

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	4
ВВЕДЕНИЕ	6
1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ	7
1.1. Описание синтаксиса языка с помощью синтаксических диаграмм	7
1.2. Анализ существующих аналогов	9
1.3. Постановка задачи	12
2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ	13
2.1. Представление векторного изображения в формате SVG	13
2.2. Описание функциональности ПС	15
2.3. Спецификация функциональных требований.	16
3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА	19
3.1. Проектирование динамических структур данных	19
3.2. Разработка алгоритма реакции на клик пользователя по полотну	21
3.3. Разработка алгоритма перемещение точки внутри линии	22
3.4. Разработка алгоритма «примагничивания фигур»	24
3.4.1. Разработка алгоритма поиска линии вблизи точки	25
3.4.2. Разработка алгоритма «примагничивания» линии к текстовой фигуре 26	
3.4.3. Разработка алгоритма «примагничивания» точки к линии	28
3.4.4. Обобщение алгоритмов «примагничивания»	29
3.5. Разработка алгоритма отрисовки линий	30
3.6. Разработка алгоритм отрисовки фигур	31
4. СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА	32
4.1. Взаимодействие между формами	32
4.2. Структура модулей программы	34
4.3. Описание модуля Main	35
4.4. Описание модуля FCanvasSizeSettings	37

4.5.	Описание модуля FHTMLView.....	38
4.6.	Описание модулей «Модели»	38
4.6.1.	Описание модуля Model	38
4.6.2.	Описание модуля Model.Lines	40
4.6.3.	Описание модуля Model.UndoStack	42
4.7.	Описание модулей «Представления»	44
4.7.1.	Описание модуля View.Canvas	44
4.7.2.	Описание модуля View.SVG	45
5.	ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ	47
5.1.	Тестирование функционала добавления фигур.....	47
5.2.	Тестирование отображения линий при разном взаимном расположении фигур.50	
5.3.	Тестирование функционала редактирования текстовых фигур.	60
5.4.	Тестирование функционала редактирования линий.....	62
5.5.	Тестирование функционала изменения размера полотна	63
5.6.	Тестирование функционала отмены изменений	64
5.7.	Тестирование функционала «примагничивания»	65
5.8.	Тестирование прочего функционала программного средства.....	66
5.9.	Вывод из прохождения тестирования	68
6.	РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ	69
	ЗАКЛЮЧЕНИЕ	73
	СПИСОК ЛИТЕРАТУРЫ.....	74
	ПРИЛОЖЕНИЕ 1	75
	ПРИЛОЖЕНИЕ 2	76

ВВЕДЕНИЕ

Для того, чтобы познакомиться и с новым языком программирования и написать несложную программу, программисту необходимо ознакомиться с грамматическим описанием языка. Грамматическое описание любого языка программирования включает в себя алфавит, синтаксис и семантику.

Для описания правил синтаксиса языка программирования применяются формализованные системы обозначений – метаязыки. Существует два основных метаязыка:

- Расширенная форма Бэкуса-Наура (РБНФ);
- Синтаксические диаграммы.

Язык РБНФ более строг и точен, более удобен для представления синтаксиса в памяти машины, более компактен. Синтаксические диаграммы же более громоздки, однако намного нагляднее и проще для понимания.

Данная курсовая работа посвящена разработке программного средства для построения синтаксических диаграмм с использованием векторной графики.

В ходе выполнения данного проекта я постараюсь понять:

1. Как реализовать работу с графикой в языке программирования Delphi;
2. Как спроектировать наиболее удобный пользовательский интерфейс графического редактора;
3. Как устроен формат SVG;

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

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.

1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1. Описание синтаксиса языка с помощью синтаксических диаграмм

Синтаксическая диаграмма позволяет графически изобразить структуру синтаксической единицы.

В метаязыках, описывающих синтаксис языка программирования, используются следующие основные понятия:^[1]

- **Метапеременная** – обозначает определенную синтаксисом конструкцию языка. Для записи метапеременных в основном используются последовательности слов на естественном языке (русский, английский или др.) и служебных слов. Для разделения слов используется символ нижнего подчеркивания (). В синтаксических диаграммах метапеременные заключаются в угловые скобки ($\langle \rangle$). Метапеременная на размеченном ребре графа означает, что этот фрагмент диаграммы должен быть детализирован подстановкой синтаксической диаграммы с именем, соответствующим данной метапеременной.

Примеры записи метапеременных:

$\langle \text{Оператор_For} \rangle$

$\langle \text{Тип_Set} \rangle$

$\langle \text{Базовый_скалярный_тип} \rangle$

- **Метаконстанты** – обозначает лексему языка программирования. В программе метаконстанте соответствует она сама. В синтаксических диаграммах метаконстанты записываются «как есть».

Примеры метаконстант:

For

Begin

Set

- **Метасимвол** – специальный символ, используемый для описания синтаксиса языка. В синтаксических диаграммах присутствует два единственных метасимвола:
 - Метасимвол “ $::=$ ” – используется для отделения имени синтаксической диаграммы.
 - Метасимвол “ $\langle \rangle$ ” – используется для обозначения метапеременных

Синтаксическая диаграмма представляет собой ориентированный граф с размеченными ребрами. Для разметки ребер используются метаконстанты и метапеременные.

Представление в виде ориентированных графов основных конструкций:

1. Выбор (Альтернатива).

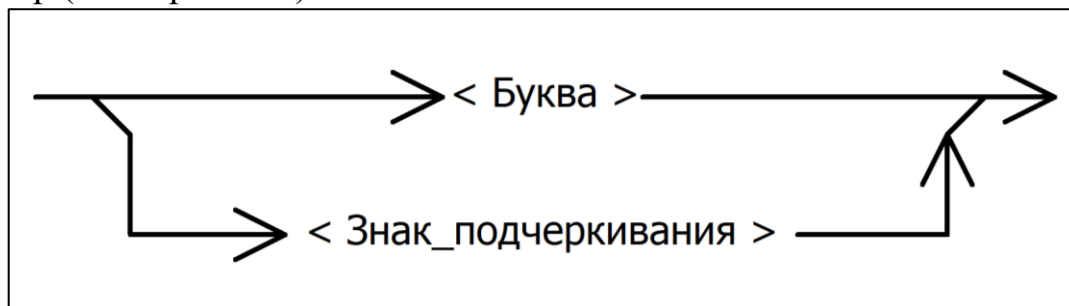


Рисунок 1.1 – пример конструкции выбора

2. Необязательная часть конструкции (Повторяется либо 1, либо 0 раз).

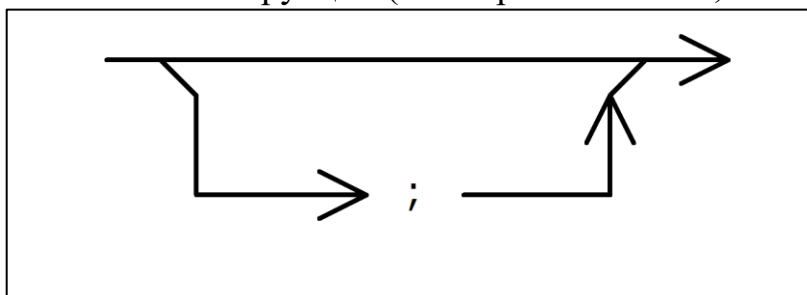


Рисунок 1.2 – пример необязательной части конструкции

3. Повторение конструкции

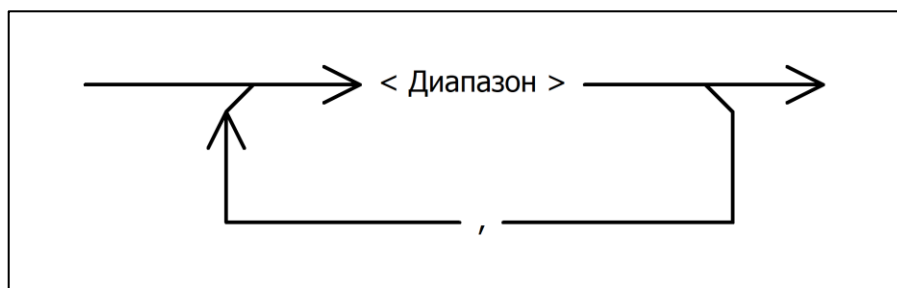


Рисунок 1.3 – пример повторения конструкции

1.2. Анализ существующих аналогов

После тщательных поисков мною не было найдено программного средства для создания синтаксических диаграмм, тем более, чтобы оно работало с векторной графикой. Поэтому в качестве аналогов я рассмотрю графические редакторы с возможностью работы с векторной графикой.

Adobe Illustrator – один из наиболее распространенных векторных графических редакторов. Разрабатывается и распространяется компанией Adobe Systems.

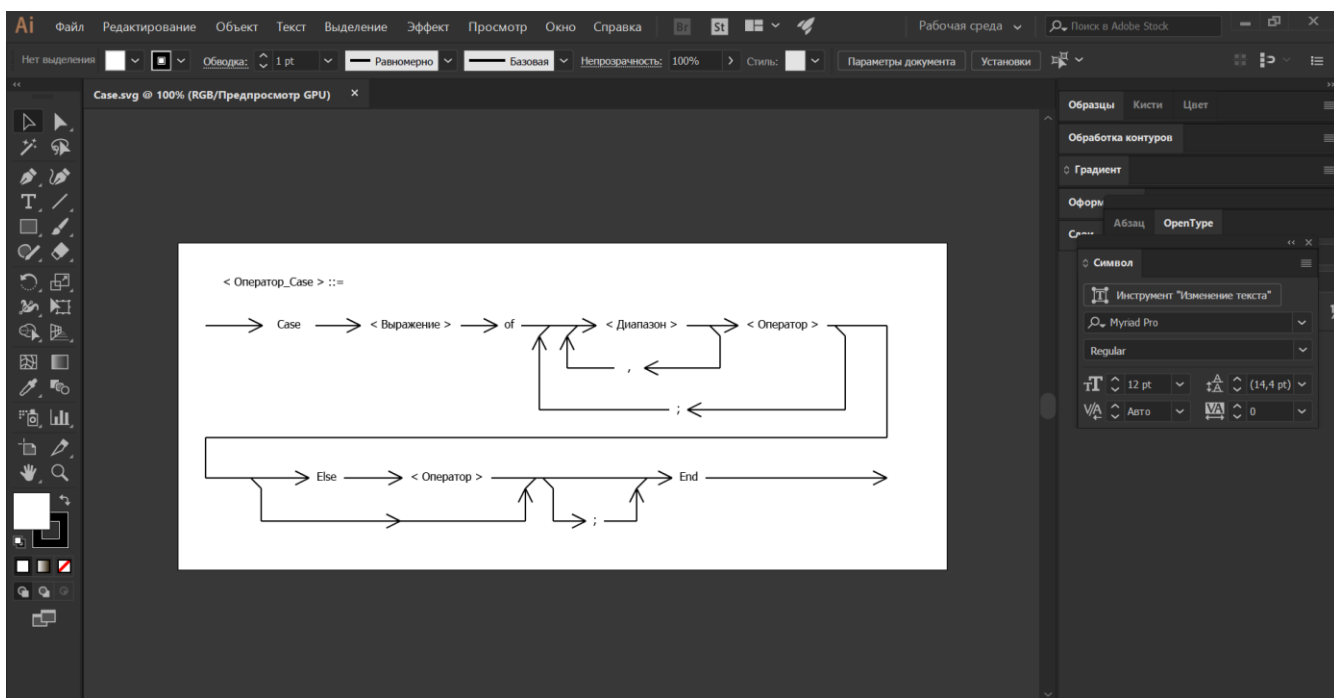


Рисунок 1.4 – Скриншот Adobe Illustrator

Плюсы:

- Удобный пользовательский интерфейс
- Стабильная работа программы
- Удобный процесс экспорта

Минусы:

- Высокая стоимость
- Программа заточена под редактирование произвольного векторного изображение, из-за чего есть много ненужных для построения синтаксических диаграмм функций, а также каждую стрелочку, линию, выравнивание фигур приходится производить самостоятельно.

CorelDRAW — графический редактор, разработанный канадской корпорацией Corel.

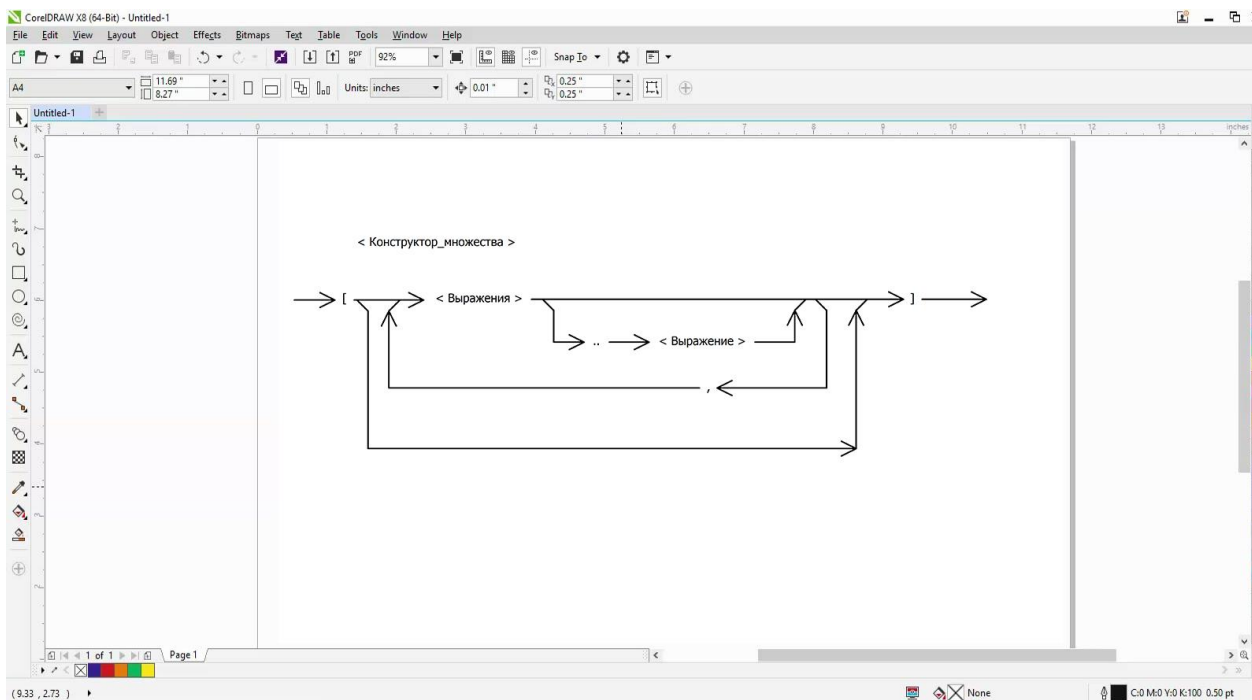


Рисунок 1.4 – Скриншот CorelDraw

Функционал CorelDraw очень схож с Adobe Illustrator.

Плюсы:

- Стабильная работа
- Удобный пользовательский интерфейс

Минусы:

- Высокая стоимость
- Программа не заточена под рисование синтаксических диаграмм

Inkscape – свободно распространяемый аналог Adobe Illustrator и Corel Draw. Работает с векторными изображениями в формате .svg.

Программа распространяется на условиях GNU (General Public License).

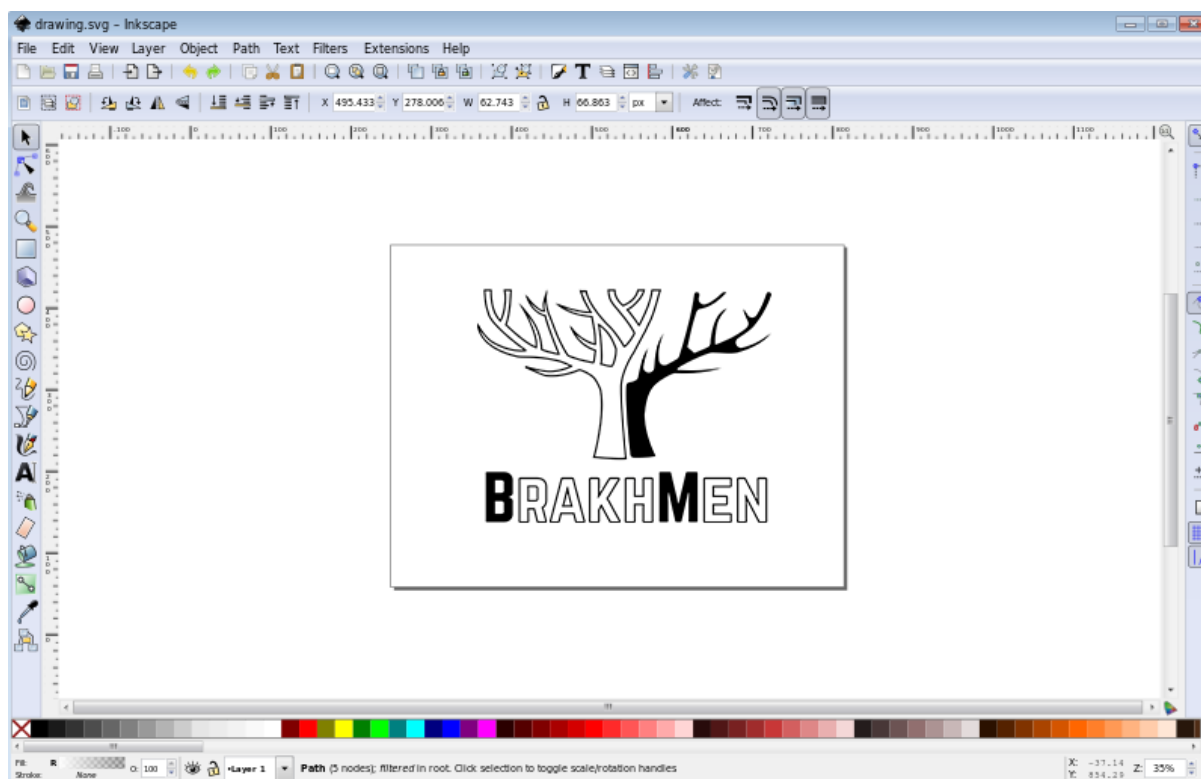


Рисунок 1.5 – скриншот inkscape

Плюсы:

- Бесплатный

Минусы:

- Программа не заточена под рисование синтаксических диаграмм
- Устаревший пользовательский интерфейс
- Повышенные требования к системным ресурсам
- «Сырость» ряда фильтров импорта
- Невозможность использования привычных горячих клавиш (например, Ctrl+C) если текущая раскладка не английская

1.3. Постановка задачи

Так как проектируемое программное средство должно быть заточено под создание синтаксических диаграмм, необходимо создать максимально удобный дизайн взаимодействия с пользователем (UX). Программа должна уметь сама рисовать мелкие элементы синтаксической диаграммы, такие как стрелки, диагональные скосы и т.д., основываясь на уже нарисованном пользователем изображении.

Чтобы программу можно было считать векторным редактором, должны быть реализованы следующие функции:

- Рисование с помощью установленных примитивных фигур;
- Изменение размеров, положения нарисованных фигур (редактирование изображения)
- Изменение размеров холста;
- Масштабирование изображения;
- Сохранение исходников изображения в типизированный файл с возможностью дальнейшего использования;
- Открытие типизированного файла с исходниками;
- Экспорт изображения в векторный формат (svg);
- Экспорт изображения в растровые форматы (bmp, png);
- Операции копирования, вставки и отмены действия.
- Переключение языка (русский/английский)

Также в программе должна присутствовать справка, включающая в себе инструкцию по использованию, описание программы и т.д.

В качестве языка программирования выбран язык Delphi, так как программа курса предмета ОАиП включает в себя данный язык, а также потому, что я хорошо освоил язык, выполняя лабораторные работы.

2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1. Представление векторного изображения в формате SVG

Векторная графика – это способ представления изображений и объектов, основанный на математическом описании элементарных географических объектов.

Одним из самых распространенных форматов файлов векторной графики является формат SVG. Формат SVG предназначен для описания двумерной векторной и смешанной графики в текстовом формате XML.

Структура документа^{[2][3]}:

- Первая строка – стандартный XML заголовок с указанием версии, кодировки.

Пример:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

- Вторая и третья строка – заголовок DOCTYPE, определяющий тип документа.

Пример:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"  
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

- Четвертая строка – корневой элемент документа с указанием пространства имен SVG

Пример:

```
<svg version="1.1"  
baseProfile="full"  
xmlns="http://www.w3.org/2000/svg"  
xmlns:xlink="http://www.w3.org/1999/xlink"  
xmlns:ev="http://www.w3.org/2001/xml-events"  
width="100%" height="100%">
```

- Далее идет остальной текст документа, завершающийся закрытием тега </svg>

Отрисовка основных фигур:

- **Описание путей**

Позволяет задать любую фигуру, описывая путь от начальной точки до конечной через промежуточные координаты. Строка с данными задается атрибутом **d** тега **path** и содержит команды, закодированные набором букв и чисел. Буквы – обозначают тип команды. Наиболее простые – М (*англ. moveto – переместить*), L (*англ. lineto – нарисовать линию*). Цифры, чаще всего, содержат координаты точек по осям X и Y.

Пример: линия из точки (100,100) в точку (100,200)

```
<path fill="none" stroke="black" d="M 100 100 L 100 200" />
```

- **Прямоугольник**

Строка задается 4-мя основными атрибутами тега **rect**: координаты X,Y левой верхней точки (Атрибуты x, y), высота и ширина (Атрибуты height и width соответственно).

Пример:

```
<rect fill="white" x="400" y="600" width="300" height="200" />
```

- **Окружность**

Строка задается 3-мя основными атрибутами тега **circle**: координаты центра (Атрибуты cx, cy), радиус (Атрибут r).

Пример:

```
<circle cx="200px" cy="200px" r="104px" fill="red"/>
```

- **Вывод текст**

Выводимый текст заключается в тег **text**, в котором в качестве атрибутов задаются свойства. Элементарные свойства для вывода текста – координаты левой верхней точки текста (атрибуты x, y).

Пример:

```
<text x="30" y="12" >Syntax Diagrams Editor</text>
```

Каждому тегу можно задать дополнительные свойства, описания и примеры которых находятся в документации по формату SVG.

2.2. Описание функциональности ПС

Программное средство должно представлять переключение режимов рисования путем нажатия соответствующей иконки в ToolBar (По умолчанию пользователю предлагается режим редактирования).

Режим рисования может принимать один из следующих значений:

- Рисование прямоугольных фигур с текстом
 - Заголовок синтаксической диаграммы
 - Метаварiable
 - Метаконстанта
- Рисование линий
- Запрет рисования (Редактирование)

В зависимости от выбранного режима пользователю должны представляться следующие возможности взаимодействия:

- Режим рисования прямоугольных фигур:
 - При клике по полотну программное средство должно отобразить введенный пользователем текст на канвасе по координатам клика.
- Режим рисования линий:
 - Первый клик левой кнопкой мыши по полотну должен активировать режим рисования линий. Каждое следующее нажатие левой кнопки мыши должно добавлять новую точку и соединить ее с предыдущей точкой данной линии. Нажатие правой кнопки мыши должно прекратить рисование линий.
- Режим редактирования:
 - При перемещении мыши курсор меняется в зависимости от того, на какую область фигуры он наведен:
 - Вершина фигуры
 - Сторона фигуры
 - Центр фигуры
 - При зажатии мыши в этом режиме, фигура должна редактироваться по следующему принципу
 - Зажата в центре – при перемещении курсора перемещается вся фигура.
 - Зажата на вершине – при перемещении курсора перемещается вершина и стороны, которым принадлежит вершина.

- Зажата сторона фигуры – при перемещении курсора перемещается сторона.
- Клик по фигуре должен выделять фигуру, по которой был произведен клик. При нажатии клавиши «delete» должна удаляться выделенная фигура. Сочетания «Ctrl + C» должно помещать в специальный «программный буфер обмена» выделенную фигуру. Сочетания «Ctrl + V» - извлекать из буфера фигуру и отображать на полотне. Сочетание «Ctrl + Z» должно «откатывать» на предыдущее действие, то есть отменять последнее изменение. Если фигура (не линия) выделена, то если изменить текст в отведенном текстовом поле и нажать «enter», текст внутри фигуры должен измениться на введенный.

Также программа должна предоставлять взаимодействие с меню. В зависимости от того, по какому элементу меню был произведен клик, программа должна уметь:

- Создавать новый файл.
- Сохранять исходный файл.
- Открывать исходный файл.
- Экспортировать в векторные и растровые форматы.
- Изменять размеры полотна.
- Включать/выключать режим «примагничивания».
- Изменять язык интерфейса

2.3. Спецификация функциональных требований.

Среди функциональных требований есть «Отображение фигур на полотне».

Спецификация данной функции может иметь следующий вид:

- Прямоугольные фигуры имеют прозрачный фон и обводку, вершины обозначаются маленькими квадратами. Текст вписан в прямоугольник с вертикальным и горизонтальным центрированием.
- Линии должны проходить через все точки, заданные пользователем путем нажатия по полотну, в заданном пользователем порядке (порядок задается порядком нажатия).
- В конце каждой линии проводится стрелка.

- Если первые две точки линии описывают вертикальный отрезок, то проводится дополнительный диагональный отрезок (см. рисунок 2.1).

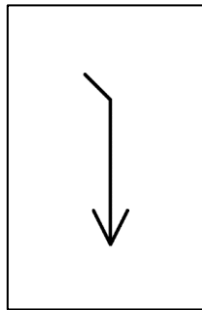


Рисунок 2.1 – дополнительный диагональный отрезок в начале вертикального участка линии.

- Если последние две точки линии описывают вертикальный отрезок, а перед этим присутствовал и горизонтальный участок линии, то проводится дополнительный диагональный отрезок (см. рисунок 2.2).

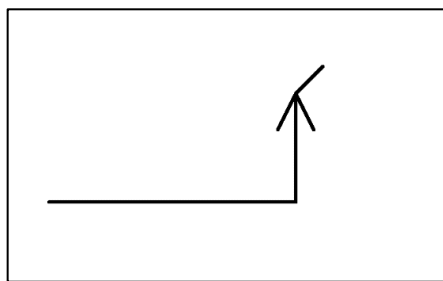


Рисунок 2.2 – дополнительный диагональный отрезок в конце вертикального участка линии.

- Если линия начинается вертикальным участком, потом содержит горизонтальный участок и заканчивает вертикальным участком, по середине горизонтального участка ставится стрелка (см. рисунок 2.3).

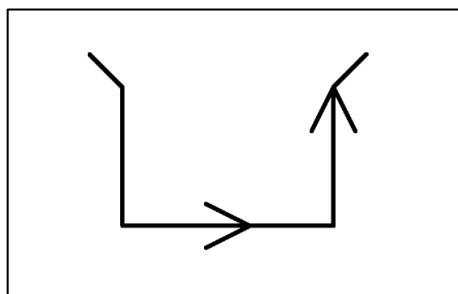


Рисунок 2.3. – стрелка по середине горизонтальной линии

- Если точка начала горизонтальной линии принадлежит вертикальному участку другой линии, то проводится дополнительный диагональный отрезок (см. рисунок 2.4).

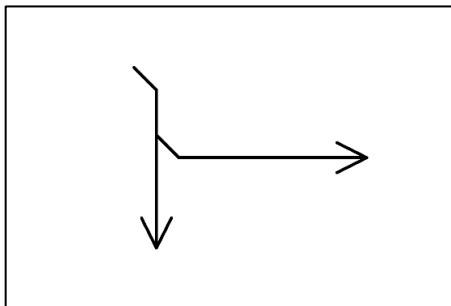


Рисунок 2.4. – дополнительный диагональный отрезок в начале горизонтального участка линии

- Если точка конца горизонтальной линии принадлежит вертикальному участку другой линии, то проводится дополнительный диагональный отрезок (см. рисунок 2.5.).

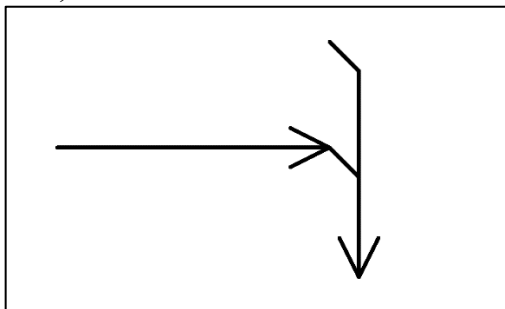


Рисунок 2.5. – дополнительный диагональный отрезок в конце горизонтального участка линии

- Пользователь должен иметь возможность изменять размер полотна
- Пользователь должен иметь возможность изменить масштаб изображения

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1. Проектирование динамических структур данных

В первую очередь, необходимо хранить фигуры. Так как пользователь будет постоянно добавлять и удалять фигуры, напрашивается использование динамических списков. Однако нужно предусмотреть следующие особенности:

- Присутствуют различные фигуры с разным способом описания.
- У линий пользователь может динамически добавлять точки.
- Информативную часть нужно сохранять в файл, а в файле необходимо как-либо хранить сохраненное разрешение полотна и проверять файл на валидность.

Поэтому было принято решение использовать в качестве записи информативной части списка запись с вариантной частью. Структура представлена на рисунке 3.1.

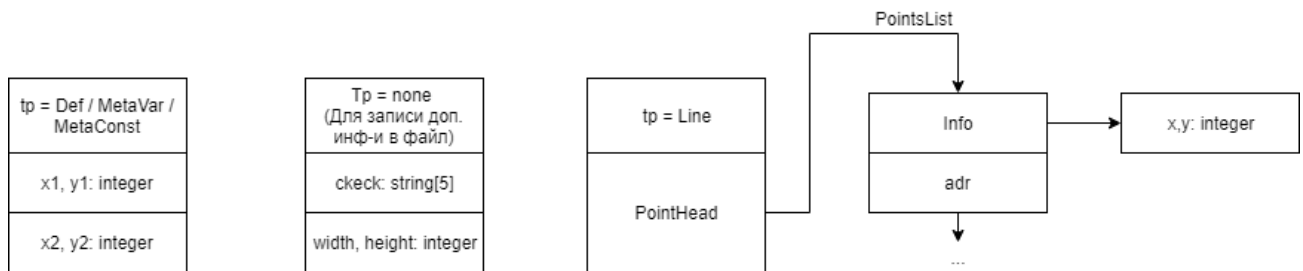


Рисунок 3.1 – структура основного списка программы

Так как фигура линии хранит в себе указатель на список точек линии, нужно придумать, как сохранять это в файл. Для этого я решил создать сделать отдельное представление записи для линий именно для сохранения в файл. При такой структуре координаты преобразуются в специальный текстовый формат, а при открытии файла, декодируются и преобразуются обратно – в список точек. Структура записи линии для записи в файл представлена на рисунке 3.2.

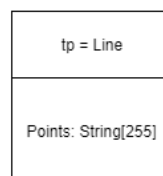


Рисунок 3.2. - структура записи линии для записи в файл

Для того, чтобы предусмотреть функционал отмены изменений, необходимо где-то хранить каждое изменение. Для этого лучше всего подходит

стек, т.е. каждое изменение будет помещаться в вершину стека. При отмене изменений будет извлекаться вершина и вершина будет перемещаться к предыдущему элементу, то есть структура будет иметь вид, приведенный на рисунке 3.3.

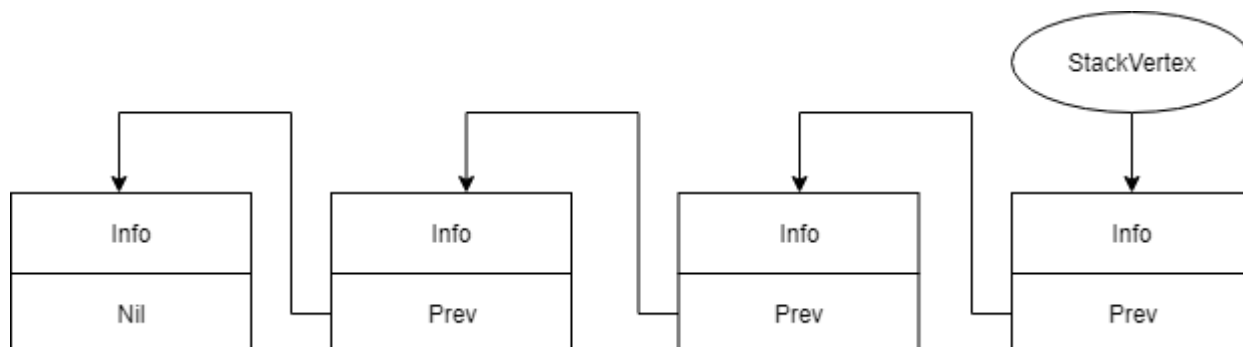


Рисунок 3.3 – структура стека изменений

Для того, чтобы не допустить извлечения самого последнего элемента, тем самым, потери указателя на вершину (т.к. он будет nil), я решил сделать переменную, отвечающую за «вариантность» информационной части стека специально типа и запретить изменять записи с таким типом.

Необходимо предусмотреть сохранение следующих изменений:

- Удаление фигуры
Запись должна содержать:
 - Ссылку на удаленную фигуру
 - Ссылку на фигуру, расположенную в списке перед удаленной.
- Добавление точки линии:
Запись должна содержать:
 - Ссылку на фигуру линии
 - Ссылку на добавленную точку в списке точек данной линии
- Добавление фигуры
Запись должна содержать:
 - Ссылку на добавленную фигуру
- Перемещение фигуры/изменение размеров фигуры (кроме линий)
Запись должна содержать:
 - Ссылку на фигуру, которую переместили
 - Предыдущее значение записи информативной части фигуры (предыдущие координаты)
- Перемещение линии / перемещение точки линии
Запись должна содержать:

- Ссылка на фигуру линии, которую переместили
- Копию старых координат всех точек, сохраненных в текстовом формате (Используя тот же принцип, как при сохранении в файл).
- Изменение текста фигуры (Кроме линий)
Запись должна содержать:
 - Ссылка на фигуру
 - Предыдущий текст
- Изменение размеров полотна
Запись должна содержать:
 - Предыдущие значения ширины и высоты полотна

А также должна быть запись, сигнализирующая конец стека (используется только для единственной записи в конце стека и больше нигде).

После анализа тех изменений, которые необходимо предусмотреть, мною было принято решение вынести в общую часть информативной записи стека ссылку на фигуру, с которой произошли манипуляции, а остальное сохранять в вариантную часть.

3.2. Разработка алгоритма реакции на клик пользователя по полотну

После того, как пользователь кликнет по полотну, необходимо обработать данное событие, и, если текущий режим не является режимом редактирования, добавить фигуру. Также необходимо учесть, что должен быть фиксированный шаг «сетки» полотна для большего удобства пользователя. То есть, после клика координаты должны «округляться» до ближайшей вершины невидимой сетки. Схема алгоритма представлена на рисунке 3.4.

Режимы рисования и редактирования я решил записывать в отдельные перечислимые типы. (Для рисования: Рисование текстовой фигуры, рисование фигуры, нет рисования, для редактирования: перемещение фигуры, перемещение вершины, перемещение стороны).

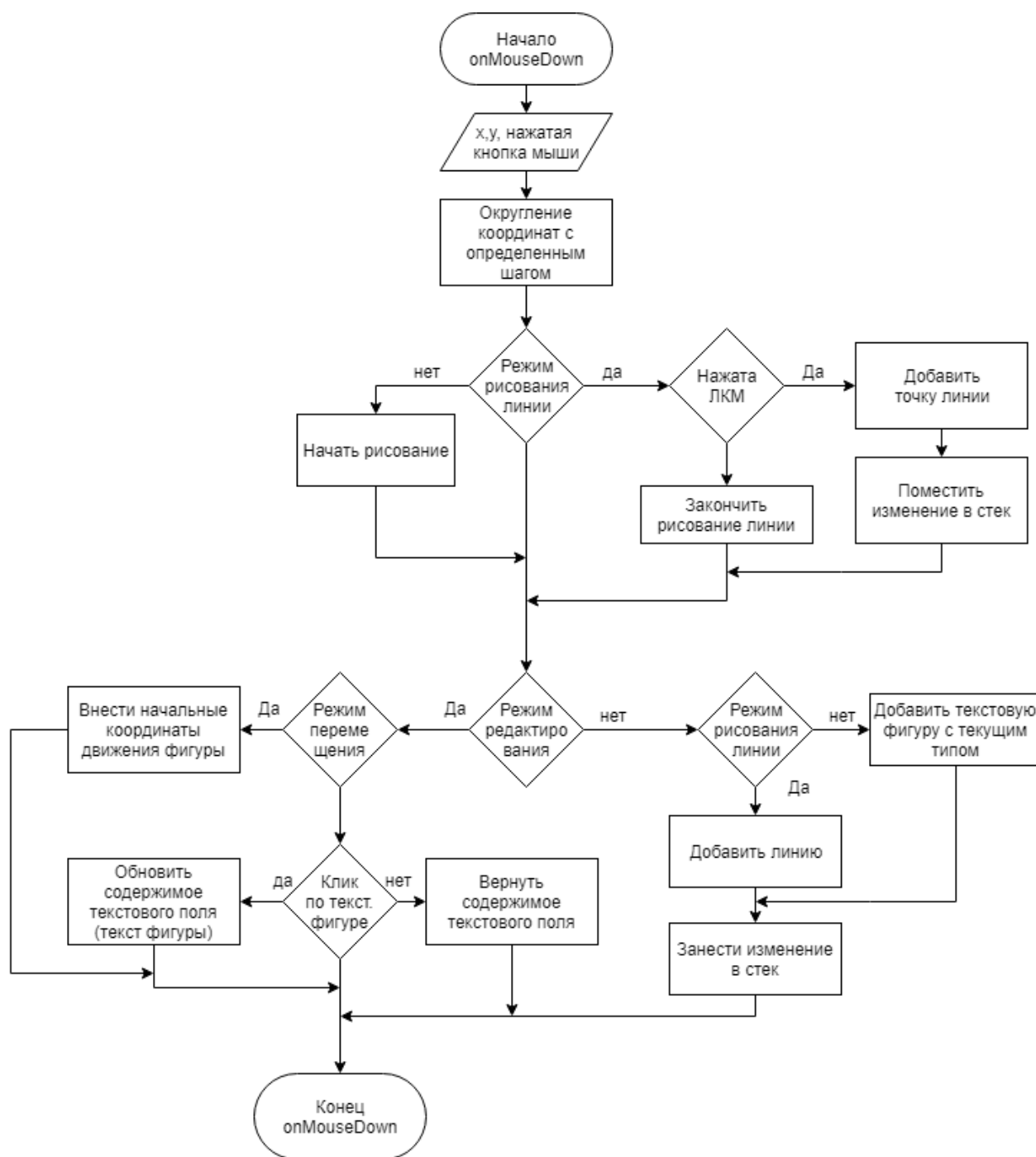


Рисунок 3.4 – схема алгоритма реакции на клик пользователя по полотну

3.3. Разработка алгоритма перемещение точки внутри линии

Если алгоритм перемещения фигуры / вершины текстовой фигуры / стороны текстовой фигуры / целой линии элементарен, из-за чего я решил даже

не описывать его в данном разделе, то алгоритм перемещения одной точки прямой имеет «подводные камни». Если движется одна линия, может двигаться еще неограниченное количество точек, ведь линии всегда должны быть параллельны одной из осей. Чтобы избежать случаев, когда участок линии проходит под углом к обоим осям, необходимо разработать правильный алгоритм перемещения точки внутри линии.

Алгоритм должен принимать на вход ссылку на «голову» списка фигур, координаты до движения точки и координаты после движения точек. В процессе работы, алгоритм должен изменить координаты других точек линии так, чтобы перемещенная точка осталась на месте и внутри линии не было участков, направленных под углом к осям.

Идея алгоритма: найти область линии, все точки которой имеют либо координату x , либо координату y , равную старому значению координаты перемещаемой точки. При этом все точки области должны идти подряд и в области не должно содержаться ни одной точки, не соответствующих данному условию. После нахождения области, нужно изменить координаты каждой точки внутри области. Пример поиска такой области приведен на рисунках 3.5а и 3.5б.

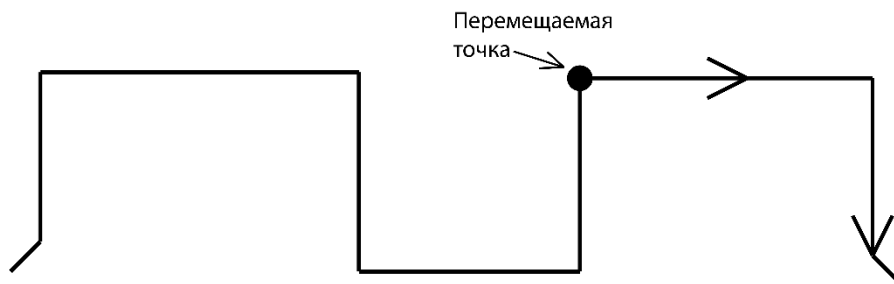


Рисунок 3.5а – исходная линия.

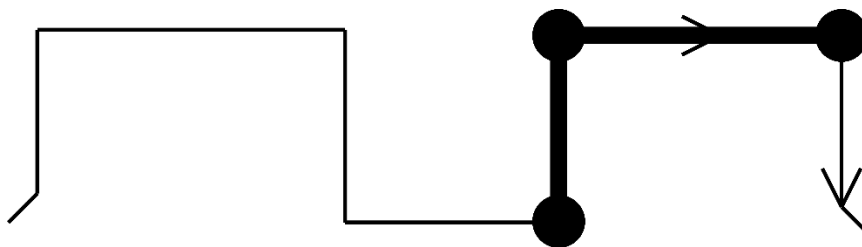


Рис 3.5б – найденная алгоритмом область

Схема данного алгоритма представлена на рисунке 3.6

Примечание: данный алгоритм перемещает только точки вокруг исходной. Координаты исходной точки должны быть изменены до применения алгоритма.

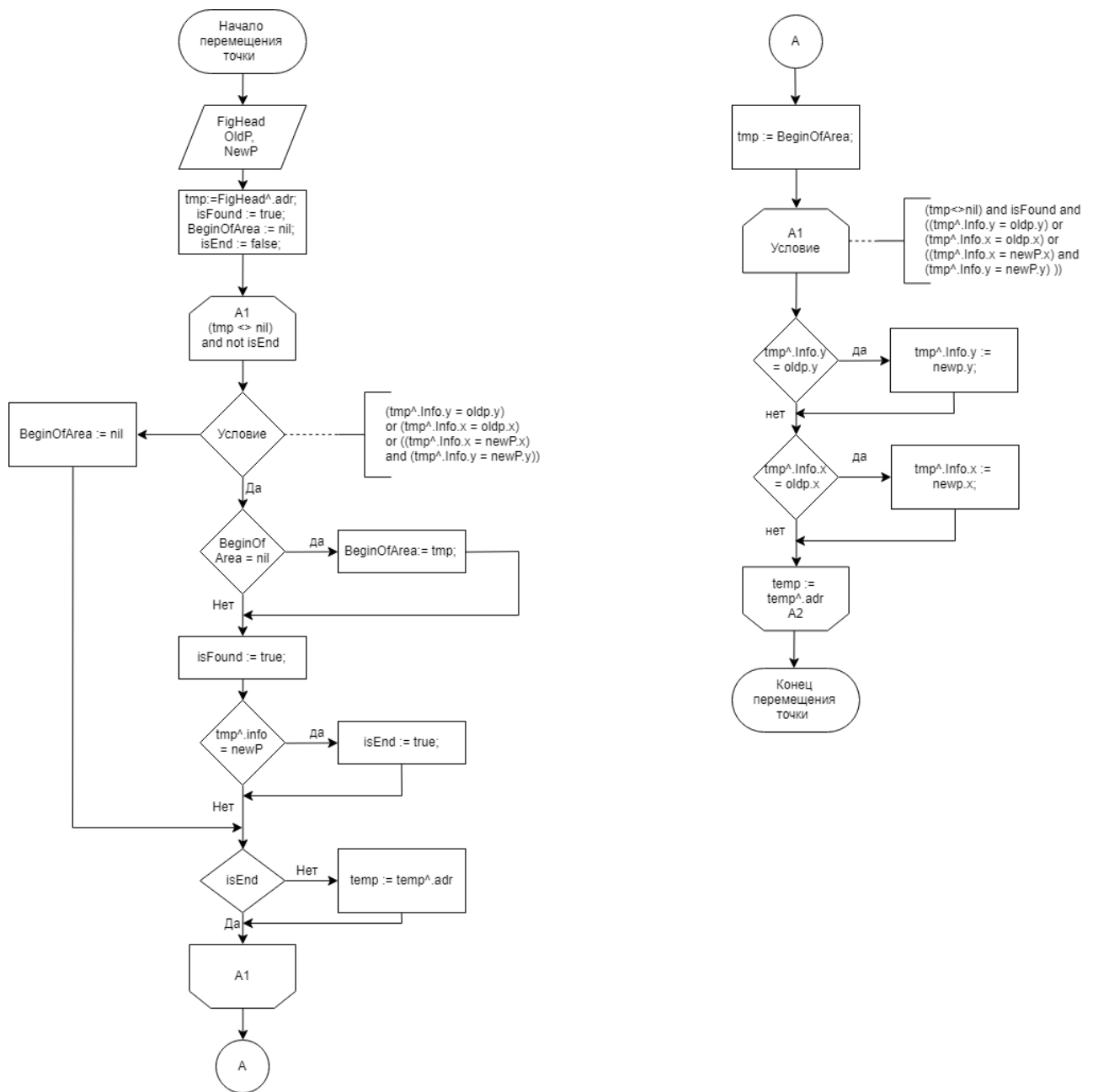


Рисунок 3.6 – схема алгоритма перемещение точки внутри линии

3.4. Разработка алгоритма «примагничивания фигур»

Чтобы пользователю было проще редактировать синтаксические диаграммы, я решил добавить функционал «примагничивания» фигур.

Требования к алгоритму:

- Если начальная/конечная точка линии находится очень близко к другой линии, координаты этой точки должны измениться так, чтобы точка стала принадлежать прямой, расположенной очень близко.
- Если начальная/конечная точка линии находится очень близко к текстовой фигуре, координаты этой точки должны измениться так, чтобы точка оказалась (см. рисунок 3.5):
 - В центре прямоугольной фигуры по оси OY
 - На краю прямоугольной фигуры по оси OX
- Если текстовые фигуры располагаются приблизительно на одном уровне по оси OY, они должны выравниваться по одной координате Y.

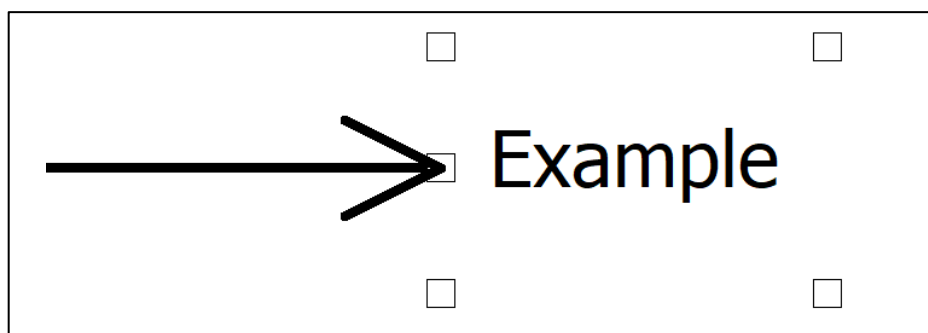


Рисунок 3.7 – пример положения точек после применения алгоритма «примагничивания» линии к текстовой фигуре.

В качестве значения, расстояние в сколько пикселей считать «близким», было принято вынести в отдельную константу для того, чтобы ее значение можно было изменить в любой момент.

3.4.1. Разработка алгоритма поиска линии вблизи точки

Так как линия представляет собой список точек, я решил вынести алгоритм поиска линии вблизи точки в отдельный алгоритм. Алгоритм должен на входе принимать ссылку на «голову» списка фигур и координату точки, вблизи которой надо найти линию. На выходе алгоритм должен вернуть точку, принадлежащую прямой, и которая будет наиболее близкая к исходной точке и ссылку на ближайшую точку списка точек прямой (nil если не найдено прямой поблизости).

Схема алгоритма представлена на рисунке 3.6.

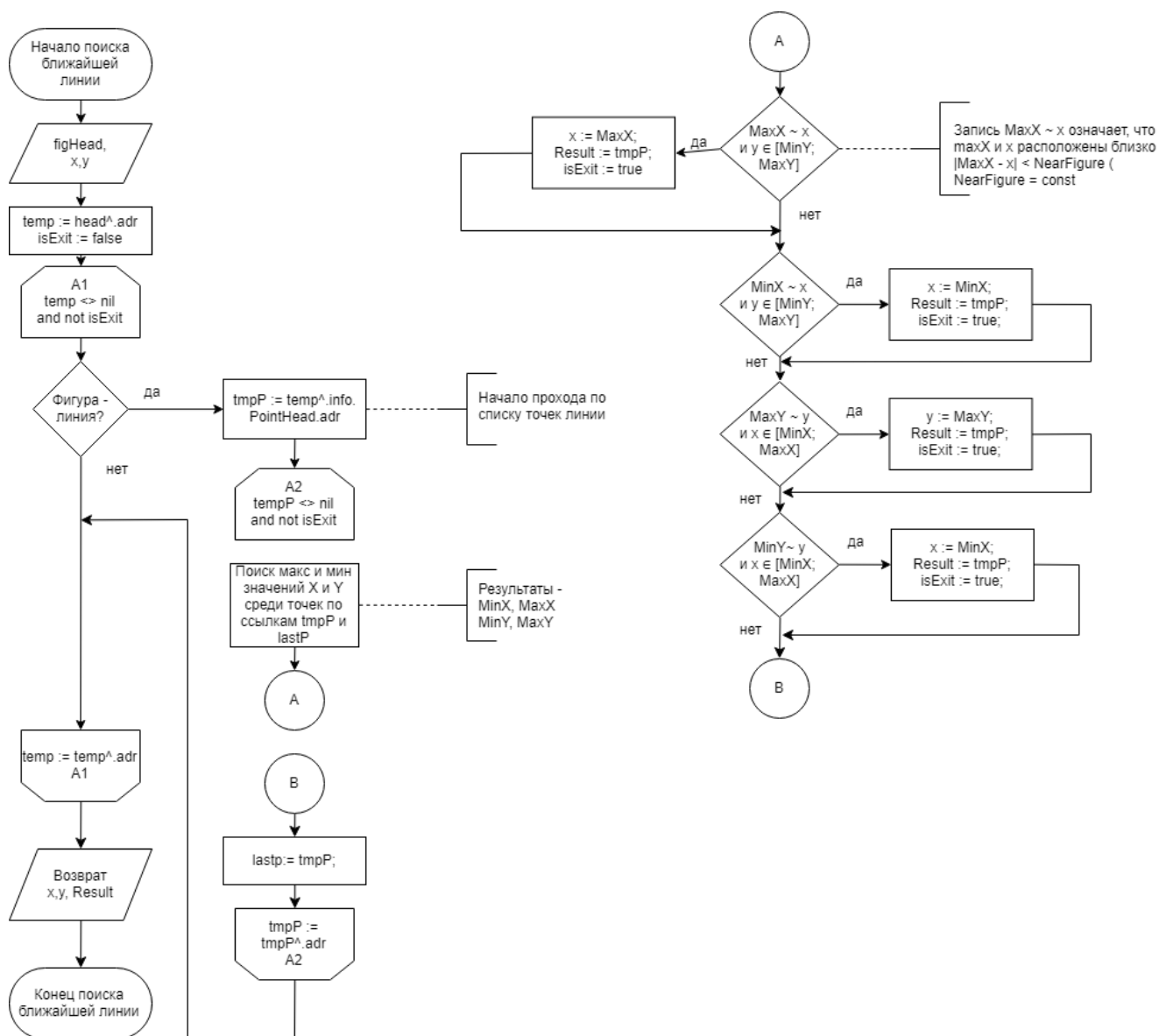


Рисунок 3.8 – схема алгоритма поиска ближайшей линии

3.4.2. Разработка алгоритма «примагничивания» линии к текстовой фигуре

Алгоритм должен принимать на входе ссылки на голову списка фигур и на точку определенной линии. Если алгоритм находит текстовую фигуру рядом с точкой, он меняет координаты этой точки и функция возвращает true. Иначе –

алгоритм ничего не меняет, и функция возвращает false. Схема алгоритма представлена на рисунке 3.7.

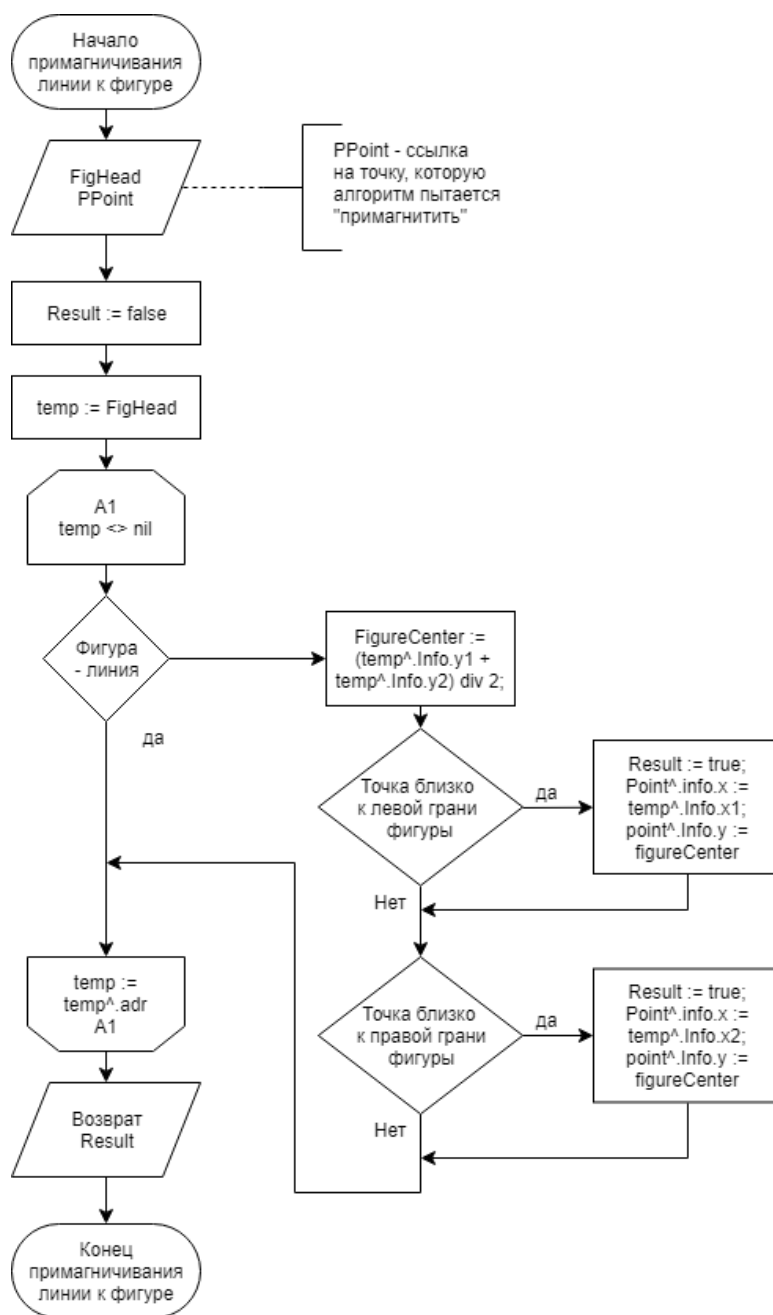


Рисунок 3.9 – Схема алгоритма «примагничивания» линии к текстовой фигуре

3.4.3. Разработка алгоритма «примагничивания» точки к линии

Замечание: Алгоритм должен перемещать только одну, самую ближайшую к точке линию. Остальные точки линии будут перемещаться ранее разработанным алгоритмом.

Схема алгоритма представлена на рисунке 3.8.

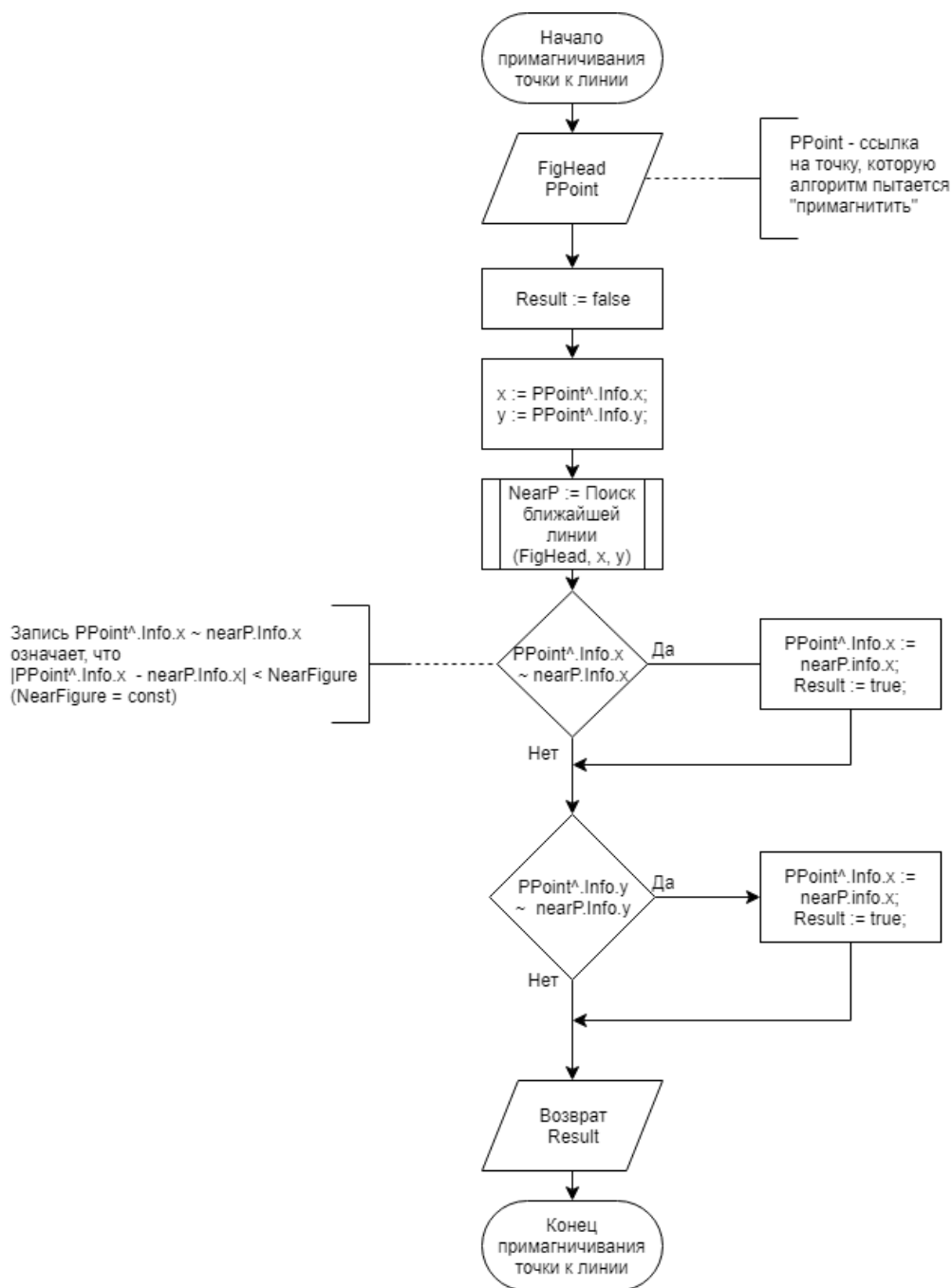


Рисунок 3.8. – схема алгоритма «примагничивания» точки к линии

3.4.4. Обобщение алгоритмов «примагничивания»

Схема обобщенного алгоритма примагничивания представлена на рисунке 3.9.

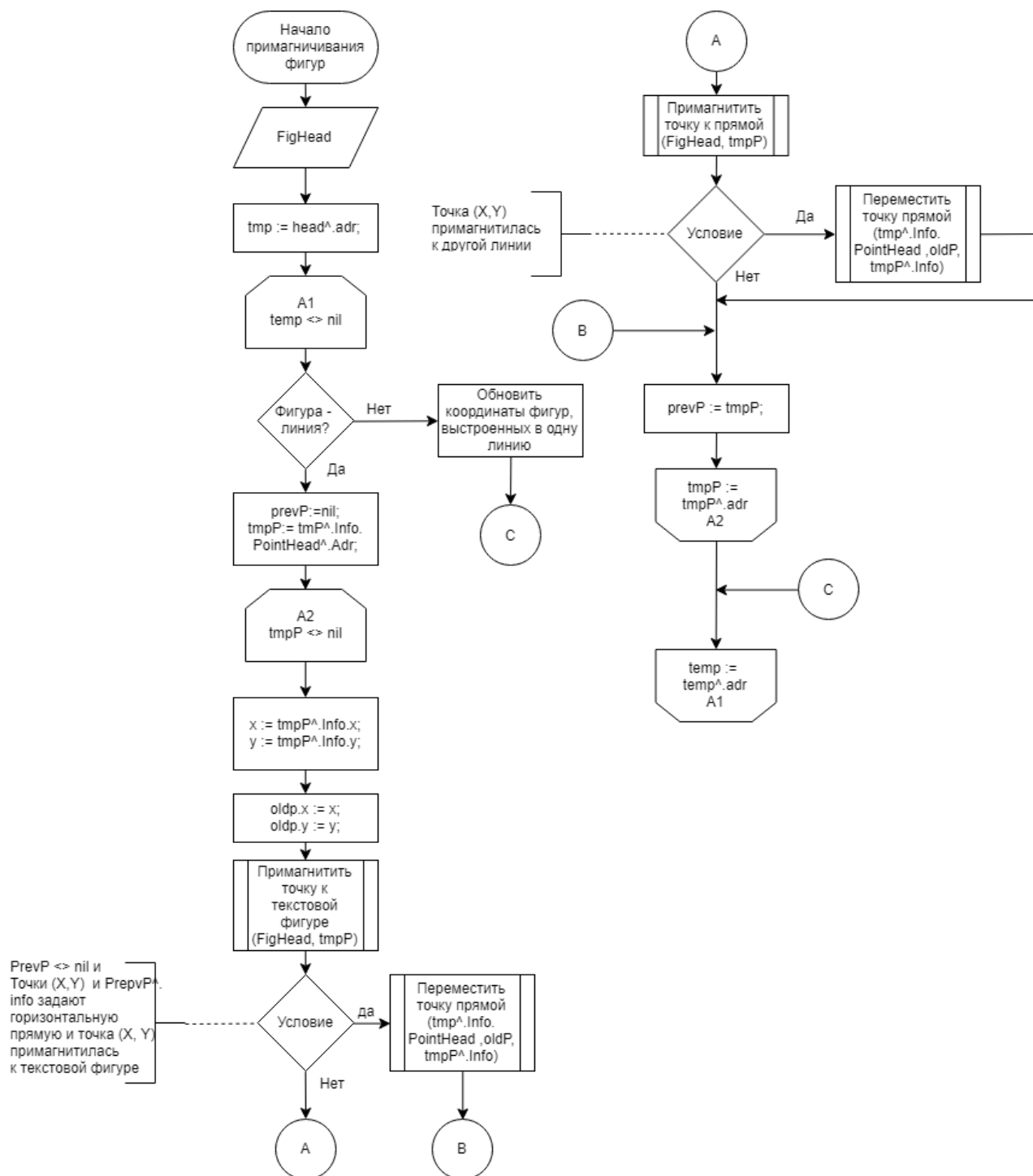


Рисунок 3.9 – общая схема алгоритма «примагничивания»

3.5. Разработка алгоритма отрисовки линий

Алгоритм отрисовки линий должен предусмотреть особенности построения синтаксических диаграмм, в том числе, достраивать части линий в зависимости от взаимного расположения фигур на полотне. Схема представлена на рисунке 3.10

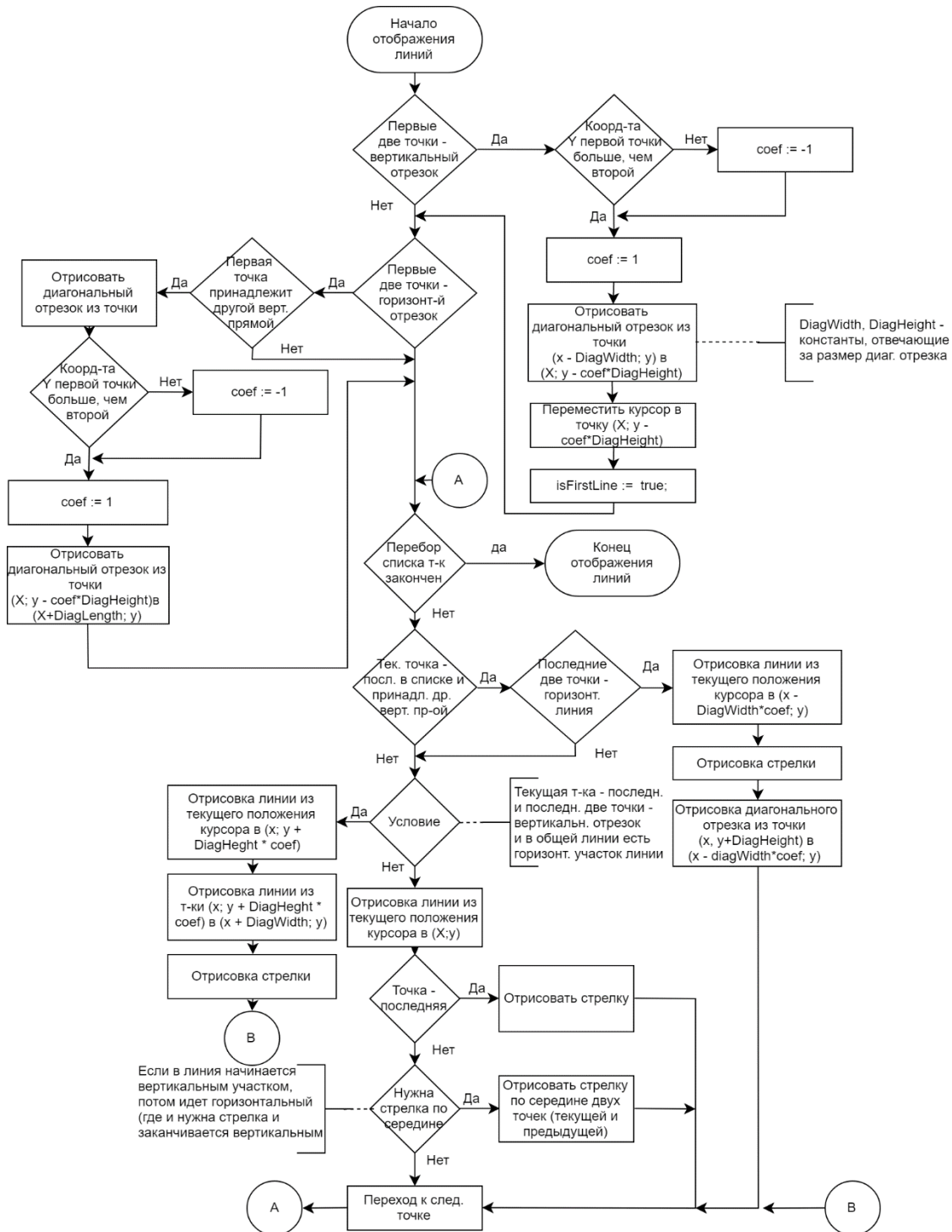


Рисунок 3.10 – Схема алгоритма отрисовки линий

3.6. Разработка алгоритм отрисовки фигур

Алгоритм отрисовки фигур является довольно простым, однако является одним из базовых в программном средстве. Схема алгоритма представлена на рисунке 3.11.

Примечание: Алгоритма отрисовки фигур (в т.ч. алгоритм отрисовки линий универсален как для отрисовки на Canvas, так и для отрисовки в SVG-формате.

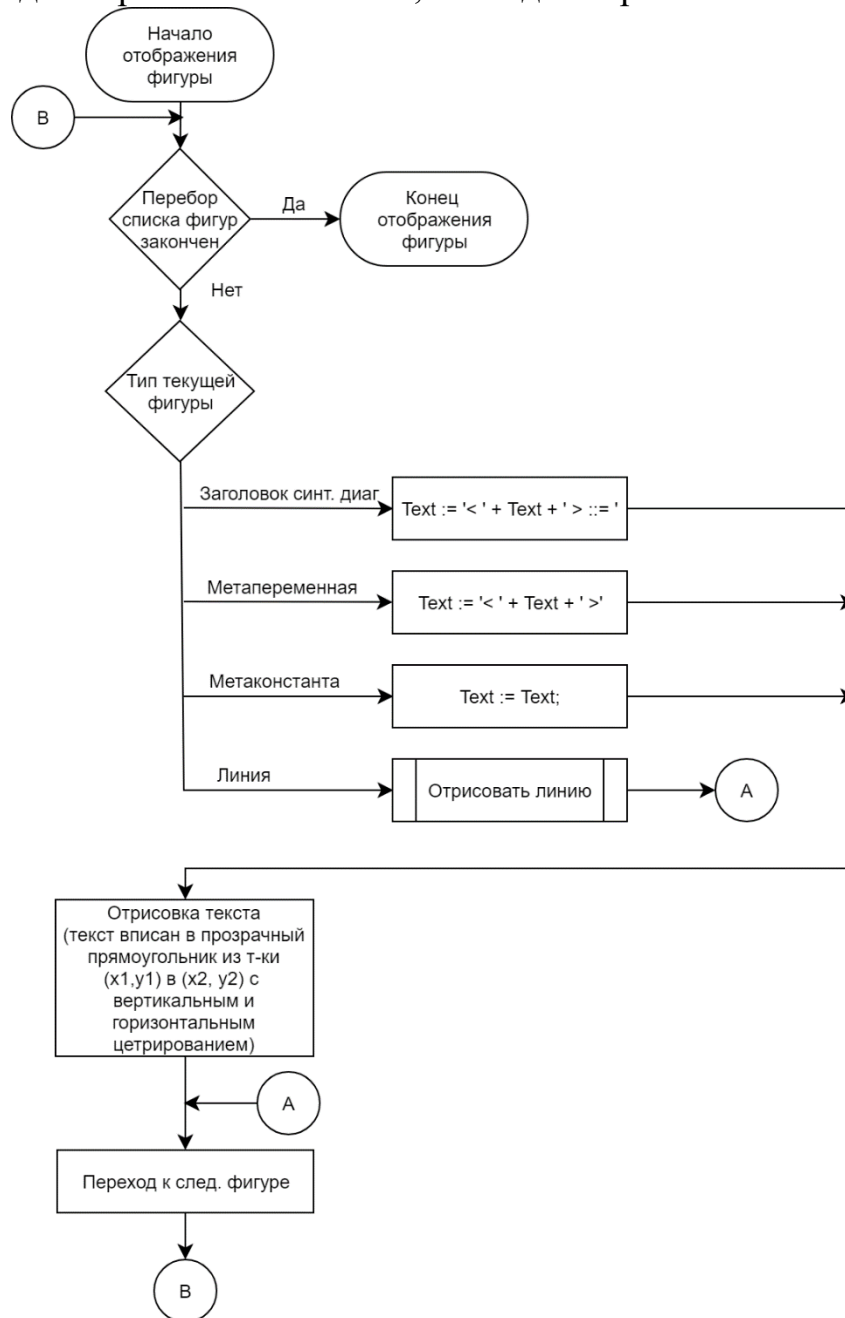


Рисунок 3.11 – схема алгоритма отрисовки фигур.

4. СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА

4.1. Взаимодействие между формами

В программном средстве используется три формы:

- Главная форма (Все основные действия происходят на ней).
- Форма изменения размеров полотна.
- Форма для отображения HTML страниц справки.

Схема взаимодействия форм отображена на рисунке 4.1.

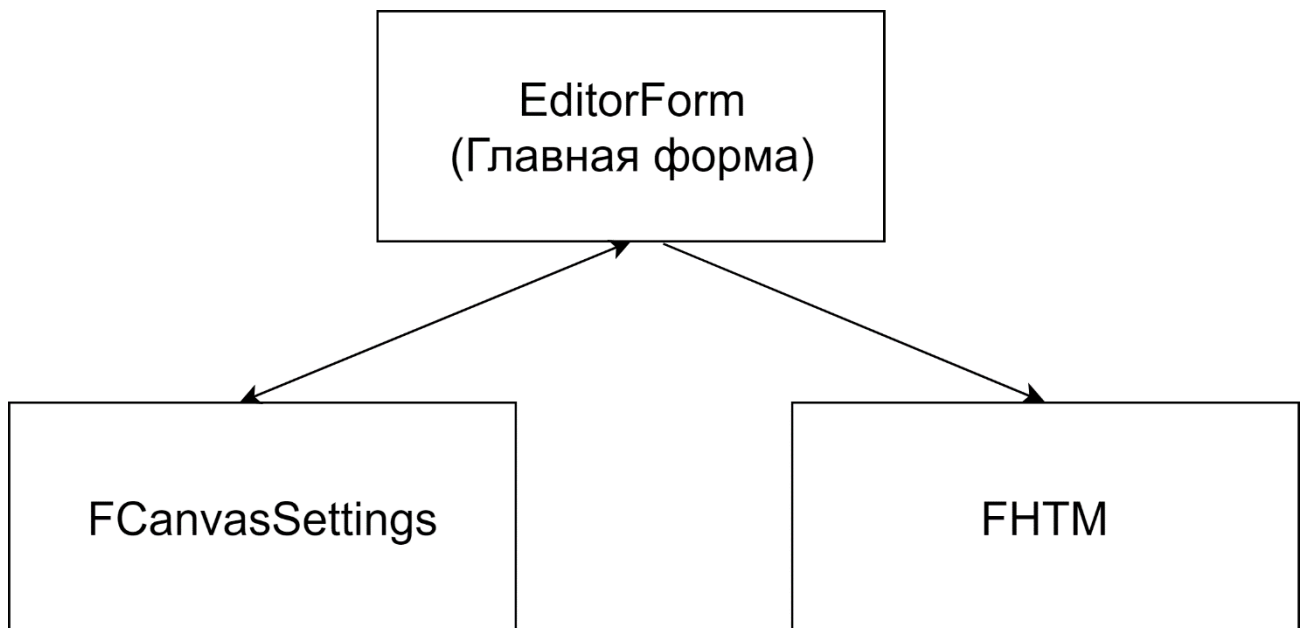


Рисунок 4.1 – схема взаимодействия форм

Форма FHTML просто отображает HTML код, заданный в ресурсах приложения и отображает его для пользователя. (см. рисунок 4.2).

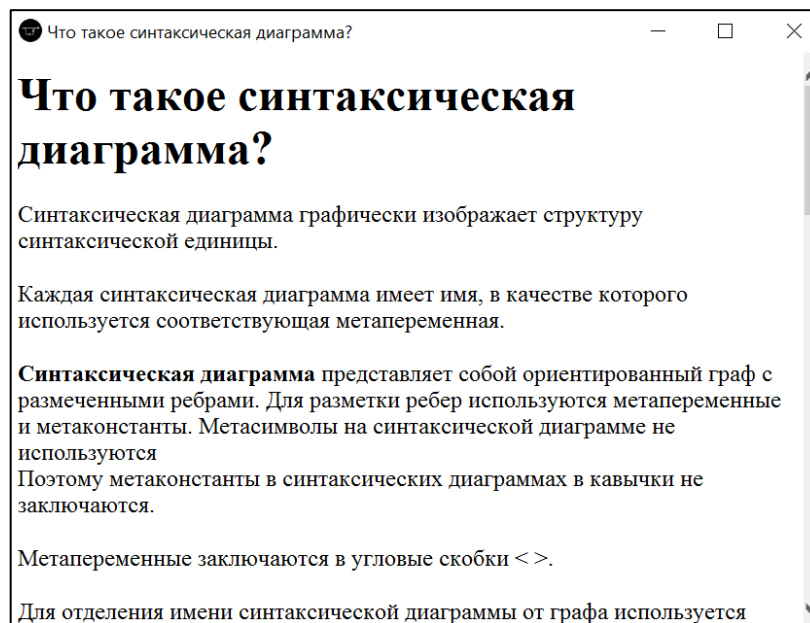


Рисунок 4.2 – скриншот формы FHTML

Форма FCanvasSettings предоставляет окно для изменения размеров полотна. В процедуру отображения данной формы из главной формы передается текущий размер Canvas. После закрытия формы с возвратом mrOk, форма возвращает новые размеры через var-параметры процедуры. (см. рисунок 4.3).

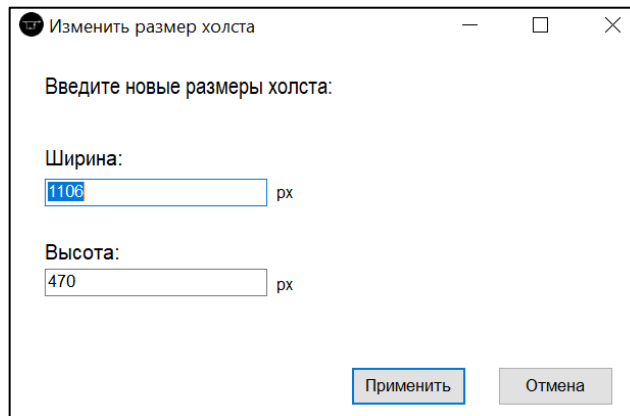


Рисунок 4.3 – скриншот формы FCanvasSettings

Форма EditorForm представляет собой главную форму с меню, tool bar и областью рисования. (см. рисунок 4.4).

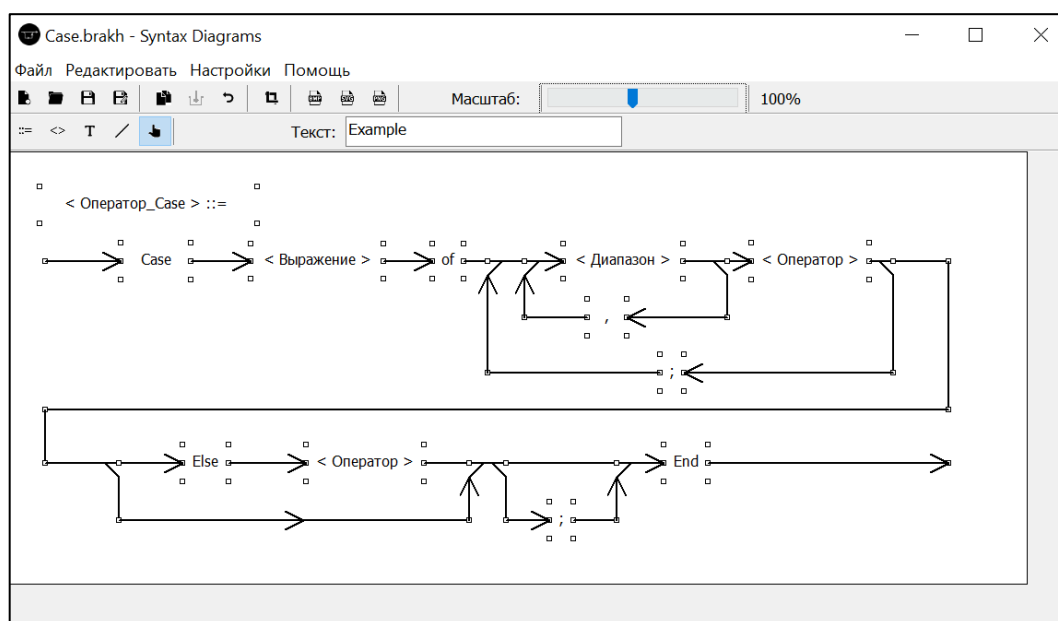


Рисунок 4.4 – скриншот главной формы.

Схема работы всей программы представлена в приложении 1. Текст программы представлен в приложении 2.

4.2. Структура модулей программы

В процессе разработки программного средства мною было выделено 11 модулей. Некоторые из модулей можно выделить в группы.

Модули форм:

- Main – модуль главной формы
- FCanvasSizeSettings – модуль формы FCanvasSettings
- FHTMLView – модуль формы FHTML

Модули «Представления» (View):

- View.Canvas – модуль с набором процедур и функции представления фигур на канвасе.
- View.SVG – модуль с набором процедур и функции представления фигур в SVG.

Модули «Модели» (Model):

- Model – модуль с набором основных процедур и функций модели.
- Model.Lines – модуль с набором процедур и функций взаимодействия модели с линиями и точками.
- Model.UndoStack – модуль с набором процедур и функций взаимодействия модели со «Стеком изменений»

Модули с данными:

- Data.Types – основные типы, необходимые для других модулей.
- Data.InitData – инициализированные данные (константы, resourcestring), необходимые для других модулей.

4.3. Описание модуля Main

Модуль Main является юнитом главной формы. Содержание модуля – методы класса формы, в основном – обработчики событий.

Описание основных подпрограмм, описанных в модуле Main приведено в таблице 4.1. (В таблице приведены только самые основные подпрограммы, многие обработчики событий, в т.ч. событий ActionList не включены в таблицу).

Таблица 4.1. – основные подпрограммы модуля Main

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
changeEditorText	Изменение содержимого текстового поля	procedure TEditorForm.changeEditorText(newtext: string);	newtext	Новый текст
tbSelectScaleChange	Обработчик события изменения ползунка масштаба	procedure TEditorForm.tbSelectScaleChange(Sender: TObject);	Sender	Объект, который сгенерировал событие
useScale	Масштабирование координат (возвращает новые координаты в var-параметры)	procedure TEditorForm.useScale (var x, y: integer);	x	Координата X
			y	Координата Y
pbMainMouseDown	Обработка нажатие мыши. Внутри обработчика – в зависимости от режима добавляется линия/добавляется фигура/редактируется фигура	procedure TEditorForm.pbMainMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);	Sender	Объект, который сгенерировал событие
			Button	Кнопка мыши
			X	Координата X
			Y	Координата Y
getFigureHead:PFigList;	Return figure head	function TEditorForm.getFigureHead:PFigList;		

pbMainMouseMove	Обработка нажатие мыши	procedure TEditorForm.pbMain MouseMove(Sender: TObject; Shift:TShiftState; X, Y: Integer);	Sender	Объект, который сгенерировал событие
			Shift	
			X	
			Y	
pbMainMouseUp	Обработка нажатие мыши	procedure TEditorForm.pbMain MouseUp(Sender: TObject; Button:TMouseButton ; Shift: TShiftState; X, Y: Integer);	Sender	Объект, который сгенерировал событие
			Button	кнопка
			Shift	
			X	
			Y	
pbMainPaint	Обработчик события перерисовки. Внутри – перерисовываются фигуры	procedure TEditorForm.pbMain Paint(Sender: TObject);	Sender	Объект, который сгенерировал событие
clearScreen;	Очистка экрана	procedure TEditorForm.clearScreen;		
analyseParams: string;	Анализ входных параметров (была ли программа открыта путем открытия файла и сходниками синт. диаграммы, возвращает путь к файлу)	function analyseParams: string;		
FormCreate	Событие при создании формы	procedure TEditorForm.FormCreate(Sender: TObject);	Sender	Объект, который сгенерировал событие
FormKeyDown	Обработка нажатий клавиши: Удаление на клавишу Delete, применение нового текста на Enter и изменение масштаба на Ctrl + '+' и Ctrl + '-'	procedure TEditorForm.FormKeyDown(Sender: TObject; var Key:Word; Shift: TShiftState);	Sender	Объект, который сгенерировал событие
			Key	Код клавиши
ExtractFileNameEx	output: input brakh	function ExtractFileNameEx(FileName:string):string;	FileName	Название
openFile	Открытие файла	function TEditorForm.openFile (mode: TFileMode):string;	mode	Режим(расширение, которое надо открыть)

saveBMPFile;	Экспорт в BMP	procedure TEditorForm.saveBMPFile;		
changeCanvasSize	Изменение размеров полотна	procedure TEditorForm.changeCanvasSize(w,h: Integer; flag:Boolean = true);	w	Новая ширина
			h	Новая высота
			flag	Флаг (записывать ли изменение в стек)
newFile;	Создание нового файла	procedure TEditorForm.newFile;		
saveFile	Сохранение файла	function TEditorForm.saveFile (mode: TFileMode):string;	mode	Режим, содержащий расширение, которое надо сохранить
savePNGFile;	Экспорт в PNG	procedure TEditorForm.savePNGFile;		
saveBrakhFile:boolean;	Сохранение исходного файла	function TEditorForm.saveBrakhFile:boolean;		
saveSVGFile;	Экспорт в SVG	procedure TEditorForm.saveSVGFile;		

4.4. Описание модуля FCanvasSizeSettings

Модуль FCanvasSizeSettings является юнитом формы FCanvasSize. Его предназначение – вернуть новые введенные пользователем размеры формы. Описание основных подпрограмм, описанных в модуле FCanvasSizeSettings приведено в таблице 4.2.

Таблица 4.2. – основные подпрограммы модуля FCanvasSizeSettings

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
ControlsToItem	Процедура возвращает в var-параметры введенные пользователем данные	procedure TFCanvasSettings.ControlsToItem(var w, h: integer);	w	
			h	

showForm	Отображение формы с отображением текущих размеров	function TFCanvasSettings.showForm(var w,h:integer):TModalResult;	w	Текущая ширина
			h	Текущая высота

4.5. Описание модуля FHTMLView

Модуль FHTMLView является юнитом формы FHTML. Его предназначение – отобразить HTML страницу со справкой, находящуюся в реурсах. Описание основных подпрограмм, описанных в модуле FHTMLView приведено в таблице 4.3.

Таблица 4.3. – основные подпрограммы модуля FHTMLView

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
WMMouseActivate	Свой обработчик события нажатия ПКМ.	procedure TFHtml.WMMouseActivate(var Msg: TMessage);	Msg	Сообщение windows
showHTML	Отображение формы с выводом нужного HTML файла в TWebBrowser	procedure TFHtml.showHTML(title, htmlres: WideString);	title	Заголовок окна
			htmlres	Название ресурса HTML файла

4.6. Описание модулей «Модели»

4.6.1. Описание модуля Model

Модуль Model является основным юнитом модели. В модуле присутствуют подпрограммы взаимодействия с фигурами, функции отмены изменений и т.д. Описание основных подпрограмм, описанных в модуле Model приведено в таблице 4.4.

Таблица 4.4 – основные подпрограммы модуля Model

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
createFigList	Создание списка фигур	procedure createFigList(var head: PFigList);	head	Ссылка на голову списка фигур
copyFigure	Копирование фигуры	procedure copyFigure(head: PFigList; copyfigure:PFigList);	head	Ссылка на голову списка фигур
			copyfigure	Ссылка на фигуру, которую надо скопировать
addFigure	Добавление новой фигуры и возврат ссылки на нее	function addFigure(head: PFigList; x,y: integer; ftype: TType; Text:String = 'Kek'):PFigList;	head	Голова
			x	Координата X, куда нужно добавить фигуру
			y	Координата Y, куда нужно добавить фигуру
			ftype	Тип фигуры
			Text	Текст фигуры
getClickFigure	Функция озвращает фигуру, по которой был клик	function getClickFigure(x,y:integer; head: PFigList):PFigList;	x	Координата X клика
			y	Координата Y клика
			head	Ссылка на голову списка фигур
removeFigure	Функция выполняет логическое удаление фигуры и возвращает ссылку на предшествующую удаленной фигуру фигуру (для помещения в стек изменений)	function removeFigure(head: PFigList; adr: PFigList):PFigList;	head	Ссылка на голову списка фигур
			adr	Адрес фигуры, которую надо удалить
removeAllList	Удаление списка фигур	procedure removeAllList(head:P FigList);	head	Ссылка на голову списка фигур
ChangeCoords	Изменение координат фигуры (перемещение/изменение размеров и тд)	procedure ChangeCoords(F: PFigList; EM: TEditMode; x,y:integer; var	F	Указатель на редактируемую фигуру
			EM	Тип редактирования

		TmpX, TmpY: integer);		(перемещение/изменение вершины/движение стороны и тд)
			x	Старая координата X
			y	Старая координата Y
			TmpX	Новая координата X
			TmpY	Новая координата Y
magnetize WithFigures	«Примагничивание» линий к текстовым фигурам	function magnetizeWithFigures (head: PFigList; Point: PpointsList):Boolean;	head	Ссылка на голову списка фигур
			Point	Точка, к которой примагничивается фигура
MagnetizeL ines	«Примагничивание» линий к фигурам (текстовым и нилиям) (если расположены рядом, они соединяются)	procedure MagnetizeLines(head: PfigList);	head	Ссылка на голову списка фигур
undoChang es	Отмена изменений	procedure undoChanges(UndoRe c: TUndoStackInfo; Canvas: TCanvas);	UndoRec	Запись из стека
			Canvas	Объект Canvas

4.6.2. Описание модуля Model.Lines

Модуль Model.Lines является дополнительным юнитом модели. В модуле присутствуют подпрограммы взаимодействия с линиями. Описание основных подпрограмм, описанных в модуле Model.Lines приведено в таблице 4.5.

Таблица 4.5 – основные подпрограммы модуля Model.Lines

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
changeLine CoordsFromStr	Изменение координат определенной линии путем парсинга строки с координатами	procedure changeLineCoordsFromStr (head: PPointsList; st:string);	head	Указатель на голову списка точек линии.
			st	Строка с координатами в

				заранее заданном формате: "X1/Y1""X2/Y2" ...
copyPointList	Создать копию списка точек. (Для того, чтобы скопировать линию)	function copyPointList(copiedHead: PPointsList):PPointsList;	copiedHead	Указатель на голову списка точек, который надо скопировать
addLine	Добавление линии в список фигур, создание списка точек линии, добавление в него первой точки	function addLine(head: PFigList; x,y: integer):PFigList;	head	Указатель на список фигур
			x	Координата X первой точки линии
			y	Координата Y первой точки линии
removeTrashLines	Удаление «мусорных» линий (линий, состоящих из одной точки, причем рисование линии закончено)	procedure removeTrashLines(head: PFigList; curr: PFigList);	head	Указатель на голову списка фигур
			curr	Указатель на текущую линию.
addNewPoint	Добавление новой точки в список точек линии	function addNewPoint(var head: PPointsList; x,y:integer):PPointsList;	head	Указатель на голову списка точек линии
			x	Координата X добавляемой точки
			y	Координата Y добавляемой точки
checkLineCoords	Процедура превращает линии под углом в вертикальные или горизонтальные в зависимости от угла наклона.	procedure checkLineCoords(head: PPointsList);	head	Указатель на список точек линии
MoveLine	Перемещение всех точек, связанных с точкой, перемещенной из координат (onlp.x, oldp.y) в (newp.x, newp.y)	procedure MoveLine(head: PPointsList; oldp, newp: TPointsInfo);	head	Указатель на голову списка точек линии
			oldp	Запись со старыми координатами перемещенной точки

			newp	Запись с новыми координатами перемещенной точки
moveALLinePoint	Перемещение всех точек линии, изменяя координаты x на dx и координаты y на dy	procedure moveALLinePoint(head: PPointsList; dx, dy: integer);	head	Указатель на голову списка точек линии
			dx	Смещение по x
			dy	Смещение по y
searchNearLine	Функция ищет линию вблизи точки (в области, заданной в константе) и возвращает найденный координаты через var-параметры. Результат функции – ближайшая точка прямой. (nil если линий не найдено).	function searchNearLine(head: PFigList; var x,y: integer):PPointsList;	head	Голова
			x	Координата X, около которой ищутся линии
			y	Координата Y, около которой ищутся линии

4.6.3. Описание модуля Model.UndoStack

Модуль Model.Lines является дополнительным юнитом модели. В модуле присутствуют подпрограммы взаимодействия с линиями.

Основная идея взаимодействия со стеком – каждое изменение пользователя добавляет запись в стек. Если пользователь хочет отменить изменение – извлекается запись из вершины стека и обрабатывается в модели.

Обработке поддаются следующие изменения:

- Удаление фигуры (В стеке содержится ссылка удаленной страницы и ссылка на фигуру, расположенную перед удаленной)
- Добавление точки линии (В стеке содержится ссылка на линию и ссылка на добавленную точку)
- Добавление фигуры (В стеке содержится ссылка на добавленную фигуру)
- Перемещение фигуры/изменение размеров фигуры (кроме линий) (В стеке содержится ссылка на фигуру, которую переместили и предыдущее значение записи информативной части фигуры (предыдущие координаты))
- Перемещение линии / перемещение точки линии (В стеке содержится ссылка на фигуру линии, которую переместили и копию старых координат всех точек, сохраненных в текстовом формате)

- Изменение текста фигуры (Кроме линий) (В стеке содержится ссылка на фигуру и предыдущий текст)
- Изменение размеров полотна (В стеке содержатся предыдущие значения ширины и высоты полотна)

Описание основных подпрограмм, описанных в модуле Model.Lines приведено в таблице 4.6

Таблица 4.6 – основные подпрограммы модуля Model.UndoStack

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
CreateStack	Процедура создает стек изменений и помещает в созданный элемент тип «Запрет на изменение»	procedure CreateStack(var adr: PUndoStack);	adr	Переменная, в которую необходимо поместить адрес вершины стека после создания
UndoStackPush	Процедура создает новый элемент в стеке изменений (в вершине) и перемешает указатель вершины стека на адрес созданного элемента	procedure UndoStackPush(var Vertex: PUndoStack; info: TUndoStackInfo);	Vertex	Указатель на вершину стека
			info	Информацию, которую надо внести в новый элемент стека
undoStackPop	Извлечение одной записи из вершины стека изменений и перемещение вершины стека на предыдущий элемент в стеке. Функция возвращает false и не выполняет действий, если стек пуст	function undoStackPop(var Vertex: PUndoStack; var rec: TUndoStackInfo):boolean;	Vertex	Указатель на вершину стека
			rec	Переменная, в которую возвращается извлеченная запись
isStackEmpty	Функция возвращает true, если стек изменений пуст	function isStackEmpty(Vertex: PUndoStack): Boolean;	Vertex	Указатель на вершину стека
UndoStackClear	Очистка стека изменений	procedure UndoStackClear(var vertex: PUndoStack);	vertex	Указатель на вершину стека

4.7. Описание модулей «Представления»

4.7.1. Описание модуля View.Canvas

Модуль Model.Lines является основным юнитом представления фигур на полотне. Описание основных подпрограмм, описанных в модуле Model.Lines приведено в таблице 4.7

Таблица 4.7 – основные подпрограммы модуля View.Canvas

Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
ScaleMoveTo	MoveTo с применением масштаба	procedure ScaleMoveTo(canvas:T Canvas; x,y: integer);	canvas	Canvas
			x	X
			y	Y
ScaleLineTo	LineTo с применением масштаба	procedure ScaleLineTo(canvas:TC anvas; x,y: integer);	canvas	Canvas
			x	X
			y	Y
drawArrowVertical	Отрисовка вертикальной стрелки	procedure drawArrowVertical(Ca nvas:TCanvas; x,y : integer; coef: ShortInt);	Canvas	Canvas
			x	Координата X конца стрелки
			y	Координата Y конца стрелки
			coef	Коэффициент, отвечающий за направление
drawArrow	Отрисовка горизонтальной стрелки	procedure drawArrow(Canvas:TC anvas; x,y : integer; coef: ShortInt);	Canvas	Canvas
			x	Координата X конца стрелки
			y	Координата Y конца стрелки
			coef	Коэффициент, отвечающий за направление
drawOutBoundLine	Отрисовка диагонального отрезка перед началом горизонтальной линии	procedure drawOutBoundLine(can vas: TCanvas; FirstP: TPointsInfo; tmp:PPointsList);	canvas	Canvas
			FirstP	Первая точка
			tmp	Текущая точка

drawVertexRect	Отрисовка вершин фигуры	procedure drawVertexRect(canvas: TCanvas; point: TPointsInfo; color: TColor = clBlack);	canvas	Canvas
			point	Точка с координатами
			color	Цвет
drawIncomingLine	Отрисовка диагонального отрезка после конца горизонтальной линии	procedure drawIncomingLine(canvas: TCanvas; point: TPointsInfo; coef: ShortInt);	canvas	Canvas
			point	Точка с координатами
			coef	Коэффициент, отвечающий за направление
drawArrowAtEnd	Отрисовка стрелки на конце линии	procedure drawArrowAtEnd(canvas: TCanvas; point, PrevPoint: TPointsInfo);	canvas	Canvas
			point	Координаты с текущей точки
			PrevPoint	Координаты предыдущей точки
drawLines	Отрисовка линии	procedure drawLines(Canvas: TCanvas; head: PPointsList; isVertex: boolean; scale: Real);	Canvas	Canvas
			head	Ссылка на голову списка фигур
			isVertex	Отрисовывать ли вершины фигур
			scale	Масштаб
drawFigure	Отрисовка фигур	procedure drawFigure(Canvas: TCanvas; head: PFigList; scale: real; isVertex: boolean = true);	Canvas	Canvas
			head	Ссылка на голову списка точек линии
			scale	Масштаб
			isVertex	Отрисовывать ли вершины фигур

4.7.2. Описание модуля View.SVG

Модуль Model.SVG является основным юнитом представления фигур в SVG. Описание основных подпрограмм, описанных в модуле Model.SVG приведено в таблице 4.8.

Таблица 4.7 – основные подпрограммы модуля View.SVG

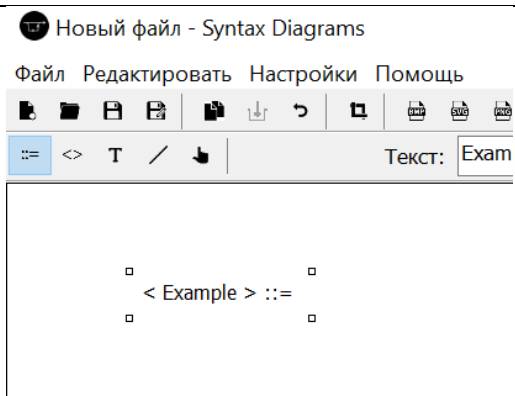
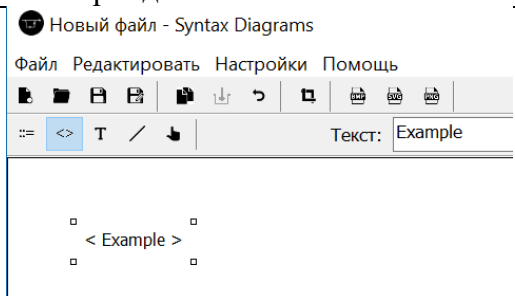
Имя подпрограммы	Описание	Заголовок подпрограммы	Имя параметра	Назначение параметра
getSVGOpenTag	Функция возвращает открытие тега svg	function getSVGOpenTag(h,w: integer):UTF8String;	h	Высота полотна
			w	Ширина полотна
writePatch	Функция возвращает тег path, описывающий линию из точки в точку	function writePatch(Point1, Point2: TPointsInfo; color: UTF8String = 'black'; width:Integer =Default_LineSVG_Width):UTF8String;	Point1	Координаты первой точки
			Point2	Координаты второй точки
			color	Цвет
			width	Ширина линии
writeSVGText	Функция возвращает тег text, описывающий текстовую фигуру	function writeSVGText(Figure: TFigureInfo; text: UTF8String; family:UTF8String = 'Tahoma'; size:integer = 16):UTF8String;	Figure	Описание фигуры
			text	Текст
			family	Название шрифта
			size	Размер шрифта
htmlspecialchars	Преобразование таких символов, как “<”, “>” и тд в HTML-сущности	function htmlspecialchars(s: UTF8String):UTF8String;	s	Исходная строка
exportToSVG	Процедура создает и записывает текстовый файл в формате XML с векторным представлением синтаксической диаграммы в формате SVG	procedure exportToSVG(head: PFigList; w,h: Integer; path:UTF8String; title: UTF8String; desc: UTF8String);	head	Указатель на голову списка фигур
			w	Ширина полотна
			h	Высота полотна
			path	Путь для сохранения файла
			title	Название файла
			Desc	Описание файла

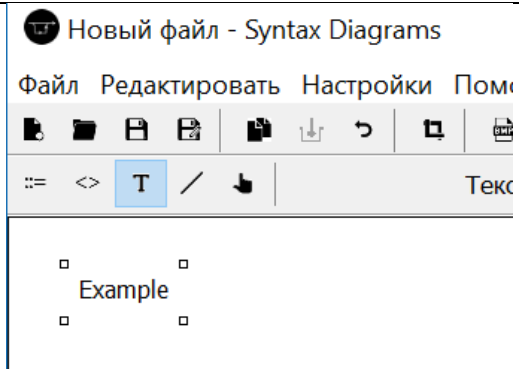
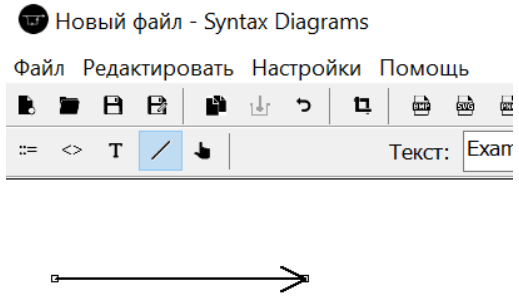
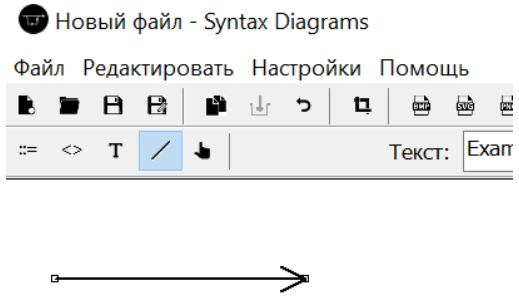
5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

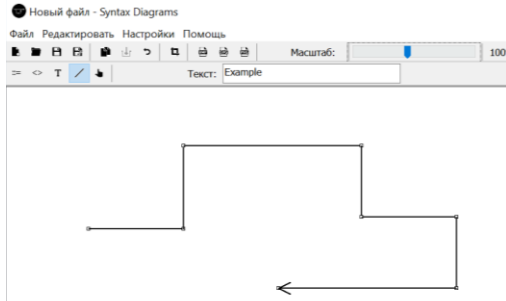
Проведено тестирование программного средства. Тестирование программного средства производилась на персональном компьютере с установленной операционной системой Windows 10. Дополнительно работоспособность была проверена на виртуальной машине с установленной Windows XP.

5.1. Тестирование функционала добавления фигур.

Таблица 5.1 - тестирование функционала добавления фигур

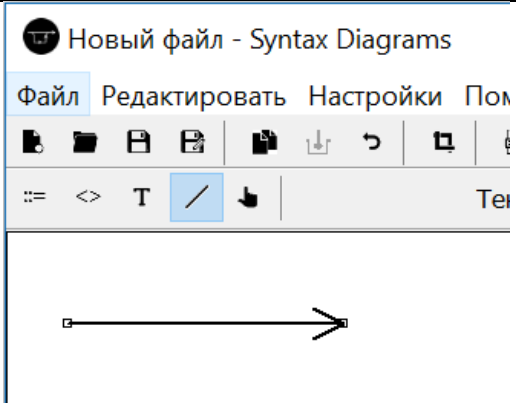
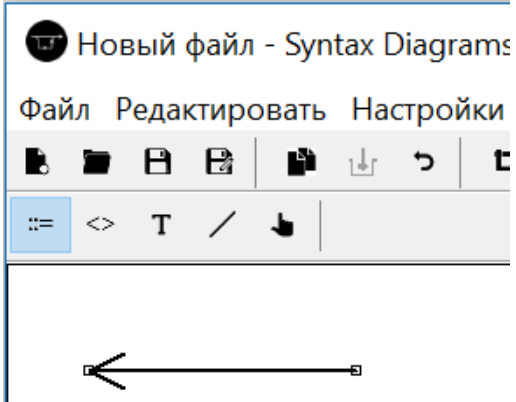
Но ме р тес та	Тестируе мая функцион альность	Последователь ность действий	Ожидаемый результат	Полученный результат
0	Добавлени е фигуры «Заголовок СД»	1.Выбор в качестве текущей фигуры «Заголовок СД» 2.Кликнуть по полотну	Фигура добавится, текст заклучится в «< >» и в конце добавится «::=»	 <p>Тест пройден!</p>
1	Добавлени е фигуры «Метапере менная»	1.Выбор в качестве текущей фигуры «Метапеременна я» 2.Кликнуть по полотну	Фигура добавится, текст заклучится в «< >»	 <p>Тест пройден!</p>

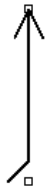
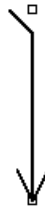
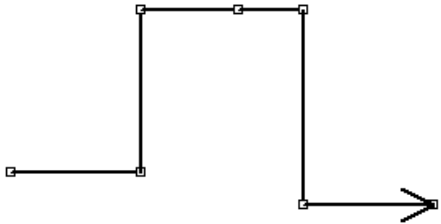
2	Добавление фигуры «Метаконстанта»	1.Выбор в качестве текущей фигуры «Метаконстанта» 2.Кликнуть по полотну	Фигура добавится, текст выведется «как есть»	 <p>Тест пройден!</p>
3	Добавление линии	1. Выбор в качестве текущей фигуры «Линия» 2. Кликнуть один раз по полотну (Левой кнопкой мыши). 3. Кликнуть второй раз по полотну на одном уровне по оси ОХ с прошлой точкой (Левой кнопкой мыши). 4. Нажать ПКМ.	Добавится линия из двух точек.	 <p>Тест пройден!</p>
4	Добавление линии (Точки не на одном уровне)	1. Выбор в качестве текущей фигуры «Линия» 2. Кликнуть один раз по полотну (Левой кнопкой мыши). 3. Кликнуть второй раз по точке, не находящийся на одном уровне ни по оси ОХ, ни по оси ОУ (Левой кнопкой мыши). 4. Нажать ПКМ.	Добавится линия, параллельная одной из осей (Спроецируется на ближайшую ось)	 <p>Тест пройден!</p>

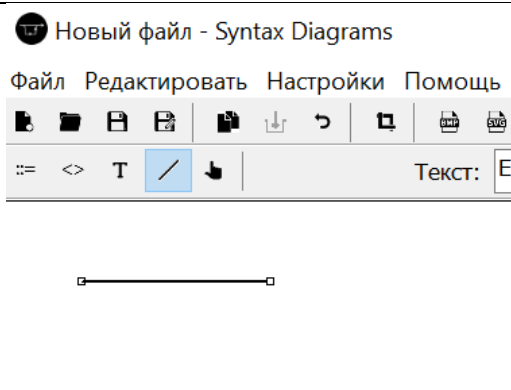
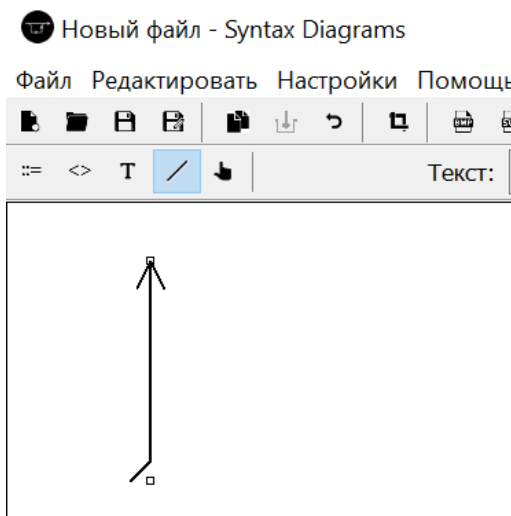
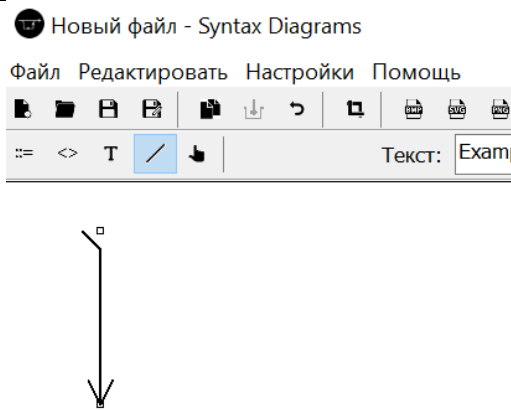
5	Добавление линии с несколькими точками	<p>1. Выбор в качестве текущей фигуры «Линия»</p> <p>2. Кликнуть один раз по полотну (Левой кнопкой мыши).</p> <p>3. Кликнуть еще несколько раз по полотну в произвольных точках. (Левой кнопкой мыши)</p> <p>4. Нажать ПКМ.</p>	Добавится линия	 <p>Тест пройден!</p>
6	Добавление линии, состоящей из одной точки	<p>1. Выбор в качестве текущей фигуры «Линия»</p> <p>2. Кликнуть один раз по полотну (Левой кнопкой мыши).</p> <p>3. Нажать ПКМ.</p>	Линия не добавится (Удалится «сборщиком мусора»)	Тест пройден
7	Добавление линии, у которой будет много точек с одинаковыми координатами	<p>1. Выбор в качестве текущей фигуры «Линия»</p> <p>2. Кликнуть один раз по полотну (Левой кнопкой мыши).</p> <p>3. Кликнуть несколько раз по этой же точке (Левой кнопкой мыши)</p> <p>4. Нажать ПКМ.</p>	Линия не добавится (Удалится «сборщиком мусора»)	Тест пройден

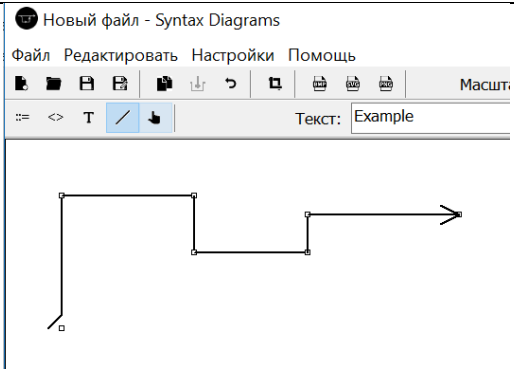
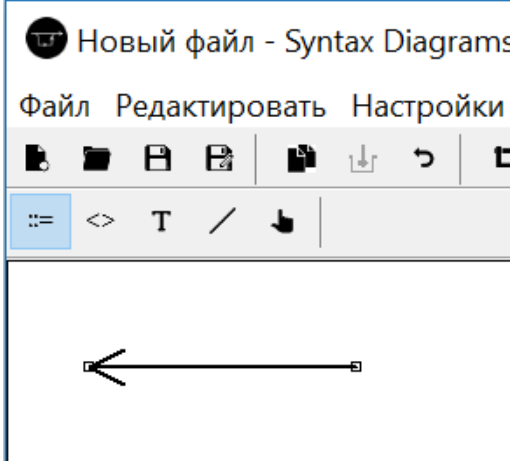
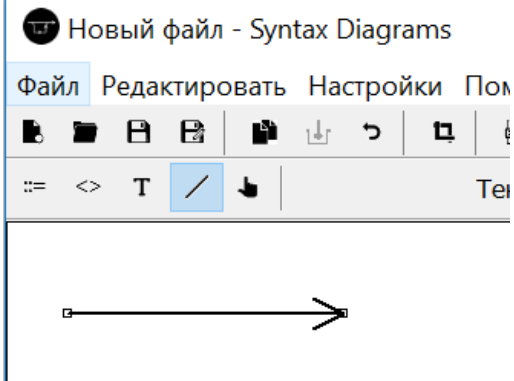
5.2. Тестирование отображения линий при разном взаимном расположении фигур.









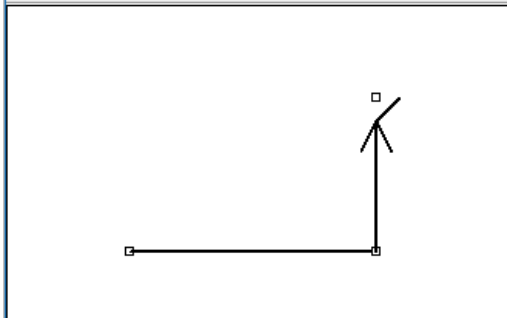








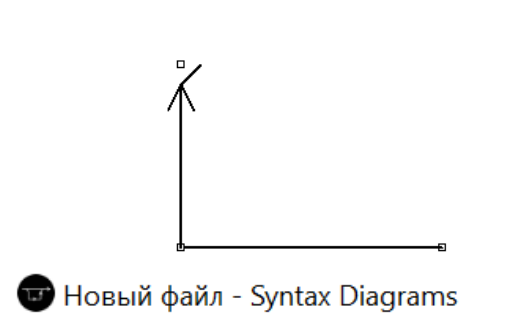








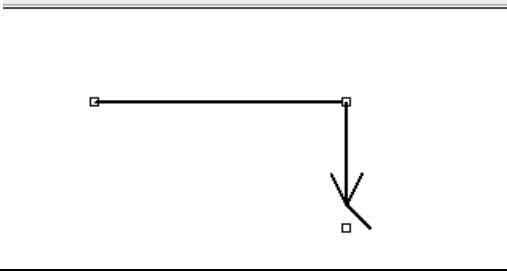
Таблица 5.2 - тестирование функционала отображения линий

Но ме р тес та	Тестиру емая функци онально сть	Последовательность действий	Ожидаемы й результат	Полученный результат
0	Отображ ение стрелки на конце линии.	1. Добавить первую точку линии 2. Добавить вторую точку линии справа от первой 3. Закончить рисование линии	Стрелка отобразитс я	 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки Пом</p> <p>Стрелка отобразится</p> <p>Тест пройден</p>
1		1. Добавить первую точку линии 2. Добавить вторую точку линии слева от первой 3. Закончить рисование линии	Стрелка отобразитс я	 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки</p> <p>Стрелка отобразится</p> <p>Тест пройден</p>

2		<ol style="list-style-type: none"> 1. Добавить первую точку линии 2. Добавить вторую точку линии сверху от первой 3. Закончить рисование линии 	Стрелка отобразится	<div> <div> <div>Новый файл - Syntax Diagrams</div> <div> <div>Файл</div> <div>Редактировать</div> <div>Настройки</div> <div>Помощь</div> </div> <div> <div>📁</div> <div>📄</div> <div>💾</div> <div>🔍</div> <div>📄</div> <div>⬇️</div> <div>↶</div> <div>📄</div> </div> <div> <div>::=</div> <div><></div> <div>T</div> <div>/</div> <div>⬇️</div> </div> <div>Текст:</div> </div>  <div>Тест пройден</div> </div>
3		<ol style="list-style-type: none"> 1. Добавить первую точку линии 2. Добавить вторую точку линии снизу от первой 3. Закончить рисование линии 	Стрелка отобразится	<div> <div> <div>Новый файл - Syntax Diagrams</div> <div> <div>Файл</div> <div>Редактировать</div> <div>Настройки</div> <div>Помощь</div> </div> <div> <div>📁</div> <div>📄</div> <div>💾</div> <div>🔍</div> <div>📄</div> <div>⬇️</div> <div>↶</div> <div>📄</div> </div> <div> <div>::=</div> <div><></div> <div>T</div> <div>/</div> <div>⬇️</div> </div> <div>Текст:</div> </div>  <div>Тест пройден</div> </div>
4		<ol style="list-style-type: none"> 1. Добавить первую точку линии 2. Добавить еще несколько точек 3. Закончить рисование линии 	Стрелка отобразится	<div> <div> <div>Новый файл - Syntax Diagrams</div> <div> <div>Файл</div> <div>Редактировать</div> <div>Настройки</div> <div>Помощь</div> </div> <div> <div>📁</div> <div>📄</div> <div>💾</div> <div>🔍</div> <div>📄</div> <div>⬇️</div> <div>↶</div> <div>📄</div> </div> <div> <div>::=</div> <div><></div> <div>T</div> <div>/</div> <div>⬇️</div> </div> <div>Текст:</div> </div>  <div>Тест пройден</div> </div>

5		<ol style="list-style-type: none"> 1. Добавить первую точку линии 2. Добавить еще несколько точек 3. Добавить последнюю точку по тем же координатам, что и предыдущая. 4. Закончить рисование линии 	Стрелка не отобразится	 <p>Тест пройден</p>
6	Отображение диагонального отрезка в начале вертикальной линии	<ol style="list-style-type: none"> 1. Добавить линию из двух точек (Вторая выше первой) 2. Закончить рисование 	Диагональный отрезок появится	 <p>Тест пройден</p>
7		<ol style="list-style-type: none"> 1. Добавить линию из двух точек (Вторая ниже первой) 2. Закончить рисование 	Диагональный отрезок появится	 <p>Тест пройден</p>

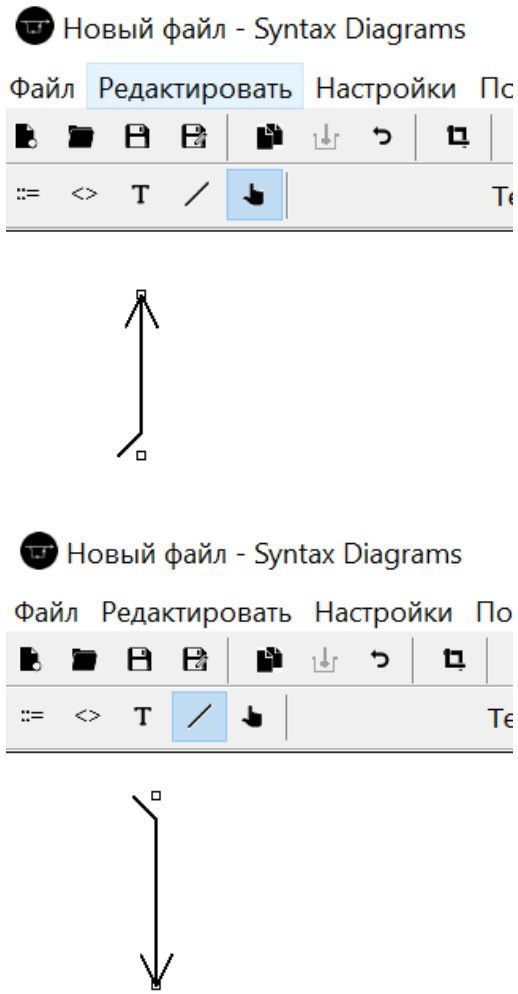
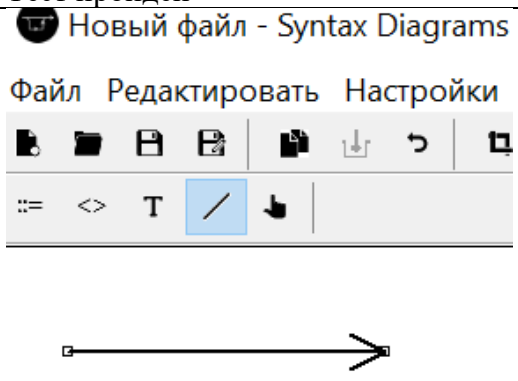
8		<p>1. Добавить линию из двух нескольких точек, причем первые две точки – вертикальный отрезок</p> <p>2. Закончить рисование</p>	<p>Диагональный отрезок появится</p>	 <p>Тест пройден</p>
9		<p>1. Добавить линию из двух точек (Вторая левее первой)</p> <p>2. Закончить рисование</p>	<p>Диагональный отрезок не появится</p>	 <p>Тест пройден</p>
10		<p>1. Добавить линию из двух точек (Вторая правее первой)</p> <p>2. Закончить рисование</p>	<p>Диагональный не появится</p>	 <p>Тест пройден</p>

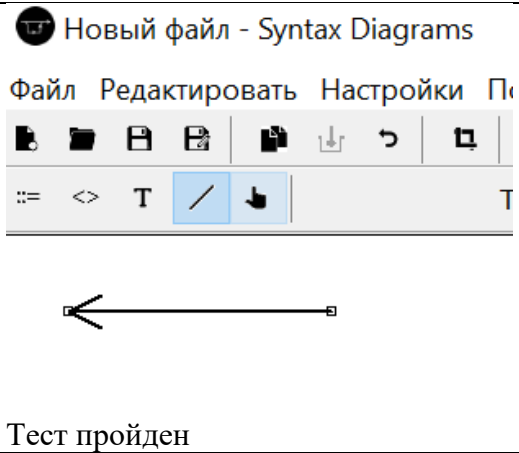
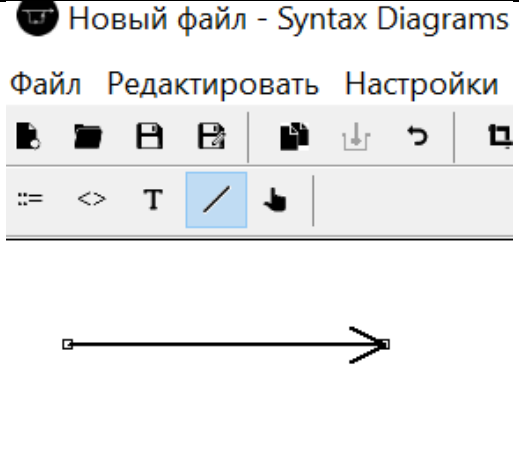
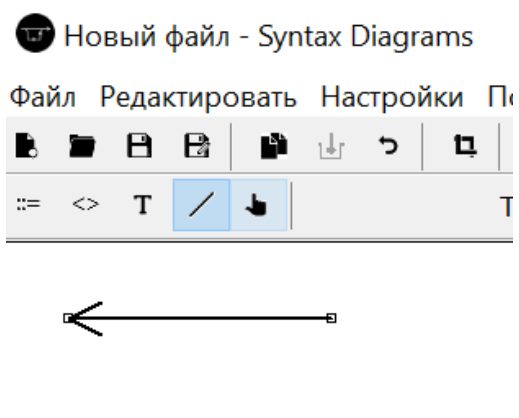
11				<div><div>Новый файл - Syntax Diagrams</div><div>Файл Редактировать Настройки Пом</div><div><div></div><div><div>≡ <> T / ↵</div><div>Текст: Те</div></div></div><div></div><div><div>Новый файл - Syntax Diagrams</div><div>Файл Редактировать Настройки Помощь</div><div><div></div><div><div>≡ <> T / ↵</div><div>Текст: Эк</div></div></div><div></div><div><div>Новый файл - Syntax Diagrams</div><div>Файл Редактировать Настройки По</div><div><div></div><div><div>≡ <> T / ↵</div><div>Текст: Те</div></div></div><div></div></div></div></div>
----	--	--	--	---

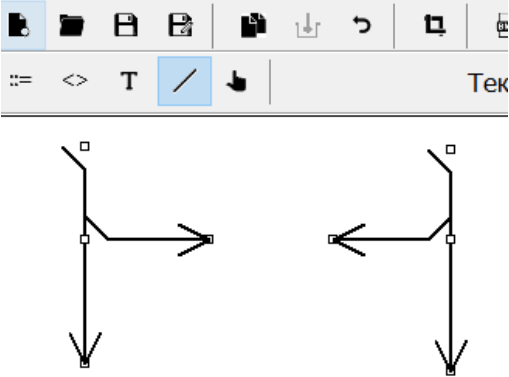
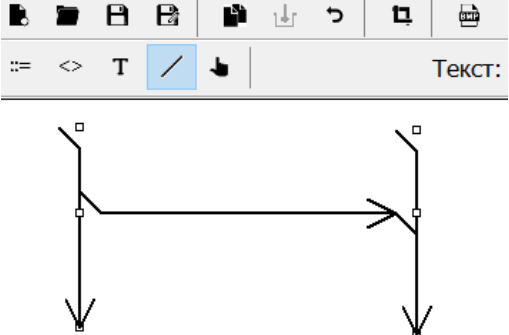
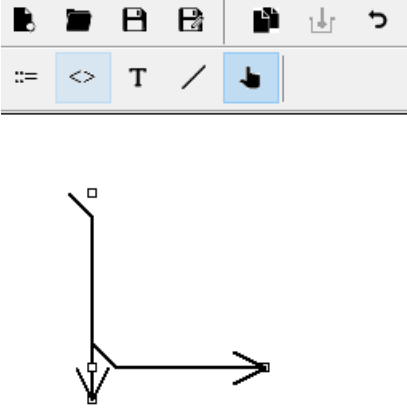
Отображение диагонального отрезка в конце вертикальной линии


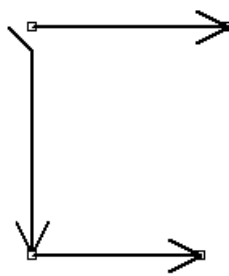

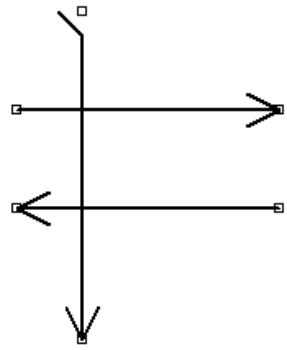
1. Добавить линию из трех точек:
Первые две – вертикальный отрезок
Последние две – горизонтальный
2. Закончить рисование

Диагональный отрезок в конце линии появится

13		<p>1. Добавить вертикальную линию из двух точек</p> <p>2. Закончить рисование</p>	<p>Диагональный отрезок в конце линии не появится</p>	 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки По</p> <p>Тест пройден</p>
14		<p>1. Добавить горизонтальную линию из двух точек</p> <p>2. Закончить рисование</p>	<p>Диагональный отрезок в конце линии не появится</p>	 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки</p>

				 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки Пк</p> <p>Тест пройден</p>
15	Отображение диагонального отрезка в начале и конце горизонтальной линии	1. Добавить горизонтальную линию из двух точек 2. Закончить рисование	Диагональный отрезок не добавится	 <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки</p>  <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки Пк</p>

16		<ol style="list-style-type: none"> 1. Добавить горизонтальную линию из двух точек. 2. Сбоку добавить вертикальную линию так, чтобы один из концов первой линии принадлежал второй. 3. Закончить рисование 	<p>Диагональ ный отрезок добавится</p>	<div data-bbox="992 159 1498 636"> <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки Пом</p>  </div> <div data-bbox="992 663 1498 1098"> <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки Помош</p>  </div> <div data-bbox="992 1125 1396 1633"> <p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настро</p>  </div>
----	--	--	--	---

17		<p>1. Добавить горизонтальную линию из двух точек.</p> <p>2. Сбоку добавить вертикальную линию так, чтобы один из концов первой линии принадлежал второй. (На самом краю второй линии)</p> <p>3. Закончить рисование</p>	Горизонтальный отрезок не отобразится	<p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки П</p>   <p>Тест пройден</p>
18		<p>1. Добавить горизонтальную линию из двух точек.</p> <p>2. Добавить вертикальную линию так, чтобы они пересекались не в первой точке</p> <p>3. Закончить рисование</p>	Горизонтальный отрезок не отобразится	<p>Новый файл - Syntax Diagrams</p> <p>Файл Редактировать Настройки П</p>   <p>Тест пройден</p>

19	Отображение стрелки по середине горизонтальной линии	<ol style="list-style-type: none"> 1. Добавить первые две точки линии. Первые две точки – вертикальный отрезок. 2. Добавить третью точку. Вторая и третья точка – горизонтальный отрезок. 3. Добавить четвертую точку. Третья и четвертая точка – вертикальная линия. 	Стрелка по середине отобразится	<div> <div> Новый файл - Syntax Diagrams Файл Редактировать Настройки </div> <div> </div> </div> <div>Тест пройден</div>
----	--	--	---------------------------------	--

5.3. Тестирование функционала редактирования текстовых фигур.

Таблица 5.3 - тестирование функционала редактирования текстовых фигур

Но ме р Т е с та	Тестируема я функциона льность	Последовательность действий	Ожидаемый результат	Полученны й результат
0	Перемещени е фигуры	<ol style="list-style-type: none"> 1. Переключить режим на «Режим редактирования». 2. Навести курсор на центр фигуры. 3. Зажать левую кнопку мыши. 4. Переместить курсор 5. Отпустить курсор 	Фигура переместится	Тест пройден
1	Изменение размеров фигуры путем перетаскива ния вершины	<ol style="list-style-type: none"> 1. Переключить режим на «Режим редактирования». 2. Навести курсор на вершину фигуры. 3. Зажать левую кнопку мыши. 4. Переместить курсор 5. Отпустить курсор 	Переместится вершина, и, следовательно, изменится размер фигуры	Тест пройден

2		1. Переключить режим на «Режим редактирования». 2. Навести курсор на вершину фигуры. 3. Зажать левую кнопку мыши 4. Переместить курсор, уменьшая фигуру до такого уровня, чтобы размеры фигуры были меньше размера текста 5. Отпустить курсор	Размер фигуры будет изменяться, причем не допустится, чтобы текст выходил за рамки	Тест пройден
3	Перемещение стороны фигуры	1. Переключить режим на «Режим редактирования». 2. Навести курсор на сторону фигуры. 3. Зажать левую кнопку мыши, уменьшая фигуру до такого уровня, чтобы размеры фигуры были меньше размера текста 4. Переместить курсор 5. Отпустить курсор	Сторона передвигается	Тест пройден
4		1. Переключить режим на «Режим редактирования». 2. Навести курсор на сторону фигуры. 3. Зажать левую кнопку мыши 4. Переместить курсор, уменьшая фигуру до такого уровня, чтобы размеры фигуры были меньше размера текста 5. Отпустить курсор	Сторона предвигается, причем не допустится, чтобы текст выходил за рамки	
5	Изменение текста внутри фигуры	1. Переключить режим на «Режим редактирования» 2. Кликнуть по текстовой фигуре. 3. Изменить содержимое текстового поля 4. Нажать клавишу «Enter»	Текст фигуры изменится	Тест пройден
6	Удаление фигуры	1. Переключить режим на «Режим редактирования» 2. Кликнуть по текстовой фигуре. 3. Нажать клавишу «Delete»	Фигура удалится	Тест пройден

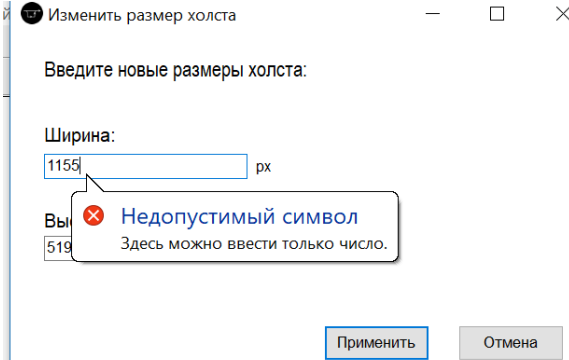
5.4. Тестирование функционала редактирования линий

Таблица 5.4 - тестирование функционала редактирования линий

Но ме р Те ста	Тестируема я функциона льность	Последовательность действий	Ожидаемый результат	Полученны й результат
0	Перемещени е всей линии	1. Переключить режим на «Режим редактирования». 2. Навести курсор на линию (не на вершину) 3. Зажать левую кнопку мыши 4. Переместить курсор 5. Отпустить курсор	Линия перемещается	Тест пройден
1	Перемещени е отдельной точки линии	1. Переключить режим на «Режим редактирования». 2. Навести курсор на линию (не на вершину) 3. Зажать левую кнопку мыши 4. Переместить курсор 5. Отпустить курсор	Переместится точка, а вместе с ней смежные точки	Тест пройден
2	Удаление линии	1. Переключить режим на «Режим редактирования» 2. Кликнуть по линии. 3. Нажать клавишу «Delete»	Линия удалится	Тест пройден
3	Копировани е и вставка фигуры	1. Нарисовать фигуру 2. Скопировать фигуру 3. Вставить фигуру	Фигура скопировалась	Тест пройден
4		1. Нарисовать фигуру 2. Скопировать фигуру. 3. Вставить фигуру. 4. Вставить фигуру. 5. Вставить фигуру. 6. Скопировать фигуру. 7. Вставить фигуру	Копирование успешно произошло, пункты 4 и 5 не дали результатов	Тест пройден
5		1. Нарисовать фигуру 2. Вставить фигуру 3. Скопировать фигуру 4. Вставить фигуру	Копирование успешно произошло, п. 2 не дал результатов	Тест пройден

5.5. Тестирование функционала изменения размера полотна

Таблица 5.5 - тестирование функционала изменения размеров полотна

Но ме р Те ста	Тестиру емая функци онально сть	Последовател ьность действий	Ожидаемый результат	Полученный результат
0	Изменен ие	1. В меню выбрать Настройка – Размер холста 2. Ввести валидные размеры	Размер полотна поменяется	Тест пройден
1	размера через форму редакти рования размера потона	1. В меню выбрать Настройка – Размер холста 2. Попробовать ввести невалидные размеры (Отрицательны е размеры/буквы /символы)	Появится предупрежден ие	 Тест пройден
2	Нажатие «Отмена после изменен ия данных»	1. В меню выбрать Настройка – Размер холста 2. Изменить размеры 3. Нажать «Отмена»	Ничего не изменится	Тест пройден
3	Изменен ие размера через редакти рование мышью	1. Выбрать в ToolBar «Изменить размер полотна» 2. Проводить курсором. 3. Нажать ЛКМ	При каждом перемещении курсора размер меняется. При нажатии ЛКМ - устанавливает ся	Тест пройден

5.6. Тестирование функционала отмены изменений

Таблица 5.6 - тестирование функционала отмены изменений

Но ме р Те ста	Тестируема я функциона льность	Последовательность действий	Ожидаемый результат	Полученны й результат
0	Отмена добавления фигуры	1. Добавить фигуру 2. Отменить изменение	Добавленная фигура удалится	Тест пройден
1	Отмена удаления фигуры	1. Добавить фигуру 2. Удалить фигуру 3. Отменить изменение	Фигура вновь появится	Тест пройден
2	Отмена изменения размеров текстовой фигуры	1. Добавить текстовую фигуру 2. Изменить размеры текстовой фигуры 3. Отменить изменение	Установятся прежние размеры	Тест пройден
3	Отмена перемещения фигуры	1. Добавить фигуру 2. Переместить фигуру 3. Отменить изменения	Фигура вернется в прежнее положение	Тест пройден.
4	Отмена добавления точки линии.	1. Добавить первую точку линии. 2. Добавить вторую точку линии. 3. Отменить изменения	Точка удалится	Тест пройден
5		1. Добавить первую точку линии. 2. Отменить изменения	Точка удалится вместе с фигурой	Тест пройден
6		1. Добавить первую точку линии. 2. Добавить вторую точку линии. 3. Отменить изменения 4. Добавить новую точку линии	Вторая точка удалится, после чего добавится новая	Тест пройден
7	Отмена изменения текста фигуры	1. Добавить фигуру 2. Изменить текст фигуры 3. Отменить изменения	Вернется исходный текст	Тест пройден
8	Совмещение	1. Добавить фигуру 2. Добавить еще одну фигуру 3. Отменить изменения 4. Изменить размеры полотна 5. Изменить текст фигуры 6. Удалить фигуру 7. Добавить линию 8. Переместить линию 9. Удалить линию 10. Отменить изменения 11. Отменить изменения	Все изменения отменяются последовательно	Тест пройден

		12. Отменить изменения		
9	Попытка отмены изменений в пустом стеке	1. Отменить изменения. 2. Отменить изменения. 3. Отменить изменения 4. Добавить фигуру 5. Отменить изменения	Пункты 1-3 не дадут результаты, фигура успешно добавилась, после чего, при отмене изменений, удалась	Тест пройден
10	Открытие файла после отмены изменений	1. Добавить фигуру 2. Открыть другой файл 3. Отменить изменения	Ничего не произойдет	Тест пройден

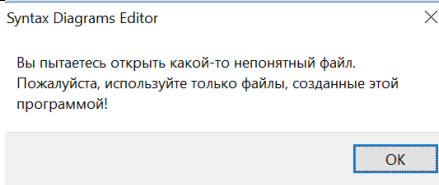
5.7. Тестирование функционала «примагничивания»

Таблица 5.7 - тестирование функционала «примагничивания»

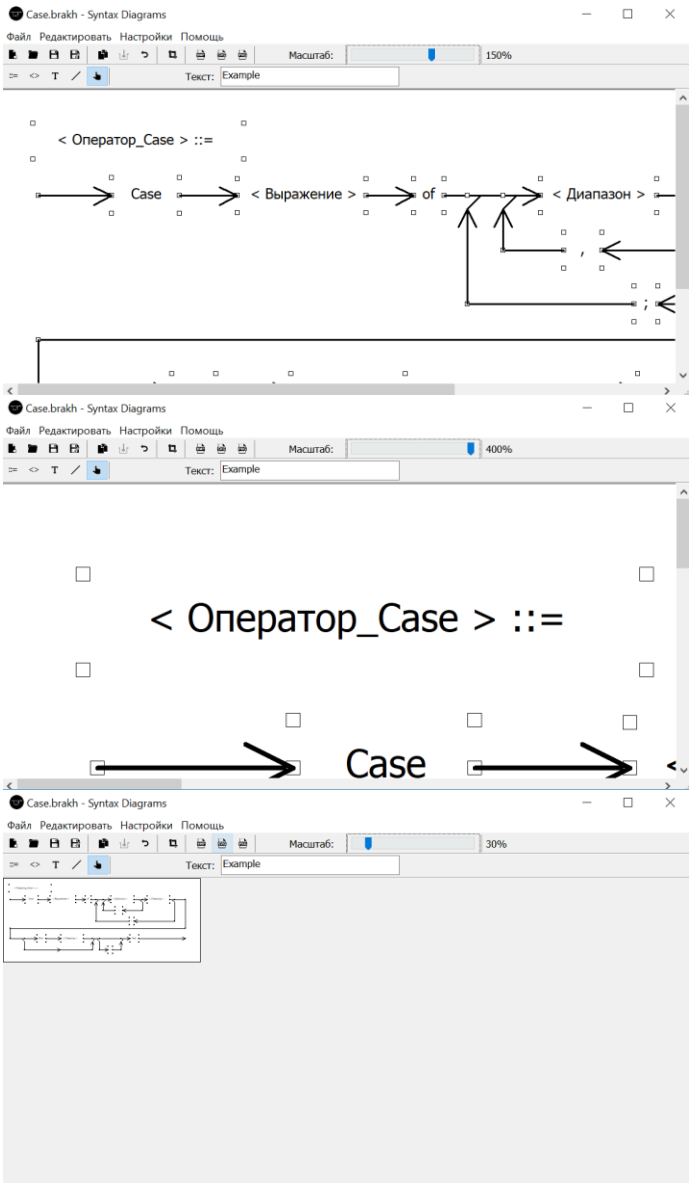
Но ме р Те ста	Тестируема я функциона льность	Последовательность действий	Ожидаемый результат	Полученны й результат
0	Линия расположена близко к другой линии	1. Нарисовать одну вертикальную линию. 2. Нарисовать вторую горизонтальную линию так, чтобы начальная или конечная точка была близко к первой линии	Линии соединятся	Тест пройден
1		1. Нарисовать одну вертикальную линию. 2. Нарисовать вторую горизонтальную линию так, чтобы начальная или конечная точка была близко к первой линии 3. Переместить первую линию на небольшое расстояние	Вместе с первой линией переместится вторая	Тест пройден
2	Линия расположена близко к фигуре	1. Нарисовать фигуру. 2. Нарисовать линии, близкую к боковой грани фигуры	Линия примагнитится к фигуре	Тест пройден
3		1. Нарисовать фигуру. 2. Нарисовать линии, близкую к боковой грани фигуры 3. Переместить фигуру на небольшое расстояние	Вместе с фигурой переместится линия	Тест пройден

5.8. Тестирование прочего функционала программного средства

Таблица 5.8 – тестирование прочего функционала программного средства

Но ме р Те ста	Тестиру емая функци онально сть	Последовательность действий	Ожидаемый результат	Полученный результат
0	Сохране ние исходно го файла и его открыти е	1. Нарисовать текстовую фигуру 2. Сохранить файл 3. Заккрыть программу 4. Открыть файл	Сохранение и открытие прошли успешно	Тест пройден
1		1. Нарисовать линию 2. Сохранить файл 3. Заккрыть программу 4. Открыть файл	Сохранение и открытие прошли успешно	Тест пройден
2		1. Нарисовать линию 2. Нарисовать текстовую фигуру 3. Сохранить файл 4. Заккрыть программу 5. Открыть файл	Сохранение и открытие прошли успешно	Тест пройден
3		1. Сохранить файл (На канвасе ничего нет) 2. Заккрыть программу 3. Открыть файл	Сохранение и открытие прошли успешно	Тест пройден
4		1. Открыть файл, созданный не программой	Сообщение об ошибке	 Тест пройден
5	Экспорт в PNG	1. Нарисовать что-то на полотне 2. Выбрать «Экспорт в PNG» 3. Выбрать путь	Файл сохранился	Тест пройден
6		1. Выбрать «Экспорт в PNG» 2. Выбрать путь	Файл сохранился	Тест пройден
7	Экспорт в BMP	1. Нарисовать что-то на полотне 2. Выбрать «Экспорт в BMP»	Файл сохранился	Тест пройден

		3. Выбрать путь		
8		1. Выбрать «Экспорт в BMP» 2. Выбрать путь	Файл сохранился	Тест пройден
9	Экспорт в SVG	1. Нарисовать что-то на полотне 2. Выбрать «Экспорт в SVG» 3. Выбрать путь	Файл сохранился	Тест пройден
10		1. Выбрать «Экспорт в SVG» 2. Выбрать путь	Файл сохранился	Тест пройден
11	Создание нового файла	1. Нарисовать что-то на полотне 2. Выбрать «Новый файл»	Покажется уведомление, что несохраненные изменения удалятся, если выбрано «ОК» - создастся новый файл	Тест пройден
12		1. Нарисовать что-то на полотне 2. Заккрыть форму	Покажется предложение сохранить изменения	Тест пройден
13		1. Заккрыть форму	Форма закроется	Тест пройден
14		1. Нарисовать что-то на полотне 2. Заккрыть форму 3. Нажать «Да» 4. Сохранить файл	Файл сохранится, форма закроется	Тест пройден
15	Заккрытие формы	1. Нарисовать что-то на полотне 2. Заккрыть форму 3. Нажать «Да» 4. Отменить сохранение	Форма не закроется	Тест пройден
16		1. Нарисовать что-то на полотне 2. Заккрыть форму 3. Нажать «Отмена»	Форма не закроется	Тест пройден
17		1. Нарисовать что-то на полотне 2. Заккрыть форму 3. Нажать «Нет»	Форма закроется, изменения не сохранятся	Тест пройден

18	Изменение масштаба	<p>1. Нарисовать что-то на полотне.</p> <p>2. Изменить масштаб путем изменения «ползунка» масштаба</p>	Масштаб изменится	 <p>Тест пройден</p>
----	--------------------	--	-------------------	---

5.9. Вывод из прохождения тестирования

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Для того, чтобы начать использование программы, необходимо запустить файл SyntaxDiag.exe, либо открыть файл с расширением «.brakh» (Расширение заранее должны быть ассоциировано с программой). После открытие программы появится окно, показанное на рисунке 6.1.

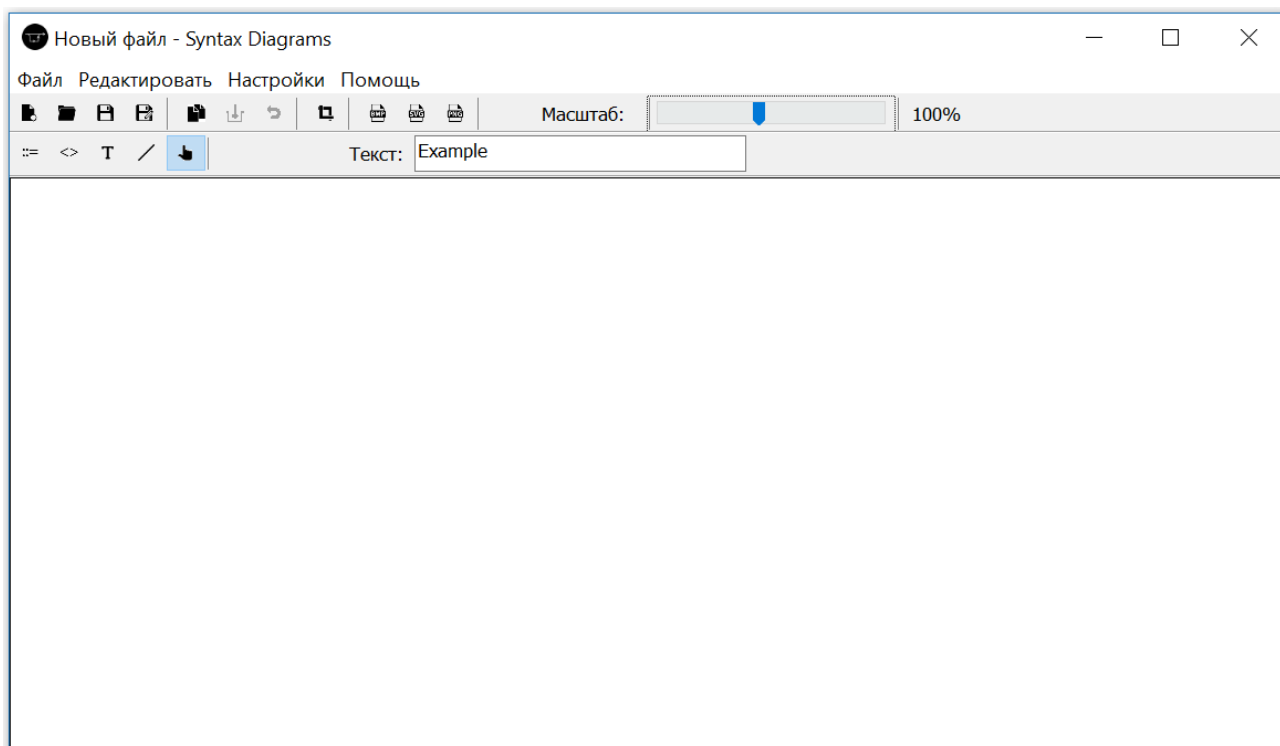


Рисунок 6.1 – окно редактора синтаксических диаграмм.

По умолчанию при открытии активен режим «Редактирования», в котором запрещено рисование. Чтобы его переключить, необходимо выбрать подходящий режим, кликнув на соответствующую иконку в ToolBar (рисунок 6.2).



Рисунок 6.2. – иконки изменения режима.

После выбора режима, можно приступить к рисованию (Если не выбран режим редактирования). Если выбран режим «линия», то каждое нажатие ЛКМ по полотну добавляет новую точку фигуры. Нажатие ПКМ – прекращает рисование текущей фигуры. Иначе, если выбран один из режимов рисования текстовых фигур – каждое нажатие мыши добавляет новую фигуру.

Чтобы изменить текст, который будет отображаться внутри фигуры, необходимо отредактировать содержимое текстового поля, приведенного на рисунке 6.3.



Рисунок 6.3 – текстовое поле, отвечающее за содержимое текстовой фигуры

Пример рисования приведен на рисунке 6.4.

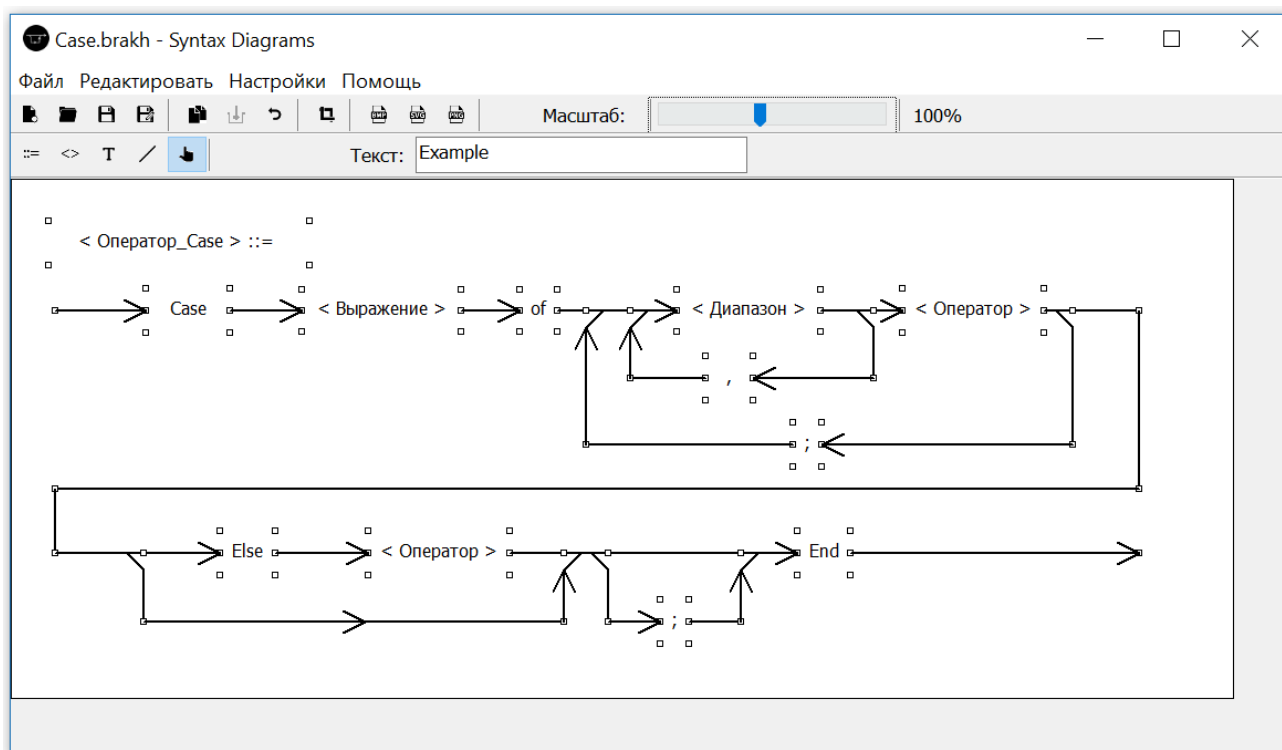


Рисунок 6.4 – пример рисования на полотне

Если выбран режим «Редактирования» (Иконка курсора), то при наведении на фигуру меняется курсор, после чего обычным зажатием и перемещением курсора можно изменять фигуру (рисунок 6.5).

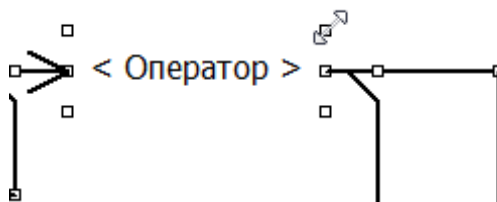


Рисунок 6.5 – изменение курсора при наведении на вершину

Для изменения текста уже добавленной фигуры, необходимо выбрать режим «Редактирование», кликнуть на фигуру, в которой нужно поменять текст, изменить содержимое текстового поля (Рисунок 6.3) и нажать Enter.

Для изменения масштаба изображение, необходимо передвинуть соответствующий «ползунок» (рисунок 6.6) в нужную сторону. Пример изменения масштаба приведен на рисунке 6.7.



Рисунок 6.6 – ползунок изменения масштаба

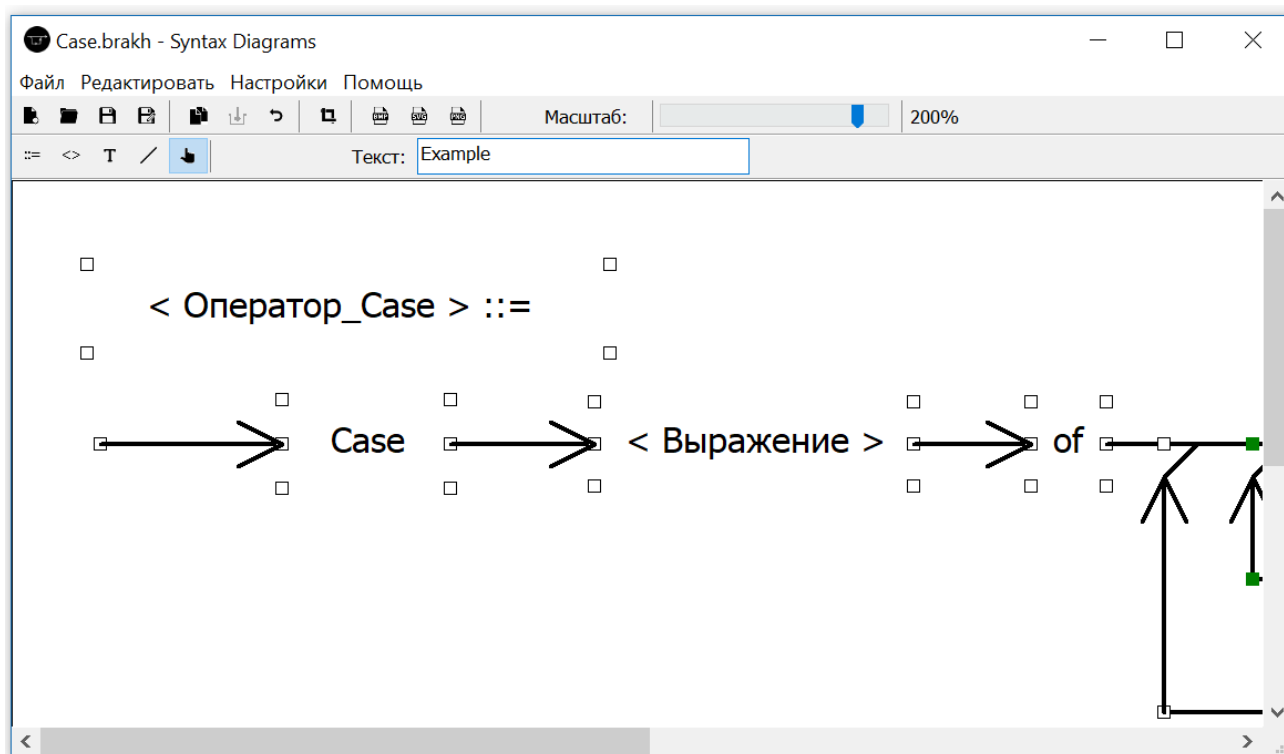


Рисунок 6.7 – пример изменения масштаба.

Также в программе присутствует удобное меню (рисунок 6.8) и toolbar (рисунок 6.9), где можно найти такие функции, как

- Открытие файла
- Сохранение исходного файла
- Экспорт в растровые форматы
- Экспорт в векторные форматы
- Создание новой синтаксической диаграммы

- Копирование фигур
- Вставка фигур
- Отмена изменений
- Изменение размера холста
- Просмотр справки
- Изменить язык (Скриншот программы после смены языка представлен на рисунке 6.10)

Файл Редактировать Настройки Помощь

Рисунок 6.8 – главное меню программы



Рисунок 6.9 – главный ToolBar программы

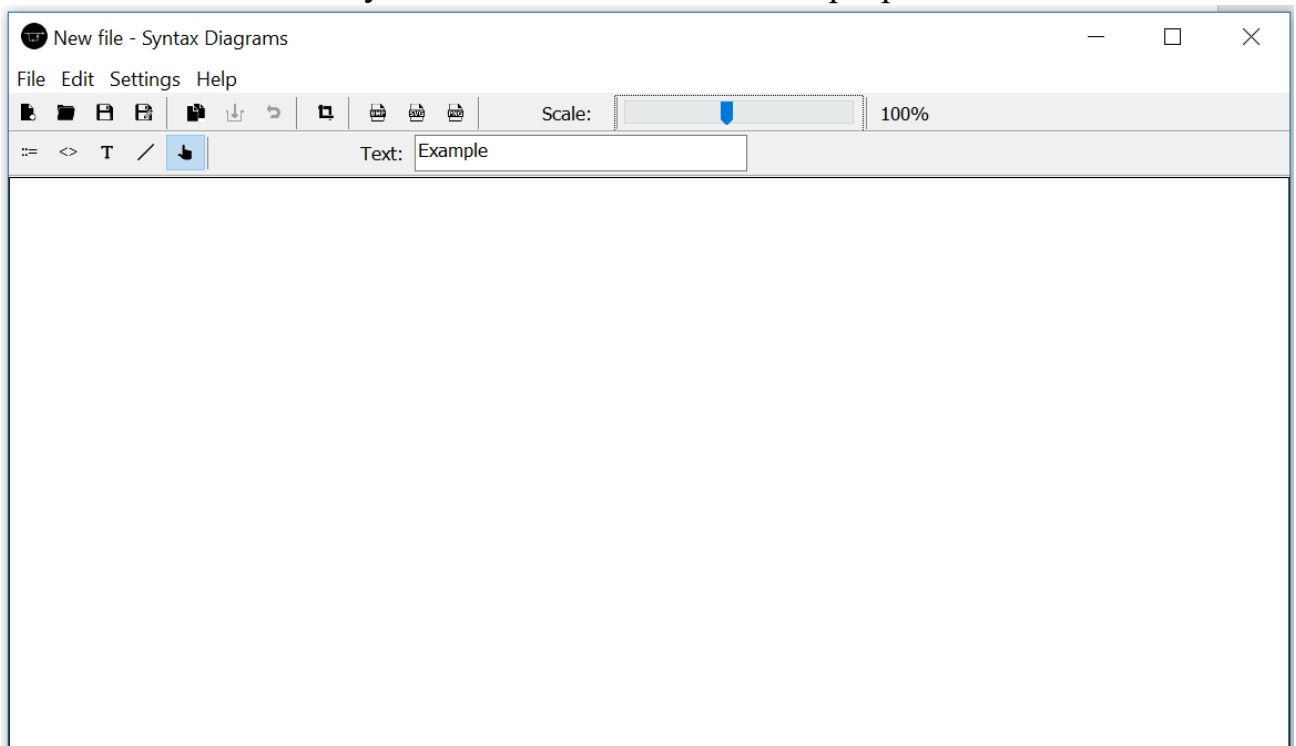


Рисунок 6.10 – скриншот главного окна после смены языка

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта было разработано приложение, имеющий достаточный функционал для построение синтаксических диаграмм. Программа содержит не только основные функции, но и ряд дополнительных, способствующих расширить функционал программного средства и улучшить взаимодействие с пользователем. Благодаря мультязычному интерфейсу, приложение может использоваться в разных странах.

При тестировании и отладке не было выявлено случаев некорректной работы программы, нестабильной работы, появления сторонних ошибок и т.д.

Простой и удобный пользовательский интерфейс позволяет пользователю легко привыкнуть к приложению и начать рисовать диаграммы. Наличие справки по использованию в самом приложении делают этот процесс еще качественнее.

Написанный код легко модифицируется для добавления новых фигур, а изменения вида может быть модифицирован путем изменения констант. В дальнейшем возможны улучшения и доработки, вносящие новый функционал в программу.

В процессе разработки, я изучил принципы работы с графикой в языке программирования Delphi, принцип работы с векторным форматом SVG и научился правильно его заполнять, принципы правильной организации пользовательского интерфейса. Полученные знания и опыт позволят легко применить их в новых проектах.

Разработанное программное средство можно применять как в обучающих целях (Например: для изучения студентами нового языка программирования), так и разработчиками языков программирования и компиляторов для того, чтобы нагляднее описать синтаксис языка в справочных материалах.

СПИСОК ЛИТЕРАТУРЫ

- [1] Глухова, Л.А. Основы алгоритмизации и программирования: учебное пособие. В 2 Ч. / Л.А. Глухова. – БГУИР, 2006 – Ч. 1. – 195 с.
- [2] Документация по SVG – MSDN [Электронный ресурс] Режим доступа: [https://msdn.microsoft.com/library/bg124132\(v=vs.85\).aspx](https://msdn.microsoft.com/library/bg124132(v=vs.85).aspx) — Дата доступа: 06.04.18
- [3] SVG – MDN web docs [Электронный ресурс] Режим доступа: <https://developer.mozilla.org/docs/Web/SVG#Documentation> — Дата доступа: 04.04.18
- [4] Help for RAD Studio 10.2 Tokyo [Электронный ресурс] Режим доступа: http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Main_Page— Дата доступа: 16.04.18
- [5] А.Н.Вальвачев, К.А.Сурков, Д.А.Сурков, Ю.М.Четырько. Программирование на языке Delphi. Учебное пособие[Электронный ресурс] Режим доступа: <http://www.rsdn.ru/?summary/3165.xml>. Дата доступа: 20.04.18

ПРИЛОЖЕНИЕ 1

Схема программы

ПРИЛОЖЕНИЕ 2

Исходный код программы

Модуль главной формы:

```
unit Main;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls, math,
  Vcl.Menus, Data.Types, View.Canvas, Data.InitData, Model, View.SVG, Vcl.Buttons,
  System.Actions, Vcl.ActnList, System.ImageList, Vcl.ImgList,
  Vcl.Imaging.pngimage, Vcl.ComCtrls, Vcl.ToolWin, Model.UndoStack, Model.Lines;

type
  TEditorForm = class(TForm)
    edtRectText: TEdit;
    MainMenu: TMainMenu;
    mnFile: TMenuItem;
    mniSave: TMenuItem;
    mniOpen: TMenuItem;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    mniToSVG: TMenuItem;
    pbMain: TPaintBox;
    mniExportToBMP: TMenuItem;
    mniExport: TMenuItem;
    mniSaveAs: TMenuItem;
    mnSettings: TMenuItem;
    mniHolstSize: TMenuItem;
    sbMain: TScrollBar;
    mniHtml: TMenuItem;
    mniWhatIsSD: TMenuItem;
    mniNew: TMenuItem;
    alMenu: TActionList;
    actNew: TAction;
    actOpen: TAction;
    actSave: TAction;
    actSaveAs: TAction;
    actExportBMP: TAction;
    actExportSVG: TAction;
    actCopy: TAction;
    mniEdit: TMenuItem;
    mniCopy: TMenuItem;
    actPast: TAction;
    mniPast: TMenuItem;
    ilMenu: TImageList;
    actCanvasSize: TAction;
    actAboutSB: TAction;
    tbarMenu: TToolBar;
    tbNew: TToolButton;
    tbOpen: TToolButton;
    tbSave: TToolButton;
    tbSaveAs: TToolButton;
    ToolButton5: TToolButton;
    tbCopy: TToolButton;
    tbPast: TToolButton;
    ToolButton1: TToolButton;
```

```

tbBMP: TToolButton;
tbSVG: TToolButton;
tbSelectFigType: TToolBar;
tbFigDef: TToolButton;
tbFigMV: TToolButton;
tbFigConst: TToolButton;
tbFigLine: TToolButton;
tbFigNone: TToolButton;
ilFigures: TImageList;
alSelectFigure: TActionList;
actFigNone: TAction;
actFigLine: TAction;
actFigDef: TAction;
actFigMetaVar: TAction;
actFigMetaConst: TAction;
lblEnterText: TLabel;
tbSelectScale: TTrackBar;
lblScale: TLabel;
lblScaleView: TLabel;
actUndo: TAction;
mniUndo: TMenuItem;
ToolButton2: TToolButton;
actHelp: TAction;
mniProgramHelp: TMenuItem;
actPNG: TAction;
mniPNGExport: TMenuItem;
tbPNG: TToolButton;
actResizeCanvas: TAction;
tbResizeCanvas: TToolButton;
ToolButton4: TToolButton;
actChangeMagnetize: TAction;
mniMagnetizeLine: TMenuItem;
procedure FormCreate(Sender: TObject);
procedure clearScreen;
procedure pbMainMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
procedure pbMainMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
procedure pbMainMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
procedure changeEditorText(newtext: string);
procedure FormResize(Sender: TObject);
function saveBrakhFile:boolean;
procedure saveSVGFile;
procedure pbMainPaint(Sender: TObject);
procedure saveBMPFile;
procedure savePNGFile;
procedure endDrawLine;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure mniNewClick(Sender: TObject);
procedure sbMainMouseWheelDown(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
procedure sbMainMouseWheelUp(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure actNewExecute(Sender: TObject);
procedure actOpenExecute(Sender: TObject);
procedure actSaveExecute(Sender: TObject);
procedure actSaveAsExecute(Sender: TObject);
procedure actExportBMPEXecute(Sender: TObject);

```

```

procedure actExportSVGExecute(Sender: TObject);
procedure actCopyExecute(Sender: TObject);
procedure actPasteExecute(Sender: TObject);
procedure actCanvasSizeExecute(Sender: TObject);
procedure actAboutSBExecute(Sender: TObject);
procedure actFigNoneExecute(Sender: TObject);
procedure actFigLineExecute(Sender: TObject);
procedure actFigDefExecute(Sender: TObject);
procedure actFigMetaVarExecute(Sender: TObject);
procedure actFigMetaConstExecute(Sender: TObject);
procedure tbSelectScaleChange(Sender: TObject);
procedure actUndoExecute(Sender: TObject);
procedure actHelpExecute(Sender: TObject);
procedure actPNGExecute(Sender: TObject);
procedure Action1Execute(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure actResizeCanvasExecute(Sender: TObject);
procedure sbMainMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure sbMainMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
procedure actChangeMagnetizeExecute(Sender: TObject);

private
    isChanged: Boolean;
    curpath: string;
    isMoveFigure: Boolean;
    oldDM: TDrawMode;
    USVertex: PUndoStack; // US - Undo Stack
    procedure switchChangedStatus(flag: Boolean);
    procedure changePath(path: string);
    procedure newFile;
    procedure updateCanvasSizeWithCoords(x,y: Integer);
public
    PBW, PBH: integer;
    FScale: Real;
    procedure useScale(var x,y: integer);
    procedure DiscardScale(var x,y: integer);
    procedure SD_Resize;
    function getFigureHead:PFigList;
    function openFile(mode: TFileMode):string;
    function saveFile(mode: TFileMode):string;
    procedure changeCanvasSize(w,h: Integer; flag: Boolean = true);
    procedure getCanvasSize(var w,h:Integer);
end;

var
    EditorForm: TEditorForm;
    FigHead: PFigList;
    CurrType: TType;
    CurrLineType: TLineType;
    CurrFigure, ClickFigure: PFigList;
    tempX, tempY: integer;
    DM: TDrawMode;
    EM: TEditMode;
    prevText:String;
    currPointAdr: PPointsList;
    CoppyFigure: PFigList;
    //FT: TFigureType;

```

```

implementation
{$R *.dfm}
{$R HTML.RES}
{$R+}
{$R-}

uses FCanvasSizeSettings, FHtmlView, Model.Files;

procedure TEditorForm.changeEditorText(newtext: string);
begin
    edtRectText.text := newtext
end;

procedure TEditorForm.switchChangedStatus(flag: Boolean);
begin
    isChanged := flag;
end;

// Select the scale of the image
procedure TEditorForm.tbSelectScaleChange(Sender: TObject);
begin
    case (Sender as TTrackBar).Position of
        1: FScale := 0.1;
        2: FScale := 0.3;
        3: FScale := 0.5;
        4: FScale := 0.8;
        5: FScale := 1;
        6: FScale := 1.2;
        7: FScale := 1.5;
        8: FScale := 1.7;
        9: FScale := 2;
        10: FScale := 4;
    end;
    lblScaleView.Caption := ' ' + IntToStr( Round(FScale*100) ) + '%';
    changeCanvasSize(PBW, PBH);
    Self.Invalidate;
end;

procedure TEditorForm.updateCanvasSizeWithCoords(x, y: Integer);
begin
    pbMain.Width := x;
    pbMain.Height := y;
    pbMain.Repaint;
end;

procedure TEditorForm.useScale(var x, y: integer);
begin
    x := Round(FScale*x);
    y := Round(FScale*y);
end;

procedure TEditorForm.pbMainMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
var x0, y0: Integer;
    UndoRec: TUndoStackInfo;
begin
    x0 := x;

```

```

y0 := y;
roundCoords(x,y); // round coords (Use steps)

if dm = DrawLine then
begin
  case button of
    TMouseButton.mbLeft:
      begin
        // Add new point of current line:
        UndoRec.PrevPointAdr := addNewPoint( CurrFigure^.Info.PointHead, Round(x / FScale),Round(y / FScale));
        // CHANGES STASCK PUSHING START
        UndoRec.ChangeType := chAddPoint;
        UndoRec.adr := CurrFigure;
        UndoStackPush(USVertex, UndoRec);
        isChanged := true;
        actUndo.Enabled := true;
        // CHANGES STASCK PUSHING END
      end;
      // If clicked button - right or middle then finish drawing
      TMouseButton.mbRight:
        begin
          // Remove lines with one point
          endDrawLine;
        end;
      TMouseButton.mbMiddle: endDrawLine;
    end;

end
else
  DM := Draw; // Begin Drawing

// If
if (EM = NoEdit) and (CurrType <> None) then
begin
  isChanged := true;
  // if at the moment nothing draws, start drawing
  if CurrType <> Line then
  begin
    // Add new figure to canvas
    CurrFigure := addFigure(FigHead, Round(x/FScale),Round(y/FScale), CurrType, edtRectText.Text);
    // CHANGES STACK PUSHING START
    UndoRec.ChangeType := chInsert;
    UndoRec.adr := Currfigure;
    CurrFigure.Info.y1 := y - abs(CurrFigure.Info.y1 - CurrFigure.Info.y2) div 2;
    UndoStackPush(USVertex, UndoRec);
    actUndo.Enabled := true;
    // CHANGES STACK PUSHING END
  end
  else if (DM <> DrawLine) and (Button = mbLeft) then
  begin
    // Add new lines to canvas
    CurrFigure := addLine(FigHead, Round(x/FScale),Round(y/FScale));
    // CHANGES STASCK PUSHING START
    UndoRec.ChangeType := chInsert;
    UndoRec.adr := Currfigure;
    DM := DrawLine;
    UndoStackPush(USVertex, UndoRec);
    actUndo.Enabled := true;
    // CHANGES STASCK PUSHING END
  end;
end;

```

```

end
else
begin
    tempX:= x; // Update coordinates for moving
    tempY:= y;
end;

// If the click occurred on the figure, we put the current variable in the
// appropriate variable
ClickFigure := getClickFigure(Round(x0/FScale), Round(y0/FScale), FigHead);
if (ClickFigure <> nil) and (ClickFigure^.Info.tp <> line) and (CurrFigure <> nil)
    and (CurrFigure^.Info.tp <> line) then
begin
    // We paste into the text field with the text of the figure the text of the current shape
    changeEditorText(String(ClickFigure^.Info.Txt));
end
else
begin
    changeEditorText(prevText);
end;

end;

procedure changeCursor(ScrollBox:TScrollBox; Mode: TEditMode);
begin
    case mode of
        NoEdit: ScrollBox.Cursor := crArrow;
        Move: ScrollBox.Cursor := crSizeAll;
        TSide: ScrollBox.Cursor := crSizeNS;
        BSide: ScrollBox.Cursor := crSizeNS;
        RSide: ScrollBox.Cursor := crSizeWE;
        LSide: ScrollBox.Cursor := crSizeWE;
        Vert1: ScrollBox.Cursor := crSizeNWSE;
        Vert2: ScrollBox.Cursor := crSizeNESW;
        Vert3: ScrollBox.Cursor := crSizeNESW;
        Vert4: ScrollBox.Cursor := crSizeNWSE;
        LineMove: ScrollBox.Cursor := crHandPoint;
    end;
end;

// Return canvas size
procedure TEditorForm.getCanvasSize(var w, h: Integer);
begin
    w := Self.pbMain.Width;
    h := self.pbMain.Height;
end;

// Return figure head
function TEditorForm.getFigureHead:PFigList;
begin
    Result:= FigHead;
end;

procedure TEditorForm.pbMainMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
var
    undorec: TUndoStackInfo;
begin
    if dm = ResizeCanvas then
        begin

```

```

    updateCanvasSizeWithCoords(x, y);
    exit;
end;

if clickfigure = nil then
    prevText:= edtRectText.Text;
if (CurrType <> Line) and (DM = DrawLine) then
    DM := nodraw;
if (dm = NoDraw) and (CurrType = None) then
begin

    EM := getEditMode(DM, Round(x/FScale),Round(y/FScale),FigHead, CurrType);
    changeCursor(sbMain, EM); // Меняем курсор в зависимости от положения мыши

    if ClickFigure <> nil then
    begin
        selectFigure(pbMain.Canvas, ClickFigure); // add green verts for selected figure
    end;
end;
if (DM = draw) and (currfigure <> nil) then
begin
    if not isMoveFigure then
    begin
        // START MOVING

        // CHANGES STACK PUSHING START
        isMoveFigure := true;
        undorec.adr := CurrFigure;
        if CurrFigure^.Info.tp <> Line then
        begin
            undorec.ChangeType := chFigMove;
            undorec.PrevInfo := CurrFigure^.Info;
        end
        else
        begin
            undorec.ChangeType := chPointMove;
            undorec.st := pointsToStr(CurrFigure^.Info.PointHead^.adr);
        end;
        UndoStackPush(USVertex, undorec);
        actUndo.Enabled := true;
        // CHANGES STACK PUSHING END
    end;

    switchChangedStatus(TRUE);
    ChangeCoords(CurrFigure, EM, x,y, tempX, tempY ); // Changes coords

    TempX:= X; // Update old coords
    TempY:= Y;
    pbMain.Repaint;
end;
end;

procedure TEditorForm.pbMainMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    if DM <> DrawLine then
    begin
        DM := NoDraw; // End draw
        checkFigureCoord(CurrFigure);
    end;
end;

```

```

if isMoveFigure then
begin
  if mniMagnetizeLine.Checked then
    SearchFiguresInOneLine(FigHead, CurrFigure);
  isMoveFigure := false;
end;
if mniMagnetizeLine.Checked then
  MagnetizeLines(FigHead);

Self.pbMain.Repaint;
pbMain.Repaint;
end;

procedure TEditorForm.pbMainPaint(Sender: TObject);
begin
  with (Sender as TPaintBox) do
  begin
    clearScreen;
    drawFigure(Canvas, FigHead, FScale); // Draw all figures, lines
  end;
end;

procedure TEditorForm.clearScreen;
begin
  if DM = ResizeCanvas then
  begin
    pbMain.Canvas.Pen.Width := 1;
    pbMain.Canvas.Pen.Style := psDash;
    pbMain.Canvas.Pen.Color := clBlack;
  end
  else
  begin
    pbMain.Canvas.Pen.Width := 1;
    pbMain.Canvas.Pen.Style := psSolid;
    pbMain.Canvas.Pen.Color := clBlack;
  end;
end;

pbMain.Canvas.Rectangle(0,0,pbMain.Width,pbMain.Height); // Draw white rectangle :)
end;

procedure TEditorForm.DiscardScale(var x, y: integer);
begin
  x := Round(x/FScale);
  y := Round(y/FScale);
end;

procedure TEditorForm.endDrawLine;
begin
  removeTrashLines(FigHead, CurrFigure);
  dm:=NoDraw;
  pbMain.Repaint;
end;

procedure TEditorForm.FormClose(Sender: TObject; var Action: TCloseAction);
var
  answer: integer;

```



```

begin
if isChanged then
begin
answer := MessageDlg(rsExitDlg,mtCustom,
[mbYes,mbNo,mbCancel], 0);
case answer of
mrYes:
begin
if saveBrakhFile then
Action := caFree
else
Action := caNone;
end;
mrNo: Action := caFree;
mrCancel: Action := caNone;
end;
end;

end;

function analyseParams: string; // The name of the file when opened is not through the program
var
params: string;
i: integer;
begin
params:="";
if ParamCount>0 then
for i := 1 to ParamCount do
begin
params := params + ParamStr(i);
if i<>ParamCount then params := params + ' ';
end;
result := params;
end;

procedure TEditorForm.FormCreate(Sender: TObject);
var path : string;
begin
// Initialise:
FScale := 1; // Default Scale
PBH := pbMain.height;
PBW := pbMain.Width;
pbMain.Width := round(pbMain.Width*Fscale);
pbMain.Height := round(pbMain.height*Fscale);
tbFigNone.Down := true;
actPast.Enabled := false;
currpath := "";
Self.DoubleBuffered := true;
switchChangedStatus(false);
createFigList(FigHead);
CurrType := None;
EM := NoEdit;
CurrFigure := nil;
clearScreen;
CoppFigure := nil;

// UNDO STACK
CreateStack(USVertex);
path := analyseParams; // Анализируем входные параметры, открыта ли программа
// открытием .brakh-файла

```

```

if path <> " then
begin
    removeAllList(FigHead);
    changePath(path);
    readFile(FigHead, path);
end;
end;

procedure TEditorForm.FormDestroy(Sender: TObject);
begin
    removeAllList(FigHead);
    Dispose(FigHead);
    UndoStackClear(USVertex);
    Dispose(USVertex);
end;

procedure TEditorForm.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
var
    UndoRec: TUndoStackInfo;
begin
    if (key = VK_DELETE) and (ClickFigure <> nil) then
    begin
        // CHANGES STACK PUSHING START
        UndoRec.adr := ClickFigure;
        UndoRec.PrevFigure := removeFigure(FigHead, ClickFigure);
        UndoRec.ChangeType := chDelete;
        actUndo.Enabled := true;
        UndoStackPush(USVertex, UndoRec);
        // CHANGES STACK PUSHING END
        switchChangedStatus(true);
        ClickFigure := nil;
        Self.clearScreen;
        drawFigure(pbMain.canvas, FigHead, FScale);
    end;
    if (key = VK_RETURN) and (ClickFigure <> nil) and (ClickFigure.Info.tp <> Line) then
    begin
        // CHANGES STACK PUSHING START
        UndoRec.adr := ClickFigure;
        UndoRec.text := ClickFigure.Info.Txt;
        UndoRec.ChangeType := chChangeText;
        UndoStackPush(USVertex, UndoRec);
        actUndo.Enabled := true;
        // CHANGES STACK PUSHING END

        // Change Caption of current figure
        ClickFigure.Info.Txt := ShortString(edtRectText.Text);
        pbMain.Repaint;
    end;

    //ShowMessage( IntToStr(key) );

    // SHORTCUT FOR SCALE UP : ctrl + "+"
    if (GetKeyState( VK_OEM_PLUS ) < 0) and (GetKeyState(VK_CONTROL) < 0) then
    begin
        tbSelectScale.Position := tbSelectScale.Position + 1;
        tbSelectScale.Update;
    end;

    // SHORTCUT FOR SCALE DOWN : ctrl + "-"

```

```

if (GetKeyState( VK_OEM_MINUS ) < 0) and (GetKeyState(VK_CONTROL) < 0) then
begin
    tbSelectScale.Position := tbSelectScale.Position - 1;
    tbSelectScale.Update;
end;
end;

procedure TEditorForm.FormKeyPress(Sender: TObject; var Key: Char);
begin
    // REMOVE BEEEEEEEEEP WHERE PRESSED "ENTER"
    if (Key = #13) and (ClickFigure <> nil) and (ClickFigure.Info.tp <> Line) then
    begin
        Key := #0;
    end;
end;

procedure TEditorForm.FormResize(Sender: TObject);
begin
    pbMain.Repaint;
end;

// Return only filename, delete other path
// Example: input: C:/data/input.brakh
//          output: input.brakh
function ExtractFileNameEx(FileName:string):string;
var
    i:integer;
begin
    i:=Length(FileName);
    if i<>0 then
    begin
        while (FileName[i]<>'\'') and (i>0) do
        begin
            i:=i-1;
            Result:=Copy(FileName,i+1,Length(FileName)-i);
        end;
    end;
end;

function TEditorForm.openFile(mode: TFileMode):string;
begin
    Result := "";
    case mode of
        FSVG:
            begin
                OpenFileDialog1.DefaultExt := 'svg';
                OpenFileDialog1.Filter := 'SVG|*.svg';
            end;
        FBRakh:
            begin
                OpenFileDialog1.DefaultExt := 'brakh';
                OpenFileDialog1.Filter := 'Source-File|*.brakh';
            end;
    end;
    if OpenFileDialog1.Execute then
    begin
        Result := OpenFileDialog1.FileName;
    end;
end;

```

```

// SAVING BMP FILE
procedure TEditorForm.saveBMPFile;
var
  path: string;
  oldScale: real;
const
  ExportScale = 4; // Create good DPI for image :)
begin
  oldScale := FScale; // Save old scale
  path := saveFile(FBmp); // Getting path of file
  if path <> '' then
    begin
      ClickFigure := nil;
      with TBitMap.Create do begin // Create bitmap
        // Change bitmap size
        width := pbMain.Width*ExportScale;
        height := pbMain.Height*ExportScale;

        // Change Scale for image
        FScale := ExportScale;

        // Draw figure for bitmap canvas
        drawFigure(Canvas, FigHead, ExportScale, false);

        FScale := oldScale; // Return old scale
        SaveToFile(path); // Save bmp
        free; // free bitmap
      end;
    end;
end;

// Change canvas size
procedure TEditorForm.changeCanvasSize(w,h: Integer; flag: Boolean = true);
var
  UndoRec: TUndoStackInfo;
begin
  if flag then
    begin
      // CHANGES STACK PUSHING START
      UndoRec.ChangeType := chCanvasSize;
      UndoRec.w := PBW;
      UndoRec.h := PBH;
      UndoStackPush(USVertex, UndoRec);
      actUndo.Enabled := true;
      // CHANGES STACK PUSHING EDD
    end;
  PBH := h; // Update global size (for scale = 1)
  PBW := w;
  useScale(W, H); // Use scale for height and width
  pbMain.width := w; // update sizes
  pbMain.height := h;
end;

// Create new diagram
procedure TEditorForm.newFile;
begin
  Self.Caption := rsNewFile + ' - Syntax Diagrams';
  removeAllList(FigHead);

```

```

    currpath := "";
    switchChangedStatus(false);
    pbMain.Repaint;
end;

procedure TEditorForm.mniNewClick(Sender: TObject);
var
    answer: Integer;
begin
    answer := MessageDlg(rsNewFileDlg,mtCustom,[mbYes,mbNo], 0);
    if answer = mrYes then
        begin
            newFile;
        end;
    end;
end;

// Change path
procedure TEditorForm.changePath(path: string);
var
    FileName: string;
begin
    FileName := ExtractFileNameEx(path);
    Self.Caption := FileName + ' - Syntax Diagrams';
    currpath := path;
end;

function TEditorForm.saveFile(mode: TFileMode):string;
begin
    Result := "";
    case mode of
        FSvg:
            begin
                saveDialog1.FileName := 'SyntaxDiagrams.svg';
                saveDialog1.Filter := 'SVG|*.svg';
                saveDialog1.DefaultExt := 'svg';
            end;
        FBrakh:
            begin
                saveDialog1.FileName := 'SyntaxDiagrams.brakh';
                saveDialog1.Filter := 'Source-File|*.brakh';
                saveDialog1.DefaultExt := 'brakh';
            end;
        FBmp:
            begin
                saveDialog1.FileName := 'SyntaxDiagrams.bmp';
                saveDialog1.Filter := 'Bitmap Picture|*.bmp';
                saveDialog1.DefaultExt := 'bmp';
            end;
        FPng:
            begin
                saveDialog1.FileName := 'SyntaxDiagrams.png';
                saveDialog1.Filter := 'PNG|*.png';
                saveDialog1.DefaultExt := 'png';
            end;
    end;
    if SaveDialog1.Execute then
        begin
            Result := SaveDialog1.FileName;
        end;
    end;
end;

```

```

end;

end;

// EXPORT TO PNG FILE
procedure TEditorForm.savePNGFile;
var
  path: string;
  oldScale: real;
  png : TPngImage;
  bitmap: TBitmap;
const
  ExportScale = 4;
begin
  oldScale := FScale; // Save old scale
  path := saveFile(FPng); // Get file path
  if path <> '' then
    begin
      ClickFigure := nil;
      try
        bitmap := TBitMap.Create; // create bitmap
        with bitmap do
          begin
            png := TPNGImage.Create; // Create PNGimage
            width := pbMain.Width*ExportScale; // Change bitmap size
            height := pbMain.Height*ExportScale;
            FScale := ExportScale; // Scale for image
            drawFigure(Canvas, FigHead, ExportScale, false); // Drawing figure for bitmap canvas
            FScale := oldScale; // return old scale
          end;
          png.Assign(bitmap); // SAVE TO PNG:
          png.Draw(bitmap.Canvas, Rect(0, 0, bitmap.Width, bitmap.Height));
          png.SaveToFile(path)
        finally
          bitmap.free;
          png.free;
        end;
      end;
    end;
end;

// SAVE SOURCE FILE
function TEditorForm.saveBrakhFile:boolean;
var
  path: string;
begin
  Result:=false;
  path := saveFile(FBrakh);
  if path <> '' then
    begin
      saveToFile(FigHead, path);
      changePath(path);
      switchChangedStatus(False);
      Result := true;
    end;
  end;
end;

// EXPORT TO SVG
procedure TEditorForm.saveSVGFile;

```

```

var
  path: string;
begin
  path := saveFile(FSvg);
  if path <> '' then
    ExportTOSvg(FigHead, pbMain.Width, pbMain.Height, path, 'Syntax Diagram Project', 'Create by BrakhMen.info');
end;

procedure TEditorForm.sbMainMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if dm = ResizeCanvas then
    begin
      changeCanvasSize(Round(X/FScale),Round(Y/FScale));
      DM := oldDM;
      actResizeCanvas.Enabled := true;
      tbResizeCanvas.Down := false;
      pbMain.Repaint;
    end;
end;

procedure TEditorForm.sbMainMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  if dm = ResizeCanvas then
    begin
      updateCanvasSizeWithCoords(x, y);
    end;
end;

procedure TEditorForm.sbMainMouseWheelDown(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if ssShift in Shift then
    begin
      with (Sender as TScrollBar).HorzScrollBar do
        Position := Position + Increment;
    end
  else
    begin
      with (Sender as TScrollBar).VertScrollBar do
        Position := Position + Increment;
    end;
end;

procedure TEditorForm.sbMainMouseWheelUp(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if ssShift in Shift then
    begin
      with (Sender as TScrollBar).HorzScrollBar do
        Position := Position - Increment;
    end
  else
    begin
      with (Sender as TScrollBar).VertScrollBar do
        Position := Position - Increment;
    end;
end;

```

```

procedure TEditorForm.SD_Resize;
begin
    self.resize;
end;

// ACTIONS IMPLIMENTATION
procedure TEditorForm.actAboutSBExecute(Sender: TObject);
begin
    // Open the form with displaying HTML
    FHTml.showHTML(rsHelpHowIsSD_Caption,rsHelpHowIsSD_ResName);
end;

// Open the form with the form size settings
procedure TEditorForm.actCanvasSizeExecute(Sender: TObject);
var
    neww, newh : integer;
begin
    neww:= PBW;
    newh := PBH;
    FCanvasSettings.showForm(neww, newh); // Open form
    changeCanvasSize(neww, newh);
    Self.Repaint;
end;

procedure TEditorForm.actChangeMagnetizeExecute(Sender: TObject);
begin
    ;
end;

procedure TEditorForm.actCopyExecute(Sender: TObject);
begin
    CoppyFigure := ClickFigure;
    actPast.Enabled := true;
end;

procedure TEditorForm.actExportBMPEXecute(Sender: TObject);
begin
    saveBMPFile;
end;

procedure TEditorForm.actExportSVGExecute(Sender: TObject);
begin
    saveSVGFile;
end;

procedure TEditorForm.actFigDefExecute(Sender: TObject);
begin
    if CurrType = Line then
        endDrawLine;
    CurrType := def;
end;

procedure TEditorForm.actFigLineExecute(Sender: TObject);
begin
    CurrType := Line;
    CurrLineType := LLine;
end;

```



```

end;

procedure TEditorForm.actFigMetaConstExecute(Sender: TObject);
begin
    if CurrType = Line then
        endDrawLine;
    CurrType := MetaConst;
end;

procedure TEditorForm.actFigMetaVarExecute(Sender: TObject);
begin
    if CurrType = Line then
        endDrawLine;
    CurrType := MetaVar;
end;

procedure TEditorForm.actFigNoneExecute(Sender: TObject);
begin
    if CurrType = Line then
        endDrawLine;
    CurrType := None;
end;

procedure TEditorForm.actHelpExecute(Sender: TObject);
begin
    FHTml.showHTML(rsHelp_Caption,rsHelp_ResName);
end;

procedure TEditorForm.Action1Execute(Sender: TObject);
begin
    showMessage('kek');
end;

// Create new Diagram
procedure TEditorForm.actNewExecute(Sender: TObject);
var
    answer: Integer;
begin
    answer := MessageDlg(rsNewFileDlg,mtCustom,[mbYes,mbNo], 0);
    if answer = mrYes then
        begin
            newFile;
            actUndo.Enabled := false;
            UndoStackClear(USVertex);
        end;
end;

end;

// Open file
procedure TEditorForm.actOpenExecute(Sender: TObject);
var
    path: string;
    answer: integer;
begin
    if isChanged then // if the diagram is changed, it is suggested to save the file
        begin
            answer := MessageDlg(rsExitDlg,mtCustom,
                [mbYes,mbNo,mbCancel], 0);
            case answer of
                mrYes:

```

```

begin
  if not saveBrakhFile then exit // Save file
end;
mrNo: ;
mrCancel: exit;
end;
end;

path := openFile(FBrakh);
if path <> " then
begin

  removeAllList(FigHead);
  actUndo.Enabled := false;
  UndoStackClear(USVertex);
  if readFile(FigHead, path) then
  begin
    changePath(path);
    switchChangedStatus(False);
  end
  else
    newFile;
  end;
end;

procedure TEditorForm.actPastExecute(Sender: TObject);
begin

  actPast.Enabled := false;
  if CoppyFigure = nil then exit;

  CopyFigure(FigHead, CoppyFigure); // Create copy of CoppyFigure
end;

procedure TEditorForm.actPNGExecute(Sender: TObject);
begin
  savePNGFile;
end;

procedure TEditorForm.actResizeCanvasExecute(Sender: TObject);
begin
  tbResizeCanvas.Down := true;
  if DM <> ResizeCanvas then
  begin
    oldDM := DM;
    DM := ResizeCanvas;
  end;
end;

procedure TEditorForm.actSaveAsExecute(Sender: TObject);
begin
  saveBrakhFile;
end;

procedure TEditorForm.actSaveExecute(Sender: TObject);
begin
  if currpath <> " then
  begin
    saveToFile(FigHead, currpath);
    switchChangedStatus(False);
  end;
end;

```

```

end
else
    saveBrakhFile;
end;

procedure TEditorForm.actUndoExecute(Sender: TObject);
var
    undoRec: TUndoStackInfo;
begin
    if undoStackPop(USVertex, undoRec) then // "Pop" an item from the stack
    begin
        undoChanges(undoRec, Canvas); // cancel changes
        if mniMagnetizeLine.Checked then
            MagnetizeLines(FigHead);
        end;

        if isEmptyStack(USVertex) then (Sender as TAction).Enabled := false;
        ClickFigure := nil;
        pbMain.Repaint;
    end;

end.

```

Модуль формы FHTML

```

unit FHtmlView;

interface

uses
    Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
    Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.OleCtrls, SHDocVw, Vcl.Menus;

type
    TFHtml = class(TForm)
        WebBrowser1: TWebBrowser;
        pmHtmlMenu: TPopupMenu;
        pmiClose: TMenuItem;
        procedure pmiCloseClick(Sender: TObject);
    private
        procedure WMMouseActivate(var Msg: TMessage); message WM_MOUSEACTIVATE;
    public
        procedure showHTML(title, htmlres: WideString);
    end;

var
    FHtml: TFHtml;

implementation

{$R *.dfm}

// Убираем стандартное контекстное меню TWebBrowser и показываем
// Свое
procedure TFHtml.WMMouseActivate(var Msg: TMessage);
begin

```

```

        try
            inherited;
            //Анализируем, какая кнопка мыши нажата
            if Msg.LParamHi = 516 then // если правая
                // показываем свое меню
                pmHtmlMenu.Popup(Mouse.CursorPos.x, Mouse.CursorPos.y);
                Msg.Result := 0;
        except

        end;

end;

procedure TFHtml.pmiCloseClick(Sender: TObject);
begin
    Self.Close;
end;

// Отображение HTML страницы из ресурсов.
procedure TFHtml.showHTML(title, htmlres: WideString);
var
    s: WideString;
    Flags, TargetFrameName, postData, Headers: OleVariant;
begin
    Self.Caption := title;
    WebBrowser1.Navigate('res://' + Application.ExeName + '/' + htmlres,
        Flags, TargetFrameName, postData, Headers);
    Self.ShowModal;
end;

end.

```

Модуль формы FCanvasSettings

```

unit FCanvasSizeSettings;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls;

type
    TFCanvasSettings = class(TForm)
        Panel1: TPanel;
        Label1: TLabel;
        lblWidth: TLabel;
        lblHeight: TLabel;
        edtWidth: TEdit;
        edtHeight: TEdit;
        btnOk: TButton;
        btnCancel: TButton;
        lbl1px: TLabel;
        Label2: TLabel;
    private
        procedure ControlsToItem(var w,h: integer);
    public
        function showForm(var w,h: integer):TModalResult;
    end;

```

```

var
  FCanvasSettings: TFCanvasSettings;

implementation
uses main;
{$R *.dfm}

// Возвращает в w,h размеры, введенные пользователем в текстовом поле
procedure TFCanvasSettings.ControlsToItem(var w, h: integer);
begin
  try
    w := StrToInt( edtWidth.Text );
    h := StrToInt( edtHeight.Text );
  except on E: EConvertError do
    ShowMessage('Ошибка ввода');
  end;
end;

function TFCanvasSettings.showForm(var w,h: integer):TModalResult;
begin
  edtWidth.Text := IntToStr(w);
  edtHeight.Text := IntToStr(h);
  Result := Self.ShowModal;
  if Result = mrOk then
    ControlsToItem(w,h);
end;

end.

```

Модуль Model

```

unit Model;
// MODEL par in MVC:
// responsible for processing information
interface

uses Data.Types, vcl.graphics, View.Canvas,vcl.dialogs, Data.InitData, math,
  View.SVG, Model.Files, Model.Lines;

function getClickFigure(x,y:integer; head: PFigList):PFigList;
function removeFigure(head: PFigList; adr: PFigList):PFigList;
procedure removeAllList(head:PFigList);
procedure ChangeCoords(F: PFigList; EM: TEditMode; x,y:integer; var TmpX, TmpY: integer);
procedure createFigList(var head: PFigList);
function addFigure(head: PFigList; x,y: integer; ftype: TType; Text:String = 'Kek'):PFigList;
function nearRound(x:integer):integer;
procedure roundCoords(var x,y:integer);
function getEditMode(status: TDrawMode; x,y: Integer; head: PFigList; CT: TType) :TEditMode;
procedure checkFigureCoord(R: PFigList);
procedure copyFigure(head: PFigList; copyfigure:PFigList);
procedure MagnetizeLines(head: PFigList);
function ScaleRound(scale: real; x: integer): integer;
procedure undoChanges(UndoRec: TUndoStackInfo; Canvas: TCanvas);
procedure SearchFiguresInOneLine(head, curr: PFigList);

implementation
uses System.Sysutils, main;

procedure createFigList(var head: PFigList);

```

```

begin
  new(head);
  head.Adr := nil;
end;

function nearRound(x:integer):integer;
begin
  Result:= round(x/NearFigure)*NearFigure;
end;

// Проверка координат, чтобы выполнялись условия  $x_2 > x_1$ ,  $y_2 > y_1$ 
// Если условие не выполняется - процедура меняет координаты местами
procedure checkFigureCoord(R: PFigList);
var
  temp:integer;
begin
  if (R<>nil) and (R^.Info.tp <> Line) then
    with R^.Info do
      begin
        if  $x_1 > x_2$  then
          begin
            temp :=  $x_1$ ;
             $x_1$  :=  $x_2$ ;
             $x_2$ :=temp;
          end;
        if  $y_1 > y_2$  then
          begin
            temp :=  $y_1$ ;
             $y_1$  :=  $y_2$ ;
             $y_2$ :=temp;
          end;
        end;
      end;
end;

function ScaleRound(scale: real; x: integer):integer;
begin
  Result := Round(X*Scale);
end;

// Создание копии фигуры
procedure copyFigure(head: PFigList; copyfigure:PFigList);
var
  newfigure: TFigureInfo;
  tmp: PFigList;
begin
  newfigure := copyfigure^.Info;
  if copyfigure^.Info.tp = Line then
    newfigure.PointHead := copyPointList(copyfigure^.Info.PointHead);
  if head = nil then exit;
  tmp := head;
  while tmp^.Adr <> nil do
    begin
      tmp := tmp^.Adr;
    end;
  new(tmp^.adr);
  tmp := tmp^.Adr;
  tmp^.Adr := nil;

```

```

tmp^.Info := newfigure;

end;

// Функция возвращает тип режима редактирования в зависимости от того,
// Что находится во координатам наведения мыши
function getEditMode(status: TDrawMode; x,y: Integer; head: PFigList; CT: TType) :TEditMode;
var
  r:TFigureInfo;
  temp: PFigList;
  tmpPoint: PPointsList;
begin
  temp := head;
  while temp <> nil do
  begin
    R := temp^.Info;
    if (status = nodraw) and (R.tp <> Line) then
    begin
      if ( (x > R.x1) and (x < R.x2) and (y > R.y1) and (y < R.y2)) then
      begin
        // Внутри объекта
        Result := Move;
      end
      else
      begin
        // За пределами объекта
        Result := NoEdit;
      end;
    end;
    if ( (x > R.x1) and (x < R.x2) and ((abs(y - R.y1) < Tolerance) or (abs(y - R.y2) < Tolerance))) then
    begin
      // Горизонтальная сторона
      if (abs(y - R.y1) < Tolerance) then
        Result := TSide
      else
        Result:= BSide;
      end;
    end;
    if ( (y > R.y1) and (y < R.y2) and ((abs(x - R.x1) < Tolerance) or (abs(x - R.x2) < Tolerance))) then
    begin
      // Вертикальная сторона
      if (abs(x - R.x1) < Tolerance) then
        Result := Lside
      else
        Result:= RSide;
      end;
    end;
    if ((abs(y-R.y1) < Tolerance) and (abs(x-R.x1) < Tolerance)) then
    begin
      // Левая верхняя вершина
      Result := Vert1;
    end;
    if (abs(y-R.y1) < Tolerance) and (abs(x-R.x2) < Tolerance) then
    begin
      // Правая верхняя вершина
      Result := Vert2;
    end;
    if (abs(y-R.y2) < Tolerance) and (abs(x-R.x1) < Tolerance) then
    begin
      // Левая нижняя вершина
      Result := Vert3;
    end;
    if (abs(y-R.y2) < Tolerance) and (abs(x-R.x2) < Tolerance) then

```

```

begin
  // Правая нижняя вершина
  Result := Vert4;
  // ?? ?? ??? ?? ??? ?? ?? \
end;
if result <> NoEdit then
begin
  CurrFigure := temp;
  exit;
end;
end
else if (status = nodraw) then
begin
  tmpPoint := R.PointHead;
  while tmpPoint <> nil do
  begin
    if (abs(y-tmpPoint^.Info.y) < Tolerance) and (abs(x-tmpPoint^.Info.x) < Tolerance) then
    begin
      CurrFigure := temp;
      currPointAdr := tmpPoint;
      Result := Move;
      exit;
    end;
    { ToDo: KEK }
    tmpPoint := tmpPoint^.Adr;
  end;
  if (CT = TType(4)) and (isBelongsLine(temp^.Info.PointHead, x,y)) then
  begin
    CurrFigure := temp;
    Result := LineMove;
    Exit;
  end;
end;
temp := temp^.Adr;
end;
Result := NoEdit;
end;

// Округление координат с заданным шагом.
procedure roundCoords(var x,y:integer);
begin
  x := round(x/step_round)*step_round;
  y := round(y/step_round)*step_round;

  searchNearLine(FigHead, x,y);
end;

// Добавление новой фигуры и возврат ссылки на нее
function addFigure(head: PFigList; x,y: integer; ftype: TType; Text:String = 'Kek'):PFigList;
var
  tmp: PFigList;
begin
  tmp := head;
  while tmp^.adr <> nil do
    tmp := tmp^.Adr;
  new(tmp^.adr);
  tmp := tmp^.Adr;
  tmp^.Adr := nil;

```



```

with tmp^.Info do
begin
  // По-умолчанию - линия - точка. В дальнейшем - размеры подстроятся под
  // размеры текста (при отрисовке)
  x1 := x;
  x2 := x;
  y1 := y;
  y2 := y;
  Txt := ShortString(text);
  Tp := ftype;
end;

Result := tmp;
end;

// Возвращает фигуру, по которой был клик
function getClickFigure(x,y:integer; head: PFigList):PFigList;
var
  tmp:PFigList;
  tmpP: PPointsList;
begin
  tmp := head^.adr;

  while tmp <> nil do
  begin
    if tmp^.Info.tp <> Line then
    begin
      if (x > tmp^.Info.x1) // Если точка клика принадлежит прямоугольной поверхности
        and // То возвращаем фигуру
        (x < tmp^.Info.x2)
        and
        (y > tmp^.Info.y1)
        and
        (y < tmp^.Info.y2)
      then
      begin
        result := tmp;
        exit;
      end;
    end
    else
    begin
      if (tmp^.Info.PointHead = nil) or (tmp^.Info.PointHead^.Adr = nil) then
      begin
        tmp := tmp^.adr;
        continue;
      end;
      tmpP := tmp^.Info.PointHead^.Adr;
      while tmpP <> nil do
      begin
        // Точка принадлежит вершине
        if (abs(y-tmpP^.Info.x) <= Tolerance) and (abs(x-tmpP^.Info.y) <= Tolerance) then
        begin
          result := tmp;
          exit;
        end;
      end;
    end;
  end;
end;

```

```

    if tmpP^.Adr <> nil then
    begin
        // Точка принадлежит отрезку
        if ((abs(y - tmpP^.Info.y) <= Tolerance*2 ) and (x > min(tmpP^.Info.x, tmpP^.adr^.Info.x)) and (x <= max(tmpP^.Info.x,
tmpP^.adr^.Info.x)) )
            or
            ((abs(x - tmpP^.Info.x) <= Tolerance*2) and (y > min(tmpP^.Info.y, tmpP^.adr^.Info.y)) and (y <= max(tmpP^.Info.y,
tmpP^.adr^.Info.y))) then
        begin
            Result:= tmp;
            exit;
        end;

    end;
    tmpP := tmpP^.Adr;

end;

end;
tmp := tmp^.Adr;
end;
Result := nil;
end;

// Функция выполняет ЛОГИЧЕСКОЕ удаление фигуры и возвращает ссылку
// На предшествующую удаленной фигуру фигуру.
// Потребность в ЛОГИЧЕСКОМ удалении обусловлено тем, что необходимо
// предусмотреть возможность отменить изменение и восстановить
// фигуру
function removeFigure(head: PFigList; adr: PFigList):PFigList;
var
    temp,temp2:PFigList;
begin
    temp := head;
    while temp^.adr <> nil do
    begin
        temp2 := temp^.adr;
        if temp2 = adr then
        begin
            temp^.adr := temp2^.adr;
            Result := temp;
        end
        else
            temp:= temp^.adr;
        end;
    end;
end;

// Полностью удалить список фигур (Кроме головы)
procedure removeAllList(head:PFigList);
var
    temp, temp2: PFigList;
begin
    temp := head^.Adr;
    while temp <> nil do
    begin
        temp2:=temp^.Adr;
        dispose(temp);
        temp:=temp2;
    end;
end;

```

```

head.Adr := nil;
end;

// Редактирование фигуры и редактирование ее координат
procedure ChangeCoords(F: PFigList; EM: TEditMode; x,y:integer; var TmpX, TmpY: integer);
var
  oldp: TPointsInfo;
begin
  if F <> nil then
    case EM of
      NoEdit:
        begin

        end;
      Move: // Перемещаем объект :)
        begin
          if F^.Info.tp = Line then
            begin
              oldp:= currPointAdr^.Info;
              currPointAdr^.Info.x := currPointAdr^.Info.x - (TmpX - x);
              currPointAdr^.Info.y := currPointAdr^.Info.y - (Tmpy - y);
              MoveLine(CurrFigure^.Info.PointHead, oldp, currPointAdr^.Info);
            end
          else
            begin
              // Смещаем объект
              // TmpX, TmpY - смещение координат относительно прошлого вызова события
              F^.Info.x1 := F^.Info.x1 - (TmpX - x);
              F^.Info.x2 := F^.Info.x2 - (TmpX - x);
              F^.Info.y1 := F^.Info.y1 - (Tmpy - y);
              F^.Info.y2 := F^.Info.y2 - (TmpY - y);
            end;
          end;
          LineMove:
          begin
            moveALILinePoint(CurrFigure^.Info.PointHead, (TmpX - x), (Tmpy - y));
          end;
          TSide:
          begin
            // смещаем верхнюю сторону
            F^.Info.y1 := F^.Info.y1 - (Tmpy - y);
          end;
          BSide:
          begin
            // Смещаем нижнюю сторону
            F^.Info.y2 := F^.Info.y2 - (Tmpy - y);
          end;
          RSide:
          begin
            // Смещаем правую сторону
            F^.Info.x2 := F^.Info.x2 - (Tmpx - x);
          end;
          LSide:
          begin
            // Смещаем левую сторону
            F^.Info.x1 := F^.Info.x1 - (Tmpx - x);
          end;
          Vert1:
          begin
            F^.Info.x1 := F^.Info.x1 - (TmpX - x);

```

```

    F^.Info.y1 := F^.Info.y1 - (Tmpy - y);
end;
Vert2:
begin
    F^.Info.x2 := F^.Info.x2 - (TmpX - x);
    F^.Info.y1 := F^.Info.y1 - (Tmpy - y);
end;
Vert3:
begin
    F^.Info.x1 := F^.Info.x1 - (TmpX - x);
    F^.Info.y2 := F^.Info.y2 - (Tmpy - y);
end;
Vert4:
begin
    F^.Info.x2 := F^.Info.x2 - (TmpX - x);
    F^.Info.y2 := F^.Info.y2 - (Tmpy - y);
end;
end;
end;

// Функция "Примагничивает" точку к фигуре и возвращает true, если
// "примагничивание" удалось
function magnetizeWithFigures(head: PFigList; Point: PPointsList):boolean;
var
    temp: PFigList;
begin
    temp := head^.adr;
    Result := false;
    while temp <> nil do
        begin
            if temp^.Info.tp <> Line then
                begin
                    if (abs( Point^.Info.x - temp^.Info.x1) < NearFigure) // Левая грань
                        and
                        ( Point^.Info.y < temp^.Info.y2 )
                        and
                        ( Point^.Info.y > temp^.Info.y1 )
                    then
                        begin
                            Result := true;
                            Point^.info.x := temp^.Info.x1;
                            point^.Info.y := (temp^.Info.y1 + temp^.Info.y2) div 2;
                        end
                    else if (abs( Point^.Info.x - temp^.Info.x2) < NearFigure) // Правая грань
                        and
                        ( Point^.Info.y < temp^.Info.y2 )
                        and
                        ( Point^.Info.y > temp^.Info.y1 )
                    then
                        begin
                            Result := true;
                            Point^.info.x := temp^.Info.x2;
                            point^.Info.y := (temp^.Info.y1 + temp^.Info.y2) div 2;
                        end;
                    end;
                end;
            temp := temp^.Adr;
        end;
    end;
end;

// Поиск фигур, расположенных приблизительно в одну линию
// И изменение координат так, чтобы они оказались точно

```

```

// в одной линии
procedure SearchFiguresInOneLine(head, curr: PFigList);
var
  temp: PFigList;
  CurrY : Integer;
  tempY : integer;
begin
  with curr^.Info do
  begin
    CurrY := y1 + (y2 - y1) div 2; // Центр по Y переданной фигуры
  end;
  temp := head^.Adr;
  while temp <> nil do
  begin
    if (temp^.Info.tp = line) or (temp = curr) then
    begin
      temp := temp^.Adr;
      continue;
    end;
    with temp^.Info do
    begin
      tempY := y1 + (y2 - y1) div 2; // Центр по Y текущей фигуры
    end;
    if abs( CurrY - tempY ) < NearFigure then
    begin
      temp^.Info.y1 := curry - (Temp^.Info.y2 - Temp^.Info.y1) div 2;
      temp^.Info.y2 := curry + (Temp^.Info.y2 - Temp^.Info.y1) div 2;
    end;

    temp := temp^.Adr;
  end;
end;

// "Примагничивание" линий к другим фигурам
procedure MagnetizeLines(head: PFigList);
var
  tmp: PFigList;
  tmpP: PPointsList;
  NearP: PPointsList;
  x,y : integer;
  oldP, newP: TPointsInfo;
  prevP: PPointsList;
begin
  tmp := head^.adr;
  while tmp <> nil do
  begin
    if tmp^.Info.tp <> Line then
    begin
      SearchFiguresInOneLine(head, tmp); // Пробуем найти фигуры в одну линию
      tmp := tmp^.Adr;
      continue;
    end;
    prevP:=nil;
    tmpP:= tmp^.Info.PointHead^.Adr;
    while tmpP <> nil do
    begin
      x := tmpP^.Info.x;
      y := tmpP^.Info.y;

```

```

// Пробуем примагнитить линию к текстовой фигуре
if (isHorLine(tmpP, prevP)) and (magnetizeWithFigures(head, tmpP)) then
begin
  oldp.x := x;
  oldp.y := y;
  MoveLine(tmp^.Info.PointHead, oldP, tmpP^.Info);
  tmpP := tmpP^.adr;
  continue;
end;

// Пробуем примгнитить линию к другой линии
NearP := searchNearLine(head, x, y);
if NearP <> nil then
begin
  oldp.x := tmpP^.Info.x;
  oldp.y := tmpP^.Info.y;
  newP.x := nearP.Info.x;
  newP.y := tmpP^.Info.y;

  if abs(tmpP^.Info.x - nearP.Info.x) < nearFigure then
  begin
    tmpP^.Info := newP;
    MoveLine(tmp^.Info.PointHead, oldP, tmpP^.Info);
  end;
  newP.x := tmpP^.Info.x;
  newP.y := nearP.Info.y;
  if abs(tmpP^.Info.y - nearP.Info.y) < nearFigure then
  begin
    tmpP^.Info := newP;
    MoveLine(tmp^.Info.PointHead, oldP, tmpP^.Info);
  end;
end;
prevP := tmpP;
tmpP := tmpP^.Adr;
end;
tmp := tmp^.Adr;

end;
end;

// Отмена изменений
procedure undoChanges(UndoRec: TUndoStackInfo; Canvas: TCanvas);
var
  tmp: PFigList;
  tmpP: PPointsList;
begin
  case UndoRec.ChangeType of
    chDelete:
      begin
        // Снова возвращаем фигуру (Она не была удалена физически, только логически)
        tmp := UndoRec.adr;
        tmp.Adr := UndoRec.PrevFigure^.Adr;
        undoRec.PrevFigure^.Adr := tmp;
      end;
    chAddPoint:
      begin
        // Удаление точки (физическое)
        tmpP := UndoRec.PrevPointAdr^.adr;
        UndoRec.PrevPointAdr^.Adr := nil;
        Dispose(tmpP);
      end;
  end;
end;

```

```

end;
chInsert:
begin
  // Удаление фигуры
  removeFigure(EditorForm.getFigureHead, UndoRec.adr)
end;
chFigMove:
begin
  // Возврат предыдущих координат фигуры
  UndoRec.adr^.Info := UndoRec.PrevInfo;
end;
chPointMove:
begin
  // Возврат предыдущих координат точек линии
  changeLineCoordsFromStr(UndoRec.adr^.Info.PointHead, UndoRec.st);
end;
chChangeText:
begin
  // Возврат предыдущего текста
  UndoRec.adr^.Info.Txt := UndoRec.text;
end;
chCanvasSize:
begin
  // Возврат прошлых размеров полотна
  EditorForm.changeCanvasSize(UndoRec.w, UndoRec.h, false);
end;
NonDeleted: // Ничего не делаем :)
;
end;
end;
end.

```

Модуль Model.Lines

```
unit Model.Lines;
```

```
interface
```

```
uses Data.Types, Data.InitData;
```

```
function isHorisontalIntersection(head: PFigList; blocked: PPointsList): boolean;
```

```
function needMiddleArrow(tmp: PPointsList; FirstP: TPointsInfo) :Boolean;
```

```
function addLine(head: PFigList; x,y: integer):PFigList;
```

```
function addNewPoint(var head: PPointsList; x,y:integer):PPointsList;
```

```
function copyPointList(cf: PPointsList):PPointsList;
```

```
function isBelongsLine(head: PPointsList; x,y: integer): Boolean;
```

```
procedure MoveLine(head: PPointsList; oldp, newp: TPointsInfo);
```

```
procedure moveALLLinePoint(head: PPointsList; dx, dy: integer);
```

```
function isHorLine(curr, prev: PPointsList):boolean;
```

```
procedure changeLineCoordsFromStr(head: PPointsList; st:string);
```

```
function searchNearLine(head: PFigList; var x,y: integer):PPointsList;
```

```
procedure removeTrashLines(head: PFigList; curr: PFigList);
```

```
implementation
```

```
uses math, System.SysUtils, vcl.dialogs, Model;
```

```
// Парсинг строки формата "X1/Y1""X2/Y2"... и изменение
```

```
// Координат списка точек линии на координаты, полученные
```

```
// при парсинге
```

```
procedure changeLineCoordsFromStr(head: PPointsList; st:string);
```

```
var
```

```

tmp: PPointsList;
xy: string;
begin
tmp:= head^.Adr;
if st <> " then
begin
Delete(st,1,1);
st := st + "";
end;
while (length(st) <> 0) and (st <> "") do
begin
xy := copy(st, 1, pos("", st)-1);
Delete(st,1, pos("", st)+1);
if (st <> ") or (xy <> ") then
begin
if tmp <> nil then
begin
tmp^.Info.x := strtoint(copy(xy, 1,pos('/', xy)-1));
tmp^.Info.y := strtoint(copy(xy, pos('/', xy)+1, length(xy)));
tmp := tmp^.Adr;
end
else
begin
ShowMessage('Error (small)');
Exit;
end;
end;
end;
end;
end;

```

```

// Полная копия списка точек
function copyPointList(cf: PPointsList):PPointsList;
var
tmp: PPointsList;
begin

```

```

new(Result);
Result^.adr := nil;
tmp := cf;
if tmp = nil then exit;
tmp := tmp^.Adr;
while tmp <> nil do
begin
addNewPoint(Result, tmp^.Info.x, tmp^.Info.y);
tmp := tmp^.Adr;
end;

```

```

end;

```

```

// Добавляем линию и возвращаем ссылку на нее
function addLine(head: PFigList; x,y: integer):PFigList;
var
tmp: PFigList;
begin
tmp := head;
while tmp^.adr <> nil do
tmp := tmp^.Adr;
new(tmp^.adr);
tmp := tmp^.Adr;

```



```

tmp^.Adr := nil;
tmp^.Info.tp := line;

new(tmp^.Info.PointHead); // Создаем список точек
tmp^.Info.PointHead^.Adr := nil;
addNewPoint(tmp^.Info.PointHead, x,y); // Добавляем первую точку

result := tmp; // Возвращаем созданную линию
end;

// Удаление "Мусорных линий" (состоящих из одной точки)
procedure removeTrashLines(head: PFigList; curr: PFigList);
var
  tmp: PFigList;
begin
  if head = nil then exit;

  tmp := head^.adr;
  while tmp <> nil do
    begin
      if tmp^.Info.tp = Line then
        begin
          if (tmp^.Info.PointHead = nil) or (tmp^.Info.PointHead^.Adr = nil) then continue;

          if (tmp^.Info.PointHead^.Adr^.Adr = nil) and (tmp = curr) then
            removeFigure(head,tmp) // Точка - единственная и фигура дорисована => удаляем
          else
            if (tmp^.Info.PointHead^.Adr^.Adr <> nil)
              and
              (tmp^.Info.PointHead^.Adr.Info.x = tmp^.Info.PointHead^.Adr^.adr.Info.x)
              and
              (tmp^.Info.PointHead^.Adr.Info.y = tmp^.Info.PointHead^.Adr^.adr.Info.y)
            then
              begin
                // Добавленно много точек, но все с одними координатами => удаляем :)
                tmp^.Info.PointHead := tmp^.Info.PointHead^.Adr;
                removeTrashLines(head,curr);
              end;
            end;
            tmp := tmp^.Adr;
          end;
        end;
      end;
    end;

  // Добавление точки линии
function addNewPoint(var head: PPointsList; x,y:integer):PPointsList;
var
  tmp :PPointsList;
  id :integer;
  px, py: integer;
begin
  tmp := head;
  while tmp^.adr <> nil do
    tmp := tmp^.Adr;
  if tmp <> head then
    begin
      px := tmp^.Info.x;
      py := tmp^.Info.y;
      // Запрещаем проводить прямую под углом.
      try
        if (arctan(abs((y-py)/(x-px))) < pi/4) then

```

```

        y:=py
    else
        x:=px;
    except on EZeroDivide do

    end;
end;
new(tmp^.adr);
Result := tmp;
tmp := tmp^.adr;
tmp^.Info.x := x;
tmp^.Info.y := y;
tmp^.Adr := nil;
end;

// Процедура превращает линии под углом в вертикальные или горизонтальные
// В зависимости от угла наклона. (Все линии в синтаксических диаграммах
// Должны быть параллельны одной из осей.
procedure checkLineCoords(head: PPointsList);
var
    tmp: PPointsList;
begin
    tmp := head^.adr;
    while tmp^.adr <> NIL do
        begin
            try
                if arctan(abs((tmp^.Adr^.Info.y-tmp^.Info.y)/(tmp^.Adr^.Info.x-tmp^.Info.x))) < pi/4 then
                    tmp^.Info.y := tmp^.adr^.Info.y
                else
                    tmp^.Info.x := tmp^.adr^.Info.x;
                except on E: EZeroDivide do
                end;
                tmp := tmp^.Adr;
            end;
        end;
    end;
end;

{Идея алгоритма: найти область линии, все точки которой имеют
либо координату x, либо координату y, равную старому значению
координаты перемещаемой точки. При этом все точки области
должны идти подряд и в области не должно содержаться ни
одной точки, не соответствующих данному условию.
После нахождения области, нужно изменить координаты
каждой точки внутри области.}
procedure MoveLine(head: PPointsList; oldp, newp: TPointsInfo);
var
    tmp: PPointsList;
    BeginOfArea: PPointsList;
    isFound: Boolean;
    isEnd: boolean;
begin
    tmp:=head^.adr;
    isFound := true;
    BeginOfArea := nil;
    isEnd := false;

    // Поиск начала искомой области
    while (tmp <> nil) and not isEnd do
        begin
            if (tmp^.Info.y = oldp.y) or (tmp^.Info.x = oldp.x) or ((tmp^.Info.x = newP.x) and (tmp^.Info.y = newP.y)) then

```

```

begin
  if BeginOfArea = nil then
    BeginOfArea:= tmp;
    isFound:= true;
    if (tmp^.Info.x = newP.x) and (tmp^.Info.y = newP.y) then
      isEnd := true;
    end
  else
    begin
      BeginOfArea := nil; // Если точка противоречит одному из перечисленных выше условий,
      // то заного ищем начало области
    end;
    if not isEnd then
      tmp := tmp^.Adr;
    end;

    tmp := BeginOfArea;
    // изменение координат точек внутри области
    while (tmp<>nil) and isFound and ((tmp^.Info.y = oldp.y) or (tmp^.Info.x = oldp.x) or ((tmp^.Info.x = newP.x) and (tmp^.Info.y =
newP.y) )) do
      begin
        if tmp^.Info.y = oldp.y then
          tmp^.Info.y := newp.y;
        if tmp^.Info.x = oldp.x then
          tmp^.Info.x := newp.x;
        tmp := tmp^.Adr;
      end;
    end;

  end;

  // Изменение координат всех точек линии на одинаковое количество
  // пикселей
  procedure moveAllLinePoint(head: PPointsList; dx, dy: integer);
  var
    tmp: PPointsList;
  begin
    tmp := head^.Adr;
    while tmp <> nil do
      begin
        tmp^.Info.x := tmp^.Info.x - dx;
        tmp^.Info.y := tmp^.Info.y - dy;
        tmp := tmp^.Adr;
      end;
    end;
  end;

  // Функция ищет линию вблизи точки и возвращает в x,y точку на прямой
  // вблизи исходной точки. Если фигура не нашлась, функция вернет nil
  function searchNearLine(head: PFigList; var x,y: integer):PPointsList;
  var
    temp: PFigList;
    tmpP: PPointsList;
    lastP: PPointsList;
    maxX, maxY, minX, minY:integer;
  begin
    Result := nil;
    temp:= head.adr;

    while temp <> nil do
      begin

```

```

if temp^.Info.tp = line then
begin
  if temp^.Info.PointHead = nil then
  begin
    temp := temp^.adr;
    Continue;
  end;

  tmpP:= temp^.Info.PointHead^.adr;
  lastP:=tmpP;
  if tmpP^.Adr <> nil then
    tmpP := tmpP^.adr;
  while tmpP <> nil do
  begin
    // Перебирает точки фигуры
    maxY := max(tmpP.Info.y, lastP.Info.y);
    minY := min(tmpP.Info.y, lastP.Info.y);
    maxX := max(tmpP.Info.x, lastP.Info.x);
    minX := min(tmpP.Info.x, lastP.Info.x);
    if (abs(MaxX - x) < NearFigure)
      and
      (y > minY)
      and
      (y < maxY) then
    begin
      x := MaxX;
      Result := tmpP;
      exit;
    end;

    if (abs(MinX - x) < NearFigure)
      and
      (y > minY)
      and
      (y < maxY) then
    begin
      x := MinX;
      Result := tmpP;
      exit;
    end;

    if (abs(MaxY - y) < nearFigure)
      and
      (x > minx)
      and
      (x < maxx) then
    begin
      y := MaxY;
      Result := tmpP;
      exit;
    end;

    if (abs(MinY - Y) < NearFigure)
      and
      (X > minX)
      and
      (X < maxX) then
    begin
      Y := MinY;

```

```

    Result := tmpP;
    exit;
end;
lastp:= tmpP;
tmpP := tmpP^.adr;
end;

end;
temp := temp^.Adr;
end;
end;

// BOOOLEAN FUNCTIONS

// Возвращает true если линия горизонтальная
function isHorLine(curr, prev: PPointsList):boolean;
begin
    if prev = nil then
        Result := (curr^.Adr <> nil) and (curr^.Info.y = curr^.adr^.Info.y)
    else
        Result := prev^.Info.y = curr^.Info.y;
    end;
end;

// Возвращает true если нужна стрелка по середине
function needMiddleArrow(tmp: PPointsList; FirstP: TPointsInfo) :Boolean;
begin
    Result := (tmp^.Adr <> nil) and (tmp^.adr^.Adr = nil) and (tmp^.Info.x <> FirstP.x)
        and (tmp^.Info.x = tmp^.adr^.Info.x) and (abs(tmp^.Info.y - tmp^.adr^.Info.y) > Tolerance*2)
end;

// Возвращает true если нужен диагональный срез
function isHorisontalIntersection(head: PFigList; blocked: PPointsList):boolean;
var
    tmp: PFigList;
    tmpP: PPointsList;
    ti1, ti2: TPointsInfo;
begin
    Result := false;
    tmp:= head^.adr;
    while tmp <> nil do
        begin
            if tmp^.Info.tp = Line then
                begin
                    if tmp^.info.PointHead = nil then
                        begin
                            tmp := tmp^.adr;
                            continue;
                        end;

                    tmpP := tmp^.Info.PointHead^.adr;
                    while (tmpP <> nil) and (tmpP^.adr <> nil) do
                        begin

                            ti1 := tmpP^.Info;
                            ti2 := tmpP^.adr.Info;

```

```

    if (abs(ti1.x- blocked.Info.x) < NearFigure)
    and (abs(ti2.x - blocked.Info.x) < NearFigure)
    and (blocked.Info.y < max(ti1.y, ti2.y))
    and (blocked.Info.y > min(ti1.y, ti2.y))
    and (tmpP <> blocked) and (tmpP^.adr <> blocked)
    then
    begin
        Result := true;
        exit;
    end;
    tmpP:= tmpP^.adr;
end;
end;
tmp := tmp^.Adr;
end;
end;

```

```

function isBelongsLine(head: PPointsList; x,y: integer): Boolean;
var
    tmp, tmp2 : PPointsList;
begin
    if (head = nil) or (head^.Adr = nil) or (head^.Adr^.Adr = nil) then exit;
    tmp := head^.adr;
    tmp2 := tmp^.Adr;
    Result := false;
    while (tmp <> nil) and (tmp2 <> nil) do
    begin
        if (tmp^.Info.x = tmp2^.Info.x) and (abs(tmp^.Info.x - x) < Tolerance) then
        begin
            if (y > min(tmp^.Info.y, tmp2^.Info.y)) and (y < max(tmp^.Info.y, tmp2^.Info.y)) then
            begin
                Result := true;
                exit;
            end;
        end;
        if (tmp^.Info.y = tmp2^.Info.y) and (abs(tmp^.Info.y - y) < Tolerance) then
        begin
            if (x > min(tmp^.Info.x, tmp2^.Info.x)) and (x < max(tmp^.Info.x, tmp2^.Info.x)) then
            begin
                Result := true;
                exit;
            end;
        end;
        tmp := tmp^.Adr;
        tmp2 := tmp2^.adr;
    end;
end;
end.

```

Модуль Model.UndoStack

```

unit Model.UndoStack;

```

```

interface

```

```

uses Data.Types; // in SD_Types - declaration of Stack type
procedure CreateStack(var adr: PUndoStack);

```

```

procedure UndoStackPush(var Vertex: PUndoStack; info: TUndoStackInfo);
function undoStackPop(var Vertex: PUndoStack; var rec: TUndoStackInfo):boolean;
function isEmpty(Vertex: PUndoStack): Boolean;
procedure UndoStackClear(var vertex: PUndoStack);

implementation
uses vcl.dialogs;

// Создание стека
procedure CreateStack(var adr: PUndoStack);
begin
  new(adr);
  adr^.Prev := nil;
  adr^.Inf.ChangeType := NonDeleted; // Всегда самый последний элемент, нельзя удалять
end;

// Добавление элемента в стек и перемещение вершины на новый элемент
procedure UndoStackPush(var Vertex: PUndoStack; info: TUndoStackInfo);
var
  tmp: PUndoStack;
begin
  if Info.ChangeType = NonDeleted then
    begin
      ShowMessage('Error'); // NonDelete - только самый последний элемент стека
    end
  else
    begin
      new(tmp);
      tmp^.Prev := vertex;
      vertex := tmp; // Перемещение вершины
      tmp^.Inf := info;
    end;
  end;
end;

// Если стек пуст, возвращает false, иначе
// Извлекает из стека одного элемент, возвращает запись в переменную rec
// И перемещение вершины стека на предыдущий элемент
function undoStackPop(var Vertex: PUndoStack; var rec: TUndoStackInfo):boolean;
var
  tmp: PUndoStack;
begin
  Result := true;
  if (vertex^.Inf.ChangeType <> NonDeleted) then // Если стек не пуст
    begin
      tmp := vertex;
      rec := tmp^.Inf;
      Vertex := tmp^.Prev; // Перемещение вершины
      Dispose(tmp);
    end
  else
    Result := false;
  end;
end;

// Возвращает true если стек пуст
function isEmpty(Vertex: PUndoStack): Boolean;
begin
  Result := Vertex^.Inf.ChangeType = NonDeleted;
end;

```

```

// Очистка стека
procedure UndoStackClear(var vertex: PUndoStack);
var
  tmp: PUndoStack;
begin
  while vertex.Inf.ChangeType <> NonDeleted do
    begin
      tmp := vertex;
      vertex := vertex.Prev;
      Dispose(tmp);
    end;
  end;
end.

```

Модуль View.Canvas

```

unit View.Canvas;
// VIEW par in MVC:
// responsible for displaying information

interface
uses Data.Types, vcl.graphics, Data.InitData, vcl.dialogs;

// ### VIEW PART PROCEDURES ###

// A search of the list of figures and their drawing
procedure drawFigure(Canvas:TCanvas; head:PFigList; scale: real; isVertex:boolean = true);
procedure drawSelectFigure(canvas:tcanvas; figure: TFigureInfo);
procedure drawSelectLineVertex(canvas: TCanvas; Point: TPointsInfo);
function beginOfVertLine(tmp:PPointsList;firstP: TPointsInfo):boolean;
procedure selectFigure(canvas: TCanvas; head:PFigList);

implementation
uses main, Model, Model.Lines;

// MoveTo with using scale
procedure ScaleMoveTo(canvas:TCanvas; x,y: integer);
var scale : real;
begin
  scale := EditorForm.FScale;
  canvas.MoveTo( ScaleRound(scale,x), ScaleRound(scale, y) );
end;
// LineTo with using scale
procedure ScaleLineTo(canvas:TCanvas; x,y: integer);
var scale : real;
begin
  scale := EditorForm.FScale;
  canvas.LineTo( ScaleRound(scale,x), ScaleRound(scale, y) );
end;

procedure drawArrowVertical(Canvas:TCanvas; x,y : integer; coef: ShortInt);
begin
  // Draw Vertical Arrow
  ScaleMoveTo(Canvas,x,y);
  ScaleLineTo(Canvas,x-Arrow_Height,y+Arrow_Width*coef);
  ScaleMoveTo(Canvas,x,y);
  ScaleLineTo(Canvas,x+Arrow_Height,y+Arrow_Width*coef);

```



```

    ScaleMoveTo(Canvas,x,y);
end;

// Draw horisontal arrow
procedure drawArrow(Canvas:TCanvas; x,y : integer; coef: ShortInt);
begin
    ScaleMoveTo(Canvas,x,y);
    ScaleLineTo(Canvas,x-Arrow_Width*coef,y-Arrow_Height);
    ScaleMoveTo(Canvas,x,y);
    ScaleLineTo(Canvas,x-Arrow_Width*coef,y+Arrow_Height);
    ScaleMoveTo(Canvas,x,y);
end;

procedure selectFigure(canvas: TCanvas; head:PFigList);
var
    tmp : PPointsList;
begin
    //ShowMessage('kek');
    if head^.Info.tp <> line then
    begin
        // Рисуем вершины
        drawSelectFigure(canvas, head^.Info);
    end
    else
    begin
        //showmessage('kek');
        if head^.Info.PointHead = nil then exit;
        tmp := head^.Info.PointHead^.adr;
        while tmp <> nil do
            begin
                drawSelectLineVertex(canvas,tmp^.info);
                tmp := tmp^.Adr;
            end;
        end;
    end;
end;

// Draw out bound line ( \----- )
procedure drawOutBoundLine(canvas: TCanvas; FirstP: TPointsInfo; tmp:PPointsList);
var
    coef: -1..1;
begin
    if FirstP.y - tmp^.adr^.Info.y < 0 then
        coef := 1
    else
        coef := -1;
    ScaleMoveTo(Canvas,tmp^.Info.x- Lines_DegLenght, tmp^.Info.y);
    ScaleLineTo(Canvas,tmp^.Info.x, tmp^.Info.y+coef*Lines_Deg);
    ScaleMoveTo(Canvas,tmp^.Info.x, tmp^.Info.y+coef*Lines_Deg);
    // canvas.Rectangle(tmp^.Info.x-VertRad,tmp^.Info.y+15*coef-VertRad, tmp^.Info.x+VertRad, tmp^.Info.y+15*coef+VertRad);
end;

// Return true if this is begin of vertical line
function beginOfVertLine(tmp:PPointsList;firstP: TPointsInfo):boolean;
begin
    result := (tmp^.Adr <> nil) and
        (FirstP.x = tmp^.adr^.Info.x) and // If vertical line
        (FirstP.y <> tmp^.adr^.Info.y);
end;

// Draw rectangles at vertex of figures

```

```

procedure drawVertexRect(canvas:TCanvas; point: TPointsInfo; color:TColor = clBlack);
var
  ScaleVertRad: integer;
begin
  canvas.Pen.Color := color;
  if color <> clBlack then
    Canvas.Brush.Color := color;

  canvas.Pen.Width := 1;
  EditorForm.useScale(Point.x, point.y);
  ScaleVertRad := ScaleRound(EditorForm.FScale, VertRad);
  canvas.Rectangle(point.x-ScaleVertRad,point.y-ScaleVertRad, point.x+ScaleVertRad, point.y+ScaleVertRad);
  canvas.Pen.Width := Round(Lines_Width*EditorForm.FScale);
  canvas.Pen.Color := clBlack;
  canvas.Brush.Color := clwhite;
end;

// Draw Incoming line ( ----| )
procedure drawIncomingLine(canvas: tcanvas; point: TPointsInfo; coef: ShortInt);
begin
  ScaleLineTo(Canvas,point.x, point.y + (Lines_Deg*coef));
  ScaleMoveTo(Canvas,point.x, point.y + (Lines_Deg*coef));
  ScaleLineTo(Canvas,point.x+Lines_DegLenght, point.y);
  drawArrowVertical(canvas, point.x, point.y+Lines_DegLenght*coef, coef);
end;

// Draw arrow at end of line
procedure drawArrowAtEnd(canvas:TCanvas; point, PrevPoint:TPointsInfo);
var
  tmpx, tmpy:integer;
begin
  tmpx := point.x - PrevPoint.x;

  if tmpx > 0 then
    drawArrow(canvas,point.x, point.y,1)
  else if tmpx < 0 then
    drawArrow(canvas,point.x, point.y,-1);

  tmpy := point.y - PrevPoint.y;
  if tmpy > 0 then
    drawArrowVertical(canvas,point.x, point.y,-1)
  else if tmpy < 0 then
    drawArrowVertical(canvas,point.x, point.y,1);
end;

// Draw lines
procedure drawLines(Canvas:TCanvas; head: PPointsList; LT: TLineType; isVertex: boolean; scale: Real);
var
  tmp: PPointsList; // Temp variable
  FirstP,PrevP: TPointsInfo; // First and Prev Point in list
  tmpx: integer;
  isFirstLine:boolean;
  point1:TPointsInfo;
  isDegEnd: boolean;
  coef: -1..1;
begin
  coef := 1;
  canvas.Pen.Width := Trunc(Lines_Width*scale); // Width For Line
  isFirstLine := false;

```

```

tmp := head;

if (tmp <> nil) and (tmp^.Adr <> nil) then
begin
  FirstP.X := tmp^.Adr^.Info.x; // Initialise First Points
  FirstP.y := tmp^.Adr^.Info.y;

  prevp.x := FirstP.X; // Initialise Preview Point
  prevp.y := FirstP.Y;

  tmp := tmp^.Adr;

  ScaleMoveTo(Canvas,tmp^.Info.x, tmp^.Info.y); // Move to first point in list

  // FIRST POINT:
  if beginOfVertLine(tmp,firstP) and (PrevP.y = tmp^.Info.y) then
  begin
    drawOutBoundLine(canvas,FirstP, tmp);
    isFirstLine := true;
  end;

  if (tmp^.Adr <> nil) and (PrevP.y = tmp^.adr^.Info.y) and isHorisontalIntersection(EditorForm.getFigureHead,tmp) then
  begin
    if (tmp^.Adr <> nil) and (FirstP.x - tmp^.adr^.Info.x < 0) then
      coef := 1
    else
      coef := -1;
    ScaleMoveTo(Canvas,tmp^.Info.x, tmp^.Info.y-Lines_DegLenght);
    ScaleLineTo(Canvas,tmp^.Info.x+Lines_Deg*coef, tmp^.Info.y);
  end;

  if isVertex then
    drawVertexRect(canvas, tmp^.Info);

  // OTHER POINTS:
  tmp := tmp^.adr;
  isDegEnd := false;
  while tmp <> nil do
  begin
    {if LT=LAdditLine then
    begin
      tmp^.Info.y := AddY;
    end;}
    if (PrevP.y = tmp^.Info.y) and (tmp^.Adr = nil) and isHorisontalIntersection(EditorForm.getFigureHead,tmp) then
    begin
      ScaleLineTo(Canvas,tmp^.Info.x-Lines_Deg*coef, tmp^.Info.y);
      // Перед \ - стрелочка
      if (PrevP.x - tmp^.Info.x > 0) and (PrevP.y = tmp^.Info.y) then
        drawArrow(canvas, tmp^.Info.x-Lines_Deg*coef, tmp^.Info.y, -1)
      else if (PrevP.y = tmp^.Info.y) then
        drawArrow(canvas, tmp^.Info.x-Lines_Deg*coef, tmp^.Info.y, 1);
      ScaleMoveTo(Canvas,tmp^.Info.x, tmp^.Info.y+Lines_DegLenght);
      ScaleLineTo(Canvas,tmp^.Info.x-Lines_Deg*coef, tmp^.Info.y);
      Point1 := tmp^.Info;
      if isVertex then
        drawVertexRect(canvas, point1);

      tmp:=tmp^.Adr;
      continue;
    end;
  end;

```

```

if isDegEnd then
begin
  drawIncomingLine(canvas, tmp^.Info, coef);
  if isVertex then
    drawVertexRect(canvas, tmp^.Info);
  tmp := tmp^.Adr;
  continue;
end
else
  ScaleLineTo(Canvas,tmp^.Info.x, tmp^.Info.y);

ScaleMoveTo(Canvas,tmp^.Info.x, tmp^.Info.y);
if isVertex then
  drawVertexRect(canvas, tmp^.Info);

// Рисуем стрелочку в конце линии
if (tmp^.Adr = nil) then
begin
  drawArrowAtEnd(canvas, tmp^.Info, prevP);
end;

if needMiddleArrow(tmp, FirstP) then // if these is incoming and outgoing lines
begin
  if isFirstLine then
  begin
    tmpx := tmp^.Info.x - PrevP.x;
    if tmpx > 0 then
      tmpx := 1
    else
      tmpx := -1;
    drawArrow(Canvas,tmp^.Info.x + 10 - (tmp^.Info.x - PrevP.x) div 2, tmp^.Info.y, tmpx);
    ScaleMoveTo(Canvas,tmp^.Info.x , tmp^.Info.y)
  end;
  if tmp^.Info.y - tmp^.adr^.Info.y < 0 then
    coef := -1
  else
    coef := 1;
  isDegEnd := true;
end
else
begin
  isDegEnd:= false;
end;
prevp.x := tmp^.Info.x;
prevp.y := tmp^.Info.y;
tmp:= tmp^.Adr;
end;

end;
canvas.Pen.Width := 1;
end;

procedure drawFigure(Canvas:TCanvas; head:PFigList; scale: real; isVertex:boolean = true);
var
  temp:PFigList;

```

```

TextW: Integer;
TextH: Integer;
TX, TY: integer;
Point: TPointsInfo;
text:string;
begin
temp := head^.adr;
while temp <> nil do
begin
with temp^.Info do
begin
if tp <> Line then
text := String(txt);
case Tp of
Def: Text := '< ' + Text + ' > ::=';
MetaVar: Text := '< ' + Text + ' >';
MetaConst: ;
line:
begin
drawLines(Canvas, temp^.Info.PointHead, temp^.Info.LT, isVertex, scale);
temp := temp^.adr;
continue; // if figure - line => draw this line and skip iteration
end;
else
;
end;
Canvas.Font.Size := Font_Size;
TextW := canvas.TextWidth(text);
textH := Canvas.TextHeight(text);
// Расчитываем координаты, чтобы текст был по середине
TX := x1 + (x2 - x1) div 2 - TextW div 2;
TY := y1 + (y2 - y1) div 2 - TextH div 2 - 3;
Canvas.Font.Size := ScaleRound(Scale, Font_Size);
// Если ширина или высота блока меньше, чем текста, то подгоняем под размер текста
if (abs(x2 - x1) < TextW) then
begin
x1 := x1 - textw div 2 - 10;
x2 := x2 + textw div 2 + 10;
end;
if (abs(y2 - y1) < TextH) then
begin
y1 := y1 - textH div 2 - 10;
y2 := y2 + textH div 2 + 10;
end;

if isVertex then
begin
// Рисуем вершины
Point.x := x1;
Point.y := y1;

drawVertexRect(canvas, Point);

Point.y := y2;
drawVertexRect(canvas, Point);

Point.x := x2;
drawVertexRect(canvas, Point);

```

```

    Point.y := y1;
    drawVertexRect(canvas, Point);
end;
Canvas.Font.Size := ScaleRound(Scale, Font_Size);
Canvas.TextOut(ScaleRound(Scale, TX),ScaleRound(Scale, TY), text);
end;

temp := temp^.Adr;
end;
end;

procedure drawSelectFigure(canvas:tcanvas; figure: TFigureInfo);
var x1,x2,y1,y2: integer;
point: TPointsInfo;
begin
    x1 := figure.x1;
    x2 := figure.x2;
    y1 := figure.y1;
    y2 := figure.y2;

    Point.x := x1;
    Point.y := y1;
    drawVertexRect(canvas, Point, clGreen);

    Point.y := y2;
    drawVertexRect(canvas, Point, clGreen);

    Point.x := x2;
    drawVertexRect(canvas, Point, clGreen);

    Point.y := y1;
    drawVertexRect(canvas, Point, clGreen);

end;

procedure drawSelectLineVertex(canvas: TCanvas; Point: TPointsInfo);
begin
    drawVertexRect(canvas,Point, clGreen);
end;

end.

```

Модуль View.SVG

```

unit View.SVG;

interface
uses Data.Types, Data.InitData;
procedure exportToSVG(head: PFigList; w,h: Integer; path:UTF8String; title: UTF8String; desc: UTF8String);

implementation
uses SysUtils, vcl.dialogs, vcl.graphics, Model, View.Canvas, main, Model.Lines;

// Заголовок SVG
const svg_head = '<?xml version="1.0" standalone="no"?>' + #10#13
    + '<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"' + #10#13
    + '"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">';

// Возвращает строку с открытием SVG-тега

```

```

function getSVGOpenTag(h,w: integer):UTF8String;
begin
  result := '<svg width="' + IntToStr(W) + '" height="' + IntToStr(H) + '#10#13'
    + '" viewBox="0 0 ' + IntToStr(W) + ' ' + IntToStr(h) + '" + '#10#13
    + ' xmlns="http://www.w3.org/2000/svg" version="1.1">'
end;

function writePatch(Point1, Point2: TPointsInfo; color: UTF8String = 'black'; width: Integer = Default_LineSVG_Width):UTF8String;
begin
  Result := '<path d="M ' + IntToStr(Point1.x) + ' '
    + IntToStr(Point1.y) + ' ' + 'L ' + IntToStr(Point2.x)
    + ' ' + IntToStr(Point2.y) + ' '
    + 'fill="none" stroke="' + color + '" stroke-width="'
    + IntToStr(width) + '" />'
end;

// Функция возвращает строку с text-тегом с заданным содержанием
function writeSVGText(Figure: TFigureInfo; text: UTF8String; family: UTF8String =
  'Tahoma'; size: integer = 16):UTF8String;
var TH, TW, TX, TY: real;
    TextW, TextH: integer;
const
  SVG_VeriticalEpsilon = 4; // Погрешность (Расстояние от верхней точки буквы до
    // верха выделяемой области больше, чем до нижней)
begin
  with figure do
    begin
      // Координаты центра прямоугольника
      TX := x1 + abs(x2 - x1) / 2;
      TY := y1 + abs(y2 - y1) / 2 + SVG_VeriticalEpsilon;
    end;

    {'<rect x="' + IntToStr(figure.x1) + '" y="' + IntToStr(figure.y1) + '"
    + ' width="' + IntToStr(figure.x2-figure.x1) + '" height="' + IntToStr(figure.y2-figure.y1) + '"
    + ' style="fill:blue;stroke:pink;stroke-width:0.1;fill-opacity:0.1;stroke-opacity:0.9" />'}

    Result:=
    // text-anchor="middle" - центрирует текст по вертикали и горизонтали относительно
    // определенной точки. Подробнее - в документации по формату SVG
    '<text text-anchor="middle" font-family = "' + family + '" font-size = "' + IntToStr(size) + '"
    + ' x="' + StringReplace(FormatFloat( '#.####', TX), ',', '.', [rfReplaceAll])
    + '" y="' + StringReplace(FormatFloat( '#.####', TY), ',', '.', [rfReplaceAll])
    + '">' + text + '</text>';
  end;

procedure drawSVGBoundLine(var f: TextFile; FirstP: TPointsInfo; tmp: PPointsList; var lastp: PPointsList);
var
  coef: -1..1;
  P1, P2: TPointsInfo;
begin
  if FirstP.y - tmp^.adr^.Info.y < 0 then
    coef := 1
  else
    coef := -1;
  p1.x := tmp^.Info.x-Lines_DegLenght;
  p1.y := tmp^.Info.y;
  p2.x := tmp^.Info.x;
  p2.y := tmp^.Info.y+coef*Lines_Deg;

```

```

writeln(f, '<!-- BOUND LINE -->');
writeln(f, writePatch(p1, p2));
lastp^.Info := p2;
end;

procedure drawSVGArrowVertical(var f: textFile; x,y : integer; coef: ShortInt);
var
  p1, p2: TPointsInfo;
begin
  // Draw Vertical Arrow
  P1.x := x;
  P1.y := y;
  P2.x := x-Arrow_Height;
  P2.y:= y+Arrow_Width*coef;
  writeln(f, '<!-- ARROS LEFT -->');
  writeln(f, writePatch(p1,p2));

  p2.x := x+Arrow_Height;
  p2.y := y+Arrow_Width*coef;
  writeln(f, '<!-- ARROW RIGHT -->');
  writeln(f, writePatch(p1,p2));
end;

procedure drawSVGArrow(var F: TextFile; x,y : integer; coef: ShortInt);
var
  p1, p2: TPointsInfo;
begin
  P1.x := x;
  P1.y := y;
  P2.x := x-Arrow_Width*coef;
  P2.y:= y-Arrow_Height;
  writeln(f, '<!-- ARROS LEFT -->');
  writeln(f, writePatch(p1,p2));

  p2.x := x-Arrow_Width*coef;
  p2.y := y+Arrow_Height;
  writeln(f, '<!-- ARROW RIGHT -->');
  writeln(f, writePatch(p1,p2));
end;

// Превращаем специальные символы в "сущности"
function htmlspecialchars(s: UTF8String):UTF8String;
begin
  s:=StringReplace(s,'&','&amp;',[rfReplaceAll, rfIgnoreCase]);
  s:=StringReplace(s,'<','&lt;',[rfReplaceAll, rfIgnoreCase]);
  s:=StringReplace(s,'>','&gt;',[rfReplaceAll, rfIgnoreCase]);
  s:=StringReplace(s,'"','&quot;',[rfReplaceAll, rfIgnoreCase]);
  result:=s;
end;

procedure drawArrowAtSVG(var f: textfile; point, PrevPoint:TPointsInfo);
var
  tmpx, tmpy:integer;
begin
  tmpx := point.x - PrevPoint.x;
  if tmpx > 0 then
    drawSVGArrow(f,point.x, point.y,1)
  else if tmpx < 0 then
    drawSVGArrow(f,point.x, point.y,-1);

```



```

tmpy := point.y - PrevPoint.y;
if tmpy > 0 then
  drawSVGArrowVertical(f,point.x, point.y,-1)
else if tmpy < 0 then
  drawSVGArrowVertical(f,point.x, point.y,1);
end;

procedure drawIncomingLineSVG(var f: textfile; point: TPointsInfo; coef: ShortInt; var d: PPointsList);
var
  p1,p2: TPointsInfo;
begin
  point.y := point.y - coef*Lines_DegLenght;
  p1 := point;
  p2:= point;
  p2.y := p2.y + (Lines_Deg*coef);

  p1 := p2;

  p2.x := point.x+Lines_DegLenght;
  p2.y := point.y;
  writeln(f, '<!-- Incoming Line -->');
  writeln(f, writePatch(p1,p2));
  drawSVGArrowVertical(f, point.x, point.y+Lines_DegLenght*coef, coef);
  {point.y := point.y + 2*coef*Lines_DegLenght;}
end;

// Экспорт в SVG
procedure exportToSVG(head: PFigList; w,h: Integer; path:UTF8String; title: UTF8String; desc: UTF8String);
var
  f: TextFile;
  Point1, Point2: TPointsInfo;
  tmp: PFigList;
  tmpx: integer;
  tmpP, prevP: PPointsList;
  firstP: TPointsInfo;
  isFirstLine,isDegEnd :Boolean;
  coef: ShortInt;
  text: UTF8String;
  prev: TPointsInfo;
  curr:TPointsInfo;
  isChanged: boolean;
begin
  AssignFile(f, path,CP_UTF8);
  rewrite(f);
  writeln(f, svg_head);
  writeln(f, getSVGOpenTag(h,w));
  writeln(f, '<title>' + title + '</title>');

  tmp := head^.adr;
  while tmp <> nil do
  begin
    if tmp^.Info.tp = line then
    begin
      tmpP := tmp^.Info.PointHead^.adr;
      prevp := tmpP;

```

```

curr := tmpP^.Info;
firstP := tmpP^.Info;
isFirstLine := false;
if beginOfVertLine(tmpP,firstP) then
begin
  prev := prevP^.Info;
  drawSVGBoundLine(f, firstP, tmpP, prevp);
  writeln(f, '<!-- Line After Bound -->');
  Writeln(f, writePatch(prevP^.Info, tmpP^.adr.Info));
  prevP^.Info := prev;
  tmpP := tmpP^.Adr;
  isFirstLine := true;
end;
isChanged := false;
if isHorisontalIntersection(EditorForm.getFigureHead,tmpP) then
begin
  if (tmpP^.Adr <> nil) and (FirstP.x - tmpP^.adr^.Info.x < 0) then
    coef := 1
  else
    coef := -1;
  Point1 := tmpP^.Info;
  point1.y := tmpP^.Info.y-Lines_DegLenght;
  point2 := tmpP^.Info;
  point2.x := tmpP^.Info.x+Lines_Deg*coef;
  writeln(f, '<!-- Additional line Intersection: -->');
  writeln(f, writePatch(Point1,Point2,'black'));
  curr := tmpP^.Info;

  curr.x := tmpP^.Info.x+Lines_Deg*coef;
  isChanged := true;

end;

while (tmpP <> nil) and (tmpP^.Adr <> nil) do
begin
  if not isChanged then
    prev := tmpP^.Info
  else
    prev := curr;
  prevp := tmpP;
  tmpP := tmpP^.Adr;
  curr := tmpP^.Info;
  if (tmpP^.Adr = nil) and isHorisontalIntersection(EditorForm.getFigureHead,tmpP) then
  begin
    point1 := prevP.Info;
    Point2.x := curr.x-Lines_Deg*coef;
    point2.y := curr.y;
    writeln(f, '<!-- Additional line: -->');
    writeln(f, writePatch(Point1,Point2,'black'));
    if Prev.x - curr.x > 0 then
      drawSVGArrow(f, curr.x-Lines_Deg*coef, curr.y, -1)
    else
      drawSVGArrow(f, curr.x-Lines_Deg*coef, curr.y, 1);
    point1.x := curr.x;
    point1.y := curr.y+Lines_DegLenght;
    point2.x := curr.x-Lines_Deg*coef;
    point2.y := curr.y;
    writeln(f, '<!-- Additional line Intersect 2: -->');
    writeln(f, writePatch(Point1,Point2,'black'));
    curr := point1;
  end;
end;

```

```

prevP := tmpP;
prev := tmpP^.Info;
tmpP:=tmpp^.Adr;
continue;
end;

if (tmpP^.Adr = nil) and (prevP.Info.x = curr.x) then
begin
  writeln(f, '<!-- LAST VERTICAL: -->');
  curr.y := curr.y + 15*coef;
end;
writeln(f, '<!-- LINE: -->');
Writeln(f, writePatch( prev, curr));
if (tmpP^.Adr = nil) and isDegEnd and (prevP^.Info.x = curr.x) then
begin

  drawIncomingLineSVG(f, curr, coef, tmpp);
  prevP := tmpP;
  prev := tmpP^.Info;
  continue;
end;

if (tmpP^.Adr = nil) then
begin
  drawArrowAtSVG(f, curr, prev);
end;

if needMiddleArrow(tmpp, FirstP) then // if these is incoming and outgoing lines
begin
  if isFirstLine then
  begin
    tmpx := curr.x - Prev.x;
    if tmpx > 0 then
      tmpx := 1
    else
      tmpx := -1;
    drawSVGArrow(f,curr.x + Arrow_Height - (curr.x - PrevP^.info.x) div 2, curr.y, tmpx);

  end;
  if curr.y - tmpP^.adr^.Info.y < 0 then
    coef := -1
  else
    coef := 1;
  isDegEnd := true;
end
else
begin
  isDegEnd:= false;
end;
end;
end
else
begin // Other Figures
text:= AnsiToUtf8( tmp^.Info.Txt );
case tmp^.Info.tp of
  Def: Text := '<' + Text + '> '::= '';
  MetaVar: Text := '<' + Text + '>';
  MetaConst: ;
end;
text := htmlspecialchars(text);

```

```

    Writeln(f, writeSVGText(tmp^.Info, text));
end;
tmp := tmp^.Adr;
end;

```

```

writeln(f, '</svg>'); // Зарытие тега SVG
close(f);
end;

end.

```

Модуль Data.Types

```

unit Data.Types;

interface
type
    TDrawMode = (Draw, NoDraw, DrawLine, ResizeCanvas); // Режим рисования
    TFileMode = (FSvg, FBrakh, FBmp, FPng); // Режим открытия/сохранения файла
    TLineType = (LLine);
    TEditMode = (NoEdit, Move, TSide, BSide, RSide, LSide, Vert1, Vert2, Vert3, Vert4, LineMove);
    // Режим редактирования
    TType = (Def, MetaVar, MetaConst, Line, None);
    // Режим фигуры

    // СПИСОК ТОЧЕК НАЧАЛО
    TPointsInfo = record
        x,y: integer;
    end;
    PPointsList = ^TPointsList;
    TPointsList = record
        Info: TPointsInfo;
        Adr: PPointsList;
    end;
    // СПИСОК ТОЧЕК КОНЕЦ

    // СПИСОК ФИГУР НАЧАЛО
    TFigureInfo = record
        case tp: TType of
            Def, MetaConst, MetaVar: (x1,x2,y1,y2: integer; Txt: string[255]);
            Line: (PointHead: PPointsList; LT: TLineType);
            None: (Check: string[5]; Width, Height: Integer);
        end;
    PFigList = ^FigList;
    FigList = record
        Info: TFigureInfo;
        Adr: PFigList;
    end;
    // СПИСОК ФИГУР КОНЕЦ

    // ПРЕДСТАВЛЕНИЕ ФИГУР В ФАЙЛЕ
    TFigureInFile = record
        case tp: TType of
            Def, MetaConst, MetaVar: (Txt: string[255]; x1,x2,y1,y2: integer);
            Line: (Point: String[255]; LT: TLineType);
            None: (Check: string[5]; Width, Height: Integer);
        end;
    end;

```

```

end;

// UNDO STACK
TChangeType = (chDelete, chInsert, chAddPoint, chFigMove, chPointMove, chChangeText, chCanvasSize, NonDeleted);
TUndoStackInfo = record
    adr: PFigList;
Case ChangeType : TChangeType of
    chDelete: (PrevFigure: PFigList); // Удаление фигуры
    chAddPoint: (PrevPointAdr: PPointsList); // Добавление точки в линии
    chInsert: (); // Добавление фигуры
    chFigMove: (PrevInfo: TFigureInfo); // Перемещение/изменение размеров фигур. PrevInfo - координаты "бэкапа"
    chPointMove: (st: string[255]);
    chChangeText: (text: string[255]);
    chCanvasSize: (w,h: Cardinal);
    NonDeleted: (); // Используется для обозначения последней записи стека, которую нельзя рор
end;

PUndoStack = ^TUndoStack;
TUndoStack = record
    Inf: TUndoStackInfo;
    Prev: PUndoStack;
end;

implementation

end.

```

Модуль Data.InitData

```

unit Data.InitData;

interface
const
    Tolerance = 5; // Кол-во пикселей, на которые юзеру можно "промахнуться"
    NearFigure = 20; // Количество пикселей, при котором идет "присоединение" фигуры;
    step_round = 20; // "Шаг сетки"
    Default_LineSVG_Width = 2; // Ширина линии в SVG
    Font_Size = 8; // Размер шрифта

resourcestring // Работа с файлами:
    rsNewFileDialog = 'Вы уверены? Все несохраненные данные будут удалены. Продолжить?';
    rsNewFile = 'Новый файл';
    rsExitDlg = 'Вы внесли изменения.. А не хотите ли Вы сохраниться перед тем, как выйти?';
    rsInvalidFile = 'Вы пытаетесь открыть какой-то непонятный файл. Пожалуйста, используйте только файлы, созданные этой программой!';
    rsTrashFile = 'Файл поврежден!';
resourcestring // Раздел помощи
    rsHelpHowIsSD_Caption = 'Что такое синтаксическая диаграмма?';
    rsHelp_Caption = 'Помощь';
const // Название ресурсов
    rsHelpHowIsSD_ResName = 'help1';
    rsHelp_ResName = 'help2';

    // ### VIEW PART CONSTANTS ###
const
    VertRad = 3; // Радиус вершины

    Arrow_Width = 20; // Ширина "дюбки" стрелки

```

```
Arrow_Height = 10; // Высота стрелки

Lines_Width = 2; // ширина линии стрелки
Lines_Deg = 15; // Высота диагонального "среза"
Lines_DegLenght = 15; // Ширина диагонального "среза"
implementation
end.
```