas.Pen.Width := 2;

FCanvas.LineTo(StartPoint.X, StartPoint.Y);

DrawDot(StartPoint);

DrawDot(FinishPoint);

end;

FCanvas.Brush.Style := bsClear;

FCanvas.Font := FText.Font;

Size := FCanvas.TextWidth(FText.Caption);

FText.FLeft := StartPoint.X;

FText.FTop := StartPoint.Y;

FText.FWidth := FinishPoint.X - StartPoint.X;

FText.FHeigth := FinishPoint.Y - StartPoint.Y;

if FText.Width < Size then

begin

FText.FLeft := FText.Left - (Size - FText.Width) shr 1;

FText.FWidth := Size;

end;

FText.Draw;

end;

end;

function TLine.IsInside(const X, Y: Integer): Boolean;

const

threshold = 5;

function Max(a, b: Integer): Integer;

begin

Result := a;

if a < b then

Result := b;

end;

function Min(a, b: Integer): Integer;

begin

Result := a;

if a > b then

Result := b;

end;

var

a, b, C: Integer;

P, Q: TPoint;

tmp: Real;

begin

if FStart.BindToElement then

P := FStart.Element.GetCenter

else

P := FStart.Pos;

if FFinish.BindToElement then

Q := FFinish.Element.GetCenter

else

Q := FFinish.Pos;

a := P.Y - Q.Y;

b := Q.X - P.X;

C := -a \* P.X - b \* P.Y;

if (a = 0) and (b = 0) then

Result := False

else if (Min(P.X, Q.X) - threshold <= X) and (X <= Max(P.X, Q.X) + threshold)

and (Min(P.Y, Q.Y) - threshold <= Y) and (Y <= Max(P.Y, Q.Y) + threshold)

then

Result := Abs(a \* X + b \* Y + C) / Sqrt(Sqr(a) + Sqr(b)) < threshold

else

Result := False;

end;

procedure TLine.ReadFromFileStream(FileStream: TFileStream);

procedure ReadConnector(var Connector: TConnector);

begin

FileStream.Read(Connector.BindToElement, SizeOf(Connector.BindToElement));

if Connector.BindToElement then

FileStream.Read(Connector.Element, SizeOf(Integer))

else

FileStream.Read(Connector.Pos, SizeOf(Connector.Pos));

end;

var

Color: TColor;

PenWidth: Integer;

PenStyle: TPenStyle;

PenMode: TPenMode;

begin

FText.ReadFromFileStream(FileStream);

// Pen

FileStream.Read(Color, SizeOf(Color));

FPen.Color := Color;

FileStream.Read(PenWidth, SizeOf(PenWidth));

FPen.Width := PenWidth;

FileStream.Read(PenStyle, SizeOf(PenStyle));

FPen.Style := PenStyle;

FileStream.Read(PenMode, SizeOf(PenMode));

FPen.Mode := PenMode;

ReadConnector(FStart);

ReadConnector(FFinish);

end;

procedure TLine.SetCanvas(const ACanvas: TCanvas);

begin

Self.FCanvas := ACanvas;

FText.Canvas := ACanvas;

end;

procedure TLine.WriteToFileStream(FileStream: TFileStream);

procedure WriteConnector(const Connector: TConnector);

begin

FileStream.Write(Connector.BindToElement, SizeOf(Connector.BindToElement));

if Connector.BindToElement then

FileStream.Write(Connector.Element.Id, SizeOf(Connector.Element.Id))

else

FileStream.Write(Connector.Pos, SizeOf(Connector.Pos));

end;

var

Color: TColor;

PenWidth: Integer;

PenStyle: TPenStyle;

PenMode: TPenMode;

begin

FText.WriteToFileStream(FileStream);

// Pen

Color := FPen.Color;

FileStream.Write(Color, SizeOf(Color));

PenWidth := FPen.Width;

FileStream.Write(PenWidth, SizeOf(PenWidth));

PenStyle := FPen.Style;

FileStream.Write(PenStyle, SizeOf(PenStyle));

PenMode := FPen.Mode;

FileStream.Write(PenMode, SizeOf(PenMode));

WriteConnector(FStart);

WriteConnector(FFinish);

end;

{ TText }

constructor TText.Create;

begin

FBrush := TBrush.Create;

FFont := TFont.Create;

FTextFormat := DefaultTextFormat;

FCaption := '';

end;

procedure TText.Assign(AObject: TText; AScale: Real);

begin

Inherited Assign(AObject, AScale);

FCaption := AObject.Caption;

FBrush.Assign(AObject.Brush);

FFont.Assign(AObject.Font);

FFont.Height := Round(FFont.Height \* AScale);

FTextFormat := AObject.TextFormat;

end;

procedure TText.Assign(AObject: TText; XScale, YScale: Real);

begin

Inherited Assign(AObject, XScale, YScale);

FCaption := AObject.Caption;

FBrush.Assign(AObject.Brush);

FFont.Assign(AObject.Font);

FFont.Height := Round(FFont.Height \* Min(XScale, YScale));

FTextFormat := AObject.TextFormat;

end;

constructor TText.Create(AObject: TText);

begin

Inherited Create(AObject);

FCaption := AObject.Caption;

FBrush := TBrush.Create;

FFont := TFont.Create;

FBrush.Assign(AObject.Brush);

FFont.Assign(AObject.Font);

FTextFormat := AObject.TextFormat;

end;

destructor TText.Destroy;

begin

FFont.Destroy;

Inherited;

end;

procedure TText.Draw;

var

Rect: TRect;

tfCalc: TTextFormat;

begin

with TmpText do

begin

Assign(Self, AScale);

tfCalc := FTextFormat;

Include(tfCalc, tfCalcRect);

Rect.Create(FLeft, FTop + FHeigth shr 1, FLeft + FWidth,

FTop + FHeigth shr 1);

FCanvas.TextRect(Rect, FCaption, tfCalc);

if Rect.Width <= FWidth then

Rect.Create(FLeft + (FWidth - Rect.Width) shr 1,

FTop + (FHeigth - Rect.Height) shr 1, FLeft + (FWidth + Rect.Width)

shr 1, FTop + (FHeigth + Rect.Height) shr 1)

else

Rect.Create(FLeft, FTop + (FHeigth - Rect.Height) shr 1, FLeft + FWidth,

FTop + (FHeigth + Rect.Height) shr 1);

FCanvas.Font := FFont;

FCanvas.Brush := FBrush;

FCanvas.TextRect(Rect, FCaption, FTextFormat);

if FIsSelected then

begin

FCanvas.Brush.Style := bsClear;

FCanvas.Pen.Color := clBlue;

Rect.Create(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

FCanvas.Rectangle(Rect);

end;

end;

end;

function TText.IsInside(const X, Y: Integer): Boolean;

begin

Result := (X >= FLeft) and (Y >= FTop) and (X <= FLeft + FWidth) and

(Y <= FTop + FHeigth);

end;

procedure TText.ReadFromFileStream(FileStream: TFileStream);

var

CaptionSize: Integer;

Color: TColor;

BrushStyle: TBrushStyle;

FontStyle: TFontStyles;

FontSize, Orientation: Integer;

begin

Inherited;

// Caption

FileStream.Read(CaptionSize, SizeOf(CaptionSize));

SetLength(FCaption, CaptionSize);

FileStream.Read(Pointer(FCaption)^, CaptionSize \* 2);

FileStream.Read(FTextFormat, SizeOf(FTextFormat));

// Brush

FileStream.Read(Color, SizeOf(Color));

FBrush.Color := Color;

FileStream.Read(BrushStyle, SizeOf(BrushStyle));

FBrush.Style := BrushStyle;

// Font

FileStream.Read(FontStyle, SizeOf(FontStyle));

FFont.Style := FontStyle;

FileStream.Read(FontSize, SizeOf(FontSize));

FFont.Size := FontSize;

FileStream.Read(Color, SizeOf(Color));

FFont.Color := Color;

FileStream.Read(Orientation, SizeOf(Orientation));

FFont.Orientation := Orientation;

end;

procedure TText.WriteToFileStream(FileStream: TFileStream);

var

CaptionLength: Integer;

Color: TColor;

BrushStyle: TBrushStyle;

FontStyle: TFontStyles;

FontSize, Orientation: Integer;

begin

Inherited;

// Caption

CaptionLength := Length(FCaption);

FileStream.Write(CaptionLength, SizeOf(CaptionLength));

FileStream.Write(Pointer(FCaption)^, CaptionLength \* 2);

FileStream.Write(FTextFormat, SizeOf(FTextFormat));

// Brush

Color := FBrush.Color;

FileStream.Write(Color, SizeOf(Color));

BrushStyle := Brush.Style;

FileStream.Write(BrushStyle, SizeOf(BrushStyle));

// Font

FontStyle := FFont.Style;

FileStream.Write(FontStyle, SizeOf(FontStyle));

FontSize := FFont.Size;

FileStream.Write(FontSize, SizeOf(FontSize));

Color := FFont.Color;

FileStream.Write(Color, SizeOf(Color));

Orientation := FFont.Orientation;

FileStream.Write(Orientation, SizeOf(Orientation));

end;

{ TElement }

constructor TElement.Create;

begin

Inherited;

FTextFormat := DefaultTextFormat;

FFont := TFont.Create;

FBrush := TBrush.Create;

FPen := TPen.Create;

end;

procedure TElement.Assign(AObject: TElement; AScale: Real = 1.0);

begin

Inherited Assign(AObject, AScale);

FShape := AObject.Shape;

FCaption := AObject.Caption;

FFont.Assign(AObject.Font);

FFont.Height := Round(FFont.Height \* AScale);

FBrush.Assign(AObject.Brush);

FPen.Assign(AObject.Pen);

FPen.Width := Round(FPen.Width \* AScale);

FTextFormat := AObject.TextFormat;

end;

procedure TElement.Assign(AObject: TElement; XScale, YScale: Real);

begin

Inherited Assign(AObject, XScale, YScale);

FShape := AObject.Shape;

FCaption := AObject.Caption;

FFont.Assign(AObject.Font);

// FFont.Height := Round(FFont.Height \* AScale);

FBrush.Assign(AObject.Brush);

FPen.Assign(AObject.Pen);

FTextFormat := AObject.TextFormat;

end;

constructor TElement.Create(AObject: TElement);

begin

Inherited Create(AObject);

FShape := AObject.Shape;

FBrush := TBrush.Create;

FBrush.Assign(AObject.Brush);

FCaption := AObject.Caption;

FFont := TFont.Create;

FFont.Assign(AObject.Font);

FPen := TPen.Create;

FPen.Assign(AObject.Pen);

FTextFormat := AObject.TextFormat;

end;

destructor TElement.Destroy;

begin

FFont.Free;

FBrush.Free;

FPen.Free;

Inherited;

end;

procedure TElement.Draw(const AScale: Real = 1.0);

var

Rect: TRect;

tfCalc: TTextFormat;

tmp: TElement;

begin

if FIsVisible then

begin

with TmpElement do

begin

Assign(Self, AScale);

FCanvas.Pen := FPen;

FCanvas.Brush := FBrush;

FCanvas.Font := FFont;

case FShape of

stRectangle, stSquare:

Canvas.Rectangle(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

// stRoundRect: ;

// stRoundSquare: ;

stEllipse, stCircle:

Canvas.Ellipse(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

end;

tfCalc := FTextFormat;

Include(tfCalc, tfCalcRect);

Rect.Create(FLeft, FTop + FHeigth shr 1, FLeft + FWidth,

FTop + FHeigth shr 1);

Rect.Width := Rect.Width - FPen.Width - 3;

FCanvas.TextRect(Rect, FCaption, tfCalc);

if Rect.Width <= FWidth - FPen.Width - 3 then

Rect.Create(FLeft + (FWidth - Rect.Width) shr 1,

FTop + (FHeigth - Rect.Height) shr 1, FLeft + (FWidth + Rect.Width)

shr 1, FTop + (FHeigth + Rect.Height) shr 1)

else

begin

Rect.Create(FLeft, FTop + (FHeigth - Rect.Height) shr 1, FLeft + FWidth,

FTop + (FHeigth + Rect.Height) shr 1);

Rect.Left := Rect.Left + (FPen.Width shr 1) + 2;

Rect.Width := Rect.Width - FPen.Width - 3;

end;

FCanvas.Brush.Style := bsClear;

FCanvas.TextRect(Rect, FCaption, FTextFormat);

// Rect.Create(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);;

// FCanvas.TextRect(Rect, FCaption, FTextFormat);

if FIsSelected then

begin

FCanvas.Brush.Style := bsClear;

FCanvas.Pen.Color := clBlue;

FCanvas.Pen.Width := 3;

case FShape of

stRectangle, stSquare:

FCanvas.Rectangle(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

// stRoundRect: ;

// stRoundSquare: ;

stEllipse, stCircle:

FCanvas.Ellipse(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

end;

end;

end;

end;

end;

function TElement.IsInside(const X, Y: Integer): Boolean;

function IsInEllipse(const X, Y, a2, b2: Integer): Boolean;

begin

if (a2 = 0) or (b2 = 0) then

Result := False

else

Result := (Sqr(X) / Sqr(a2) + Sqr(Y) / Sqr(b2)) <= 0.25;

end;

begin

case FShape of

stRectangle, stSquare:

Result := (X >= FLeft) and (Y >= FTop) and (X <= FLeft + FWidth) and

(Y <= FTop + FHeigth);

// stRoundRect: ;

// stRoundSquare: ;

stEllipse, stCircle:

Result := IsInEllipse(X - FLeft - (FWidth shr 1),

Y - FTop - (FHeigth shr 1), FWidth, FHeigth);

end;

end;

procedure TElement.ReadFromFileStream(FileStream: TFileStream);

var

Color: TColor;

BrushStyle: TBrushStyle;

PenWidth: Integer;

PenStyle: TPenStyle;

PenMode: TPenMode;

FontStyle: TFontStyles;

FontSize: Integer;

Orientation: Integer;

CaptionSize: Integer;

CharSize: ^SmallInt;

begin

Inherited;

FileStream.Read(FShape, SizeOf(FShape));

// Caption

FileStream.Read(CaptionSize, SizeOf(CaptionSize));

SetLength(FCaption, CaptionSize);

FileStream.Read(Pointer(FCaption)^, CaptionSize \* 2);

FileStream.Read(FTextFormat, SizeOf(FTextFormat));

// Brush

FileStream.Read(Color, SizeOf(Color));

FBrush.Color := Color;

FileStream.Read(BrushStyle, SizeOf(BrushStyle));

FBrush.Style := BrushStyle;

// Pen

FileStream.Read(Color, SizeOf(Color));

FPen.Color := Color;

FileStream.Read(PenWidth, SizeOf(PenWidth));

FPen.Width := PenWidth;

FileStream.Read(PenStyle, SizeOf(PenStyle));

FPen.Style := PenStyle;

FileStream.Read(PenMode, SizeOf(PenMode));

FPen.Mode := PenMode;

// Font

FileStream.Read(FontStyle, SizeOf(FontStyle));

FFont.Style := FontStyle;

FileStream.Read(FontSize, SizeOf(FontSize));

FFont.Size := FontSize;

FileStream.Read(Color, SizeOf(Color));

FFont.Color := Color;

FileStream.Read(Orientation, SizeOf(Orientation));

FFont.Orientation := Orientation;

end;

procedure TElement.WriteToFileStream(FileStream: TFileStream);

var

Color: TColor;

BrushStyle: TBrushStyle;

PenWidth: Integer;

PenStyle: TPenStyle;

PenMode: TPenMode;

FontStyle: TFontStyles;

FontSize: Integer;

Orientation: Integer;

CaptionLength: Integer;

begin

Inherited;

FileStream.Write(FShape, SizeOf(FShape));

// Caption

CaptionLength := Length(FCaption);

FileStream.Write(CaptionLength, SizeOf(CaptionLength));

FileStream.Write(Pointer(FCaption)^, CaptionLength \* 2);

FileStream.Write(FTextFormat, SizeOf(FTextFormat));

// Brush

Color := FBrush.Color;

FileStream.Write(Color, SizeOf(Color));

BrushStyle := Brush.Style;

FileStream.Write(BrushStyle, SizeOf(BrushStyle));

// Pen

Color := FPen.Color;

FileStream.Write(Color, SizeOf(Color));

PenWidth := FPen.Width;

FileStream.Write(PenWidth, SizeOf(PenWidth));

PenStyle := FPen.Style;

FileStream.Write(PenStyle, SizeOf(PenStyle));

PenMode := FPen.Mode;

FileStream.Write(PenMode, SizeOf(PenMode));

// Font

FontStyle := FFont.Style;

FileStream.Write(FontStyle, SizeOf(FontStyle));

FontSize := FFont.Size;

FileStream.Write(FontSize, SizeOf(FontSize));

Color := FFont.Color;

FileStream.Write(Color, SizeOf(Color));

Orientation := FFont.Orientation;

FileStream.Write(Orientation, SizeOf(Orientation));

end;

{ TControlObject }

constructor TControlObject.Create;

begin

FIsVisible := True;

Inherited;

end;

procedure TControlObject.Assign(AObject: TControlObject;

const AScale: Real = 1.0);

begin

Self.FIsVisible := AObject.Visible;

Self.FIsSelected := AObject.Selected;

Self.FCanvas := AObject.Canvas;

Self.FLeft := Round(AObject.Left \* AScale);

Self.FTop := Round(AObject.Top \* AScale);

Self.FWidth := Round(AObject.Width \* AScale);

Self.FHeigth := Round(AObject.Heigth \* AScale);

end;

procedure TControlObject.Assign(AObject: TControlObject;

const XScale, YScale: Real);

begin

Self.FIsVisible := AObject.Visible;

Self.FIsSelected := AObject.Selected;

Self.FCanvas := AObject.Canvas;

Self.FLeft := Round(AObject.Left \* XScale);

Self.FTop := Round(AObject.Top \* YScale);

Self.FWidth := Round(AObject.Width \* XScale);

Self.FHeigth := Round(AObject.Heigth \* YScale);

end;

constructor TControlObject.Create(AObject: TControlObject);

begin

Self.FIsVisible := AObject.Visible;

Self.FIsSelected := False;

Self.FCanvas := AObject.Canvas;

Self.FLeft := AObject.Left;

Self.FTop := AObject.Top;

Self.FWidth := AObject.Width;

Self.FHeigth := AObject.Heigth;

end;

destructor TControlObject.Destroy;

begin

Inherited;

end;

procedure TControlObject.DrawControls;

const

DotSize = 3;

procedure DrawDot(const X, Y: Integer);

begin

FCanvas.Rectangle(X - DotSize, Y - DotSize, X + DotSize, Y + DotSize);

end;

begin

if FIsVisible then

begin

FCanvas.Brush.Style := bsClear;

FCanvas.Pen.Style := psDash;

FCanvas.Pen.Color := clBlue;

FCanvas.Rectangle(FLeft, FTop, FLeft + FWidth, FTop + FHeigth);

FCanvas.Brush.Style := bsSolid;

FCanvas.Brush.Color := clGreen;

FCanvas.Pen.Style := psSolid;

FCanvas.Pen.Color := clBlack;

DrawDot(FLeft, FTop); // 0

DrawDot(FLeft + FWidth shr 1, FTop); // 1

DrawDot(FLeft + FWidth, FTop); // 2

DrawDot(FLeft + FWidth, FTop + FHeigth shr 1); // 3

DrawDot(FLeft + FWidth, FTop + FHeigth); // 4

DrawDot(FLeft + FWidth shr 1, FTop + FHeigth); // 5

DrawDot(FLeft, FTop + FHeigth); // 6

DrawDot(FLeft, FTop + FHeigth shr 1); // 7

end;

end;

function TControlObject.GetCenter(AScale: Real = 1.0): TPoint;

begin

Result.X := Round(FLeft \* AScale) + Round(FWidth \* AScale) shr 1;

Result.Y := Round(FTop \* AScale) + Round(FHeigth \* AScale) shr 1;

end;

procedure TControlObject.Move(const DeltaX, DeltaY: Integer);

begin

Inc(FLeft, DeltaX);

Inc(FTop, DeltaY);

end;

procedure TControlObject.ReadFromFileStream(FileStream: TFileStream);

begin

FileStream.Read(FId, SizeOf(FId));

FileStream.Read(FLeft, SizeOf(FLeft));

FileStream.Read(FTop, SizeOf(FTop));

FileStream.Read(FWidth, SizeOf(FWidth));

FileStream.Read(FHeigth, SizeOf(FHeigth));

end;

procedure TControlObject.SetCanvas(ACanvas: TCanvas);

begin

Self.FCanvas := ACanvas;

end;

procedure TControlObject.SetPosition(const X, Y: Integer);

begin

FLeft := X;

FTop := Y;

end;

procedure TControlObject.SetSize(const AWidth, AHeigth: Integer);

begin

FWidth := AWidth;

FHeigth := AHeigth;

end;

procedure TControlObject.WriteToFileStream(FileStream: TFileStream);

begin

FileStream.Write(FId, SizeOf(FId));

FileStream.Write(FLeft, SizeOf(FLeft));

FileStream.Write(FTop, SizeOf(FTop));

FileStream.Write(FWidth, SizeOf(FWidth));

FileStream.Write(FHeigth, SizeOf(FHeigth));

end;

{ TConnector }

procedure TConnector.OffBind;

begin

if BindToElement then

begin

BindToElement := False;

Pos := Element.GetCenter;

end;

end;

initialization

TmpElement := TElement.Create;

TmpLine := TLine.Create;

TmpText := TText.Create;

finalization

TmpElement.Free;

TmpLine.Free;

TmpText.Free;

end.

**ПРИЛОЖЕНИЕ Е**

**Текст программного модуля, реализующего окно «About»**

unit uAbout;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

type

TfrmAbout = class(TForm)

memoMain: TMemo;

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmAbout: TfrmAbout;

implementation

{$R \*.dfm}

end.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 119 ПЗ | | | | Пояснительная записка | | | | 85 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951001 119 CA | | | | "Программное средство построения и отображения сетевых и древовидных структур", А1, схема программы, чертеж. | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 119 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство построения и отображения сетевых и древовидных структур  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Северин К.М. |  |  | Т |  | |  | 85 | 85 |
| Пров. | | Фадеева Е.Е. |  |  | Кафедра ПОИТ  гр. 951001 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |