Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных технологий

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**2D И 3D графический редактор**

БГУИР КП 1-40 01 01 00 027 ПЗ

Студент Е.О.Селедцова

Руководитель В.А. Воробей

Прием \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

    Дата Отметка Подпись

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных технологий

УТВЕРЖДАЮ

Заведующий кафедрой

*––––––––––––––––––––––––*

(подпись)

––––––––––––––––––––––2020г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    *Селедцовой Елизавете Олеговне*––––––––––––––––––––––––––––––––

1. Тема проекта   2d и 3d графический редактор–––––––––––––––––––––––      –––––––––      –––––––––      –––––––––      –––––––––      –––––––––      –––––––––––––––                  ––   –––––––––––––––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи студентом законченного проекта–––––*01 июля 2020 г.–––– ––*

3. Исходные данные к проекту *Инструменты разработки: операционная система Windows 10, интегрированная среда разработки QT Creator язык программирования С++.–––––––––––––––––––––           –––––––––––––––––––––           —–––––––\_\_\_\_*

*––––––––––––––––––––––––––––––––––––––––––––––––––––––           —–––       \_\_*

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

*Введение*

*1 Постановка задачи 1.1 Описание и анализ предметной области 1.2 Обзор существующих аналогов 1.3 Информационная база задачи 2 Проектирование задачи 2.1 Интерфейс программного средства 2.2. Выбор и обоснование инструментов разработк 3 Программная реализация 3.1 Физическая структура 3.2 Описание разработанных модулей 4 Тестирование 5 Применение 5.1 Назначение и условия применения 5.2 Руководство пользователя Заключение Приложение А Приложение Б* –––––––––      –––––––––      –––––––––      –––––––––      –––––––––      ––––––––– –––––––––      –––––––––      –––––––––      –––––––––      ––  ––  ––––  ––  ––––  ––  –

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

*Блок-схема модуля удаление объектов на форме –––––––––––––––––––––––––––––*

*––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––*

6. Консультант по проекту (с обозначением разделов проекта)    *Воробей В. А. ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––*

7. Дата выдачи задания –––––*03 февраля 2020 г.––––––––––––––––––––––   –*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*Введение (1) с.– 03.02.2020 г. ––––––––––––––––––––––––––   –––––––––––––––*

*1-й раздел (1 – 3) с.–10.02.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*2-й раздел (3 – 4) с.–14.02.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*3-й раздел (2 – 5) с.–21.02.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*4-й раздел (3 – 5) с.–06.03.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*5-й раздел (3 – 5) с.–20.03.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*6-й раздел (3 – 5) с.–27.03.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*7-й раздел (5 – 8) с.–27.04.2020 г. ––––––––––––––––––––––––––   ––––––––––––*

*Заключение (1,2) с.– 06.05.2020 г. –––––––––––––––––––––––––––––––––––––––*

*Защита курсового проекта с 23.05 по 12.06 –––––––––––––––––––––––––––*

*––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ*– ––––––––––––––––Воробей В. А.*

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_– –––––– Селедцова Е.О*

(дата и подпись студента)

Содержание

[Введение 2](#_Toc38723416)

[1 Постановка задачи 4](#_Toc38723417)

[1.1 Описание и анализ предметной области 4](#_Toc38723418)

[1.2 Обзор существующих аналогов 5](#_Toc38723419)

[1.3 Информационная база задачи 7](#_Toc38723420)

[2 Проектирование задачи 8](#_Toc38723421)

[2.1 Интерфейс программного средства 8](#_Toc38723422)

[2.2. Выбор и обоснование инструментов разработки 9](#_Toc38723423)

[3 Программная реализация 11](#_Toc38723424)

[3.1 Физическая структура 11](#_Toc38723425)

[3.2 Описание разработанных модулей 17](#_Toc38723426)

[4 Тестирование 20](#_Toc38723427)

[5 Применение 23](#_Toc38723428)

[5.1 Назначение и условия применения 23](#_Toc38723429)

[5.2 Руководство пользователя 23](#_Toc38723430)

[Заключение 27](#_Toc38723431)

[Приложение А 29](#_Toc38723432)

[Приложение Б 46](#_Toc38723432)

Введение

Компьютерная графика - эта та область информационных поколений, которую ученикам хочется реально увидеть, а не слушать разговоры о ней. Поэтому большое значение имеют демонстрации на компьютере разнообразных продуктов компьютерной графики: красочных рисунков, схем, чертежей, диаграмм, образцов анимационной и трехмерной графики. Следует обратить внимание учеников на то, что любимые многими из них компьютерные игры в большинстве имеют графический интерфейс, причем достаточно сложный. Благодаря существовании прикладных графических пакетов компьютерная графика стала доступна широкому курсу пользователя.

Работа с компьютерной графикой - одна из самых популярных направлений использования персонального компьютера. Для сферы обучения средства компьютерной графики открывают принципиально новые возможности: в процессе анализа изображений учащиеся могут динамически управлять их содержанием, формой, размерами и цветом, добиваясь наибольшей наглядности.

Без компьютерной графики невозможно представить себе не только компьютерный, но и обычный, вполне материальный мир. На сегодняшний день компьютеры и компьютерная графика неотъемлемая часть жизни современного общества. Для примера назовём медицину (компьютерная томография), научные исследования (визуализация строения вещества, векторных полей и других данных), моделирование тканей и одежды, опытно-конструкторские разработки, рекламные щиты, цветные журналы, спецэффекты в фильмах - всё это в той или иной мере имеет отношение к компьютерной графике поэтому созданы программы для создания и редактирования изображений, то есть графические редакторы.

1. Постановка задачи
   1. Описание и анализ предметной области

Графический редактор - программа, позволяющая создавать графические изображения или изменять уже имеющиеся. В настоящий момент, существует достаточно большое количество графических редакторов, как довольно известных, так и не очень распространенных. Изображения в графических редакторах хранятся по-разному.

Растровое изображение хранится с помощью точек различного цвета (пикселей), которые образуют строки и столбцы. Любой пиксель имеет фиксированное положение и цвет. Хранение каждого пикселя требует некоторого количества бит информации, которое зависит от количества цветов в изображении.

Векторные изображения формируются из объектов (точка, линия, окружность и т. д.), которые хранятся в памяти компьютера в виде графических примитивов и описывающих их математических формул. Например, графический примитив точка задается своими координатами (X, У), линия - координатами начала (XI, Y1) и конца (Х2, Y2), окружность - координатами центра (X, У) и радиусом (R), прямоугольник - величиной сторон и координатами левого верхнего угла (XI, У1) и правого нижнего угла (Х2, У2) и т. д. Для каждого примитива назначается также цвет. В контексте данной курсовой работы рассматривается “ 2d и 3d графический редактор”

Основные задачи графического редактора:

* создание примитивов по координатам точек;
* создание примитивов с помощью мышки;
* выделение объектов редактирования;
* изменение цвета;
* заливка фигуры;
* удаление объекта;
* сохраннее объекта в файл;
* загрузка объекта из файла.
* изменение позиции объекта, отображение, поворот, перемещение;
* изменение масштаба рабочей области;
* создание 3d примитивов;
* анимированные графических объектов.

Компьютерной графикой в последнее время занимаются многие, что обусловлено высокими темпами развития вычислительной техники. Более 90% информации здоровый человек получает через зрение или ассоциирует с геометрическими пространственными представлениями. Компьютерная графика имеет огромный потенциал для облегчения процесса познания и творчества.

* 1. Обзор существующих аналогов

Цельюкурсовой работы является разработка 2d и 3d графического редактора.

Графический редактор Paint входит в состав стандартных программ операционной системы Windows. По своим возможностям он значительно уступает профессиональным графическим редакторам, однако располагает необходимым минимумом средств для освоения приемов работы с графической информацией.

Назначение редактора. Программа Paint представляет собой простейший редактор растровой графики, позволяющий создавать черно-белые и цветные рисунки, сохранять их в файлах. Рисунки можно выводить на печать, вставлять в другие документы, использовать в качестве фона Рабочего стола. Редактор Paint можно использовать для просмотра и правки снятых с помощью сканера и фотокамеры фотографий. Paint может использоваться для работы с точечными рисунками формата JPG, GIF, TIFF, PNG и BMP.. Показан на рисунке 1.

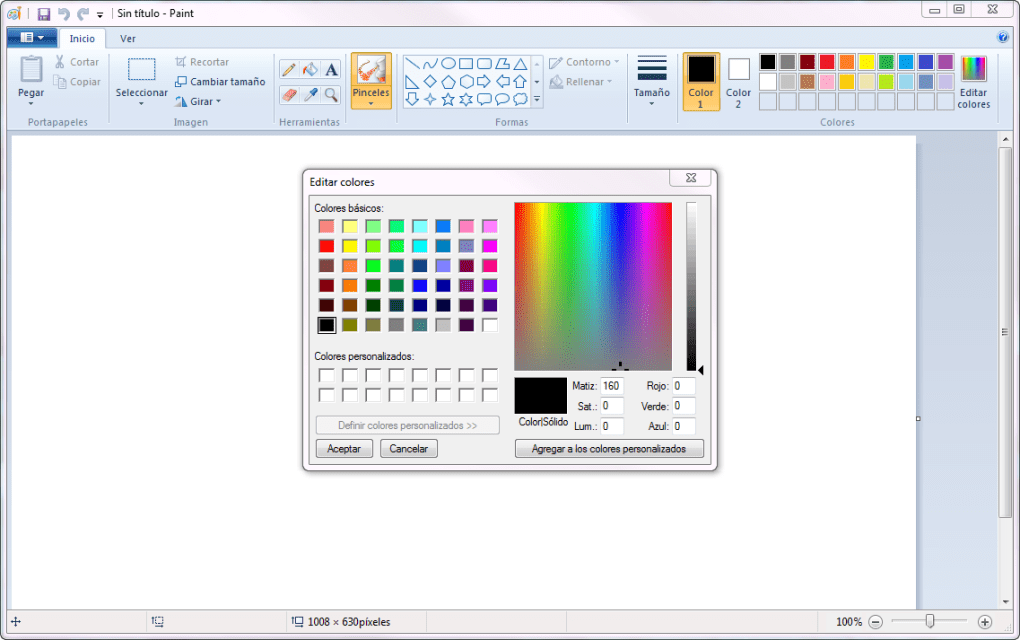


Рисунок 1-программа “ Paint ”

Вторым аналогом является программа KoloutPaint. Это стандартный графический редактор для окружения рабочего стола KDE. Он поддерживает множество форматов изображений PNG, JPG, TGA, PNM, GIF, BMP, XPM. В KolourPaint есть все что нужно для простого редактора: масштабирование, отмена/возврат действий, выделение, копирование, вставка, печать, обрезка, поворот изображения, пипетка выбора цвета и различные эффекты. Здесь также есть полнофункциональная палитра выбора цвета с поддержкой альфа-канала, ручного указания кода цвета, а также экспорта/импорта цветов. Это идеальное приложение для создания простеньких изображений и выполнения несложных корректировок.. Показан на рисунке 2

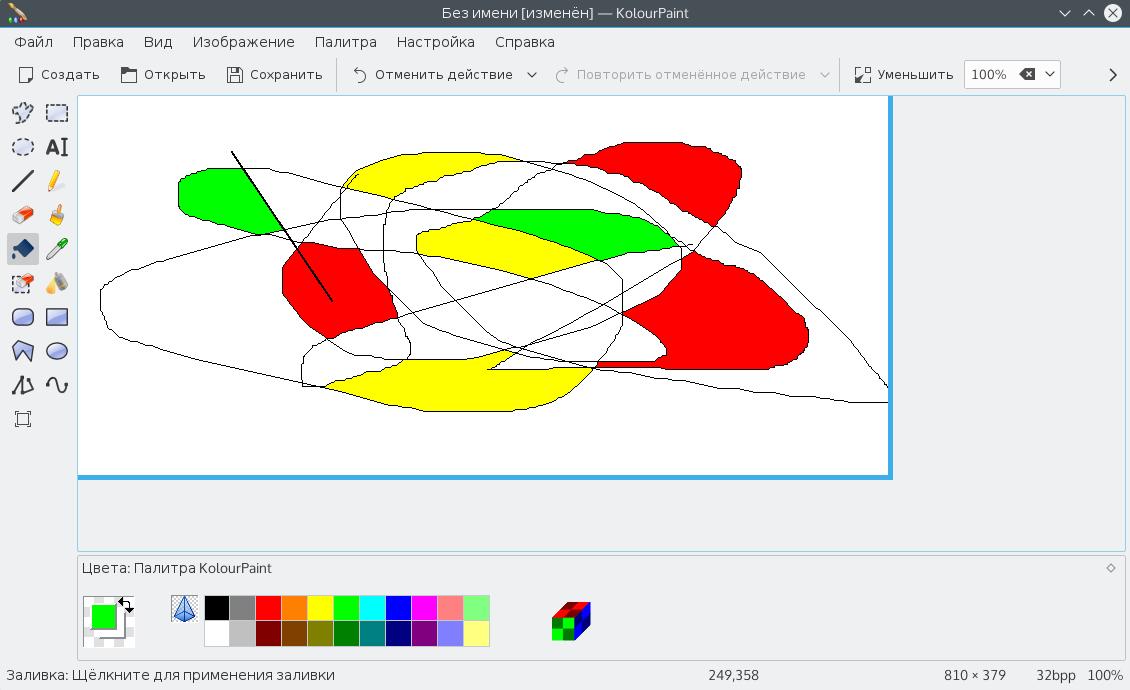


Рисунок 2-программа “ KoloutPaint.”

Основные преимущество данных программ:

* 1) Простота интерфейса;
* 2) Легок в применении;
* 3) Поддерживает большинство форматов изображений;
* 4) Распространяется бесплатно;
* 5) Нет необходимости искать программу отдельно..

Все аналоги имеют свои примущество при работе с материальной базой предприятия. Но разрабатваемый продукт – 2d и 3d графический редактор , имеет возможность моделировать 3 d объекты, а так же анимировать графическкие объекты . Что выделяет данное приложение из существующих аналогов. Данные особенности дополняют недостатки вышеперечисленных аналогов.

* 1. Информационная база задачи

Информационная база задачи представлена входными, выходными и постоянными данными.

Входными данными для данного веб-приложения являются:

* координаты графического объекта;
* открытие объекта из файла.

Выходными данными данного веб-приложения являются:

* отображение графического объекта;
* сохранение в файл.

Постоянными данными являются:

* интерфейс.

2 Проектирование задачи

# 2.1 Интерфейс программного средства

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, направленный на решение конкретной задачи.

Обмен информацией осуществляется передачей сообщений и управляющих сигналов. Сообщение – порция информации, участвующая в диалоговом обмене. По направлению передачи информации различают:

- входные сообщения, которые генерируются человеком с помощью средств ввода (клавиатуры, мыши и т.п.);

- выходные сообщения, которые генерируются компьютером в виде текстов, звуковых сигналов и/или изображений и выводятся пользователю на экран монитора или другие устройства вывода информации.

В основном пользователь генерирует сообщения следующих типов: запрос информации, запрос помощи, запрос операции или функции, ввод или изменение информации . В ответ он получает: подсказки или справки, информационные сообщения, не требующие ответа, приказы, требующие действий, сообщения об ошибках, нуждающиеся в ответных действиях.

Данное программное обеспечение имеет простой и понятный пользователю интерфейс, показано на рисунке 3.

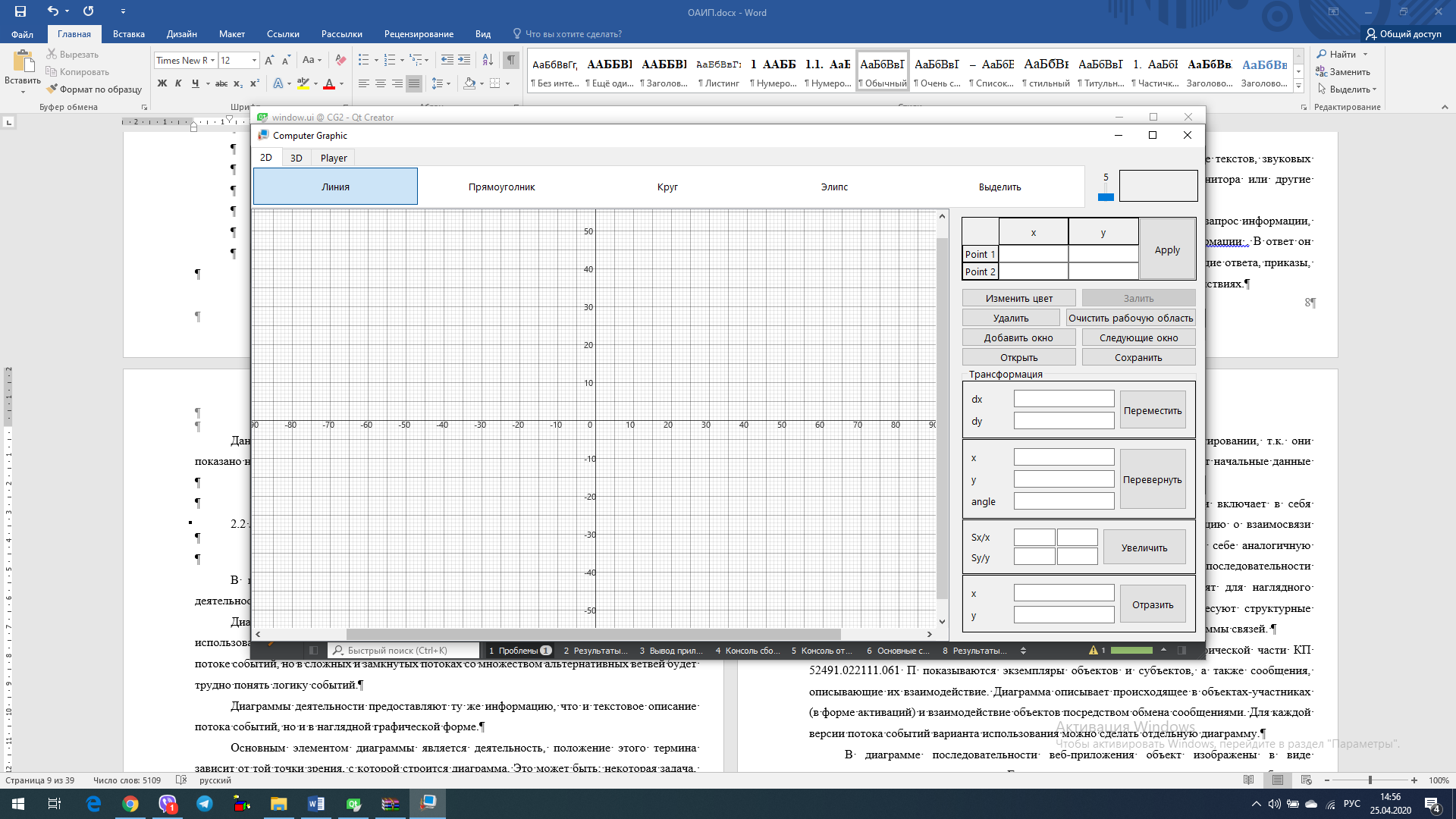


Рисунок 3 -Интерфейс 2D/3D графического редактора

Интерфейс графического редактора содержит: главное меню, рабочую область, панель инструментов, строку заголовка, строку состояния, полосы прокрутки.

# 2.2. Выбор и обоснование инструментов разработки

Для реализации поставленной задачи данного курсового проекта, а именно для создания 2D/3D графический редактор , существует достаточно много сред разработки. Среди таких сред возможность реализовать создание программы есть у Delphi, C++ Builder, Qt и Visual Studio C++.

Все вышеперечисленные среды разработки безусловно функциональны и предназначены для визуального создания программ.

Для данного проекта была выбранна Qt 5.3.0. Эта среда разработки включает в себя большой набор инструментов, достаточна проста в осваивании и тем самым эта IDE отлично подходит для студентов. Особенностью продукта является ориентирование на цель разработки. Данная среда используется для работы на практических занятиях, поэтому привычна в использовании.

Важным фактором в выборе среды разработке послужило удобство интерфейса. Расположение в окне различных панелей инструментов полностью подходят для удобной работы с проектом. В составе каждого типового проекта имеются элементы, которые по желанию можно добавлять в проект. Любой проект, в любом случае, содержит автоматически сгенерированный код, который представляет собой основу будущей программы. Qt предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач. Редактор кода в Qt особенно выигрывает, по сравнению с Delphi и C++ Builder. Предоставляется возможность полностью настроить клавиатуру под себя, настроить любую кнопку на нужное действие. Сам редактор имеет больше возможностей и более интуитивен. Присутствуют такие функции как автоматическое форматирование отступов, запись и воспроизведение макросов. Отладчик Qt не превосходит отладчик Visual C++, но он более чем пригоден, для отладки сложных приложений и с ним не возникает большого количества неудобств. Благодаря удобному интерфейсу и хорошему редактору кода, разработка приложения выполняется быстро и удобно.

Также Qt предоставляет многофункциональный редактор форм для создания графического интерфейса приложения. Конструктор Forms предлагает решения быстрой разработки при создании приложений Windows. Он позволяет наглядно разрабатывать формы на стороне клиента. Элементы управления можно перетащить на поверхность конструктора из панели элементов.

Итак, для выбора среды программирования послужили такие характеристики как: мощный и удобный интерфейс, подходящий набор инструментов, а также функциональный редактор кода. Навыки, полученные на практических занятиях, облегчили работу в данной среде, к тому же способствовали углублению в изучении всех ее возможностей. Данные характеристики позволили написать программу и максимально быстро и качественно выполнить поставленную задачу.

3 Программная реализация

3.1 Физическая структура

Класс main – основной класс точка входа приложение.

Таблица 1 Поля класса main.

|  |  |
| --- | --- |
| Поле | Описание |
| argc | Указывает на количество аргументов |
| argv | Cодержит значения аргументов переданные при запуске. |

Таблица 2 Методы класса main

|  |  |
| --- | --- |
| Метод класса | Описание |
| Window w | Создание объекта Windows |
| w.show() | Отображение окна |
| return a.exec() | Завершение программы |

Класс affine - класс для аффиного преобразования графических объектов

Таблица 3 Поля класса affine.

|  |  |
| --- | --- |
| Поле | Описание |
| transMat | [Транспонирована](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) матрица. |
| scaleMat | Матрица масштаба объекта |

Продолжение таблицы 3

|  |  |
| --- | --- |
| Поле | Описание |
| refMat | Матрица перемещения объекта |
| rotateMat | Матрица для изменение угла объекта. |

Таблица 4 Метода класса affine.

|  |  |
| --- | --- |
| Метод класса | Описание |
| translate | Получение координат с помощью транспортной матрицы |
| rotate | Получение угла поворота (angle) |
| scale | Получение аргумента масштабирования. |
| reflect | Получение координат отображенного объекта |

Класс apidrawer – класс описаны методы реализации создания графических объектов

Таблица 5 Методы класса apidrawer

|  |  |
| --- | --- |
| Метод класса | Описание |
| sqr | Получение квадрата аргумента |
| dist | Проверяет условие параллельности и пересечения прямых |

Таблица.6 Методы класса apidrawer

|  |  |
| --- | --- |
| Метод класса | Описание |
| drawLine | Получение точек координат линнии |
| drawRec**t** | Получение точек координат прямоугольника |

Продолжение таблицы 6

|  |  |
| --- | --- |
| Метод класса | Описание |
| drawCircle | Получение точек координат круга |
| drawEllipse | Получение точек координат элипса |
| drawCube | Получение точек координат куба |
| floodFill | Закрашивание объекта |

Класс circle. Класс-контейнер реализовывающие методы получение данных для построение и закрашиванием круга.

Таблица 7 Методы класса circle

|  |  |
| --- | --- |
| Метод класса | Описание |
| Get/SetFillColor | Получение атрибута цвета закрашивания круга |
| Get/SetR | Получение атрибута радиуса круга |
| Get/SetX | Получение атрибута координат точки x |
| Get/SetY | Получение атрибута координат точки y |

Класс circleinfo. Класс UI-файл Qt Designer представляет собой дерево виджетов формы в формате XML.

Таблица 8 Методы класса circleinfo

|  |  |
| --- | --- |
| Метод класса | Описание |
| setupUi | Функция для построения дерева виджетов на родительском виджете. |

Класс ellipse. Класс элипса

Таблица 9 Методы класса ellipse

|  |  |
| --- | --- |
| Метод класса | Описание |
| Set/GetFillColor | Получение атрибута цвета закрашивания элипса |
| getY/SetYRadius | Получение атрибута радиуса элипса кординаты Y |
| GetX/SetXRadius | Получение атрибута радиуса элипса кординаты X |
| fillEllipse | Закрашивание элипса |
| Get/SetX | Получение атрибута координат точки x |
| Get/SetY | Получение атрибута координат точки y |

Класс frameparser. Класс преобразовывает объекты для выводы на экран графических объектов.

Таблица 10 Методы класса frameparser

|  |  |
| --- | --- |
| Метод класса | Описание |
| setInputFile | Запись из файл и проверка на существующий файл |
| getOutPut | Запись в файл |
| removeObj | Удаление объекта |
| reset | Очистить сцену он всех объектов |
| add | Проверка на тип объекта и получение его свойств |
| rotate | Преобразование объекта для вращение графического объекта |
| scale | Преобразование объекта для изменения масштаба графического объекта |

Продолжение таблицы 10

|  |  |
| --- | --- |
| changeColor | Преобразование объекта для изменения цвета графического объекта |
| fillColor | Преобразование объекта для заливки графического объекта |

Класс item. Класс для получение выделенных объектов.

Таблица 11 Методы класса item

|  |  |
| --- | --- |
| Метод класса | Описание |
| drawPixel | Конструкторы с различными входными данными, для использование полиморфизма. |
| shape | Получение типа графического объекта |
| mousePressEvent | Событие нажатие мышкой на графический объект |

Класс line. Класс-контейнер реализовывающие методы получение данных для построения линии.

Таблица 12 Методы класса line

|  |  |
| --- | --- |
| Метод класса | Описание |
| Line | Конструкторы с различными входными данными, для использование полиморфизма. |
| getType() | Получение типа графического объекта |
| getFillColor() | Получение цвета графического объекта |
| Get/SetY2/Y1 | Получение атрибута координат точек Y1/Y2 |

Продолжение таблицы 12

|  |  |
| --- | --- |
| Метод класса | Описание |
| Get/SetX2/X1 | Получение атрибута координат точек X1/X2 |

Класс scene. Класс реализовывающей методы рабочей области окна.

Таблица 13 Методы класса scene

|  |  |
| --- | --- |
| Метод класса | Описание |
| translateItem | Определение типа графического объекта. |
| rotateItem | Проверка объекта и вращение объекта на рабочей области . |
| scaleItem | Проверка объекта и масштабирование объекта на рабочей области . |
| changeColor | Изменение цвета графического объекта на рабочей области. |
| drawBackground | Отображение основных элементов рабочей области. |

Класс scene3d. Класс реализовывающей методы рабочей области окна.

Таблица 14 Методы класса scene3d

|  |  |
| --- | --- |
| Метод класса | Описание |
| addCube | Конструктор формирует 3d объект дублирую половину графического объекта , возвращает объект куб. |
| addSphere | Конструктор формирует 3d объект дублирую половину графического объекта , возвращает объект сфера. |
| addCylinder | Конструктор формирует 3d объект дублирую половину графического объекта , возвращает объект цилиндр. |

3.2 Описание разработанных модулей

Сегодня практически невозможно представить приложение, в котором нет графического интерфейса пользователя (GUI -Graphical User Interface). Windows API обладает необходимыми инструментами для создания GUI. Однако их использование требует больших затрат времени и практического опыта. Даже библиотека MFC, которая призвана облегчить процесс написания GUI приложений на С++ для ОС Windows, не дает той простоты и легкости создания программ, как хотелось бы. Но самый большой недостаток этих инструментов и библиотек — это платформозависимость. В отличие от MFC библиотека Qt кроссплатформенная. При этом она предоставляет возможности создавать GUI в интерактивном режиме.

Библиотека Qt — это множество классов (более 500), которые определены в модулях, например — QtCore, QtGui, QtWidgets, QtOpenGL. Модуль QtCore в основном отвечает за связь с ОС и управление событиями. Модули QtGui и QtWidgets содержит классы для программирования GUI. Модуль QtOpenGL делает возможным использование OpenGL.

В файле main.cpp сначала создается объект класса QApplication, который осуществляет контроль и управление приложением. Для его создания в конструктор этого класса необходимо передать два аргумента. Первый аргумент представляет собой информацию о количестве аргументов в командной строке, с которой происходит обращение к программе, а второй — это указатель на массив символьных строк, содержащих аргументы, по одному в строке. Любая использующая Qt программа с графическим интерфейсом должна создавать только один объект этого класса, и он должен быть создан до использования операций, связанных с пользовательским интерфейсом. После создания элементы управления Qt по умолчанию невидимы, и для их отображения необходимо вызвать метод show().

В последней строке программы приложение запускается вызовом QApplication::exec(). С его запуском приводится в действие цикл обработки событий, определенный в классе QCoreApplication, являющимся базовым для QApplication. Этот цикл передает получаемые от системы события на обработку соответствующим объектам. Он продолжается до тех пор, пока не будет вызван статический метод QCoreApplication::exit(), либо не закроется окно последнего элемента управления. По завершению работы приложения метод QApplication::exec() возвращает значение целого типа, содержащее код, информирующий о его завершении.

После успешного выполнения предыдущих команд, скомпилированная программа будет размещена в директории текущего проекта (в Windows вы можете использовать nmake вместо make. Исполняемые файлы будут размещены в директориях debug или release, которые создадутся командой make. qmake — это утилита, которая создает файлы конфигурации Qt-проекта, если ей передан аргумент -project. После создания файла конфигурации (.pro), qmake генерирует Makefile, который используется утилитой make для сборки приложения. Позже, мы рассмотрим процесс написания собственных .pro файлов.

Открыть файл можно с помощью функции [open()](http://doc.qt.nokia.com/4.7/qiodevice.html#open). Она возвращает логическое true в случае успешного завершения. Если файл открыть не удалось, мы используем класс [QMessageBox](http://doc.qt.nokia.com/4.7/qmessagebox.html) для того, чтобы показать пользователю всплывающее окно с сообщением об ошибке (смотрите описание класса [QMessageBox](http://doc.qt.nokia.com/4.7/qmessagebox.html), для получения более подробной информации).

Чтение данных становится тривиальной задачей, благодаря классу [QTextStream](http://doc.qt.nokia.com/4.7/qtextstream.html) — оберткой над объектом [QFile](http://doc.qt.nokia.com/4.7/qfile.html). Функция [readAll()](http://doc.qt.nokia.com/4.7/qtextstream.html#readAll) возвращает содержимое файла, как объект [QString](http://doc.qt.nokia.com/4.7/qstring.html). Это содержимое мы поместим в поле для редактирования текста. Затем, мы закрываем файл с помощью функции [close()](http://doc.qt.nokia.com/4.7/qiodevice.html#close), чтобы вернуть файловый дескриптор операционной системе.

Реализация 2D графики реализована с помощью класса QPainter мы можем рисовать на любом объекте,унаследованном от класса QPaintDevice (QWidget, QPrinter, QImage, QGLFramebufferObject и др.). Можно рисовать геометрические фигуры, пиксельные карты, текст. Для начала рассмотрим классы, которые могут быть полезны при работе с QPainter.

Классы QPoint и QPointF служат для задания положения точки в двухмерной системе координат. QPoint — для целых чисел и QPointF — для вещественных. К точкам применимы операции сложения, вычитания, умножения, деления:

Классы QSize и QSizeF служат для хранения размера. Кроме методов, аналогичных методам классов QPoint и QPointF, данные классы обладают методом scale(), позволяющим масштабировать графический объект.

Классы QRect и QRectF служат для хранения прямоугольных областей(координат верхнего левого угла и размера):

QRectF(QPointF point, QSizeF size);

Классы QLine и QLineF описывают прямую линию. Классы QPolygon и QPolygonF описывают замкнутую фигуру, образованную прямыми линиями.

C помощью класса QColor можно хранить информацию о цвете. Среда Qt поддерживает 3 цветовые модели: RGB, CMYK и HSV. Для цветовой модели RGB существует структура QRgb.   
Метод QWidget::paintEvent() вызывается для виджетов, которые должны быть перерисованы.   
В большинстве случаев рисование производится одним объектом QPainter на нескольких объектах рисования. Для сохранения старых настроек рисования(при переходе на новый объект) можно воспользоваться методами QPainter::save() и QPainter::restore(). QPainter::save() помещает в стек установки, а QPainter::restore() — извлекает. Для рисования контуров фигуры необходимо передать объекту класса QPainter объект класса QPen (перо) с помощью QPainter::setPen(). С помощью соответствующих методов можно установить стиль пера(цвет, толщину, вид концов линии). Для заполнения замкнутых контуров используется кисть т.е. объект класса QBrush. По аналогии с классом QPen кисть устанавливается методом QPainter::setBrush(). В него можно передать либо объект QBrush, либо один из предопределенных стилей BrushStyle.

Класс QPainter позволяет поворачивать, масштабировать, смещать объекты, делать скос. Для этих операций имеются соответствующие методы: rotate(), scale(), translate(), sheap().

4 Тестирование

С технической точки зрения, тестирование есть процесс выполнения приложения на некоторых входных данных и проверка получаемых результатов с целью подтвердить их корректность по отношению к ожидаемому результату. Сравнение промежуточных результатов в процессе тестирования с полученными независимо эталонными значениями позволяет локализовать ошибки, выявить их причины и исправить исходный код программы. Одной из основных проблем тестирования является определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождение множества тестов, обладающих этим свойством.

Тестирование ПО может выполняться в соответствии со стратегией «белого ящика» (whitebox testing) или «черного ящика» (blackbox testing).

Тестирование по принципу «белого ящика» (структурное тестирование) предполагает анализ логики работы программы и корректности ее работы при известной внутренней структуре программы. Другими словами, в этом случае тестировщик имеет доступ к исходному коду тестируемых программ и строит тесты, исходя из структуры программ. Исчерпывающее структурное тестирование предполагает выполнение каждого пути в программе, однако на практике выполнение каждого пути обычно не реализуемо, поэтому ограничиваются покрытием операторов, решений и/или условий.

Метод покрытия операторов обеспечивает выполнение каждого оператора программы, по крайней мере, один раз. Метод покрытия решений предполагает, что при тестировании каждое решение примет значения «истина» или «ложь» хотя бы один раз. Покрытие решений обычно удовлетворяет критерию покрытия операторов. Наилучшим методом является покрытие условий, при котором все возможные результаты каждого условия в решении выполняются, по крайней мере, один раз.

Тестирование по принципу «черного ящика» (поведенческое тестирование), которое является наиболее распространенным, заключается в анализе программной системы без использования знаний о внутреннем устройстве тестируемого объекта. В этом случае система представляет собой «черный ящик» для тестировщика, задача которого – проверить соответствие поведения системы заданным требованиям, а также проверить поведение системы в критических ситуациях (при подаче на вход некорректных данных).

Можно выделить три основных этапа тестирования. На первом этапе выполняется ручная разработка тестов или автоматическая генерация для конкретной среды тестирования. По результатам прогона программы на тестах создается протокол результатов тестирования. На завершающем этапе выполняется оценка результатов тестирования и принимается решение о продолжении или остановке тестирования.

Существуют различные признаки, по которым можно выполнить классификацию видов тестирования ПО. В частности, по степени изолированности компонентов тестируемой системы выделяют следующие виды (или уровни) тестирования:

- модульное;

- интеграционное;

- системное.

По степени автоматизации различают ручное и автоматизированное тестирование.

Выполнение тест кейса:

Тест-кейс-Тестирования “2D/3D графского редактора”

Таблица 4.1-Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Действие | Фактический результат | Ожидаемый результат |
| 1 | Создать графический объект “Линия” | В рабочей области графического редактора отображается линия с заданными координатами. | Отобразится линия. |
| 2 | Создать графический объект “Прямоугольник” | В рабочей области графического редактора отображается прямоугольник или квадрат с заданными координатами. | Отобразится прямоугольник или квадрат. |
| 3 | Создать графический объект “Круг” | В рабочей области графического редактора отображается круг или квадрат с заданными координатами. | Отобразится круг. |
| 4 | Создать графический объект “Элипс” | В рабочей области графического редактора отображается элипс или круг или квадрат с заданными координатами. | Отобразится круг или элипс. |
| 5 | Нажать кнопку “Выделить объект” | В рабочей области графического редактора отображается пунктирная область для выделения примитивов. | Отобразится пунктирная область. |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Действие | Ожидаемый результат | Фактический результат |
| 6 | Нажать кнопку “Изменить цвет” | Отображается области с возможной палитрой цветов для изменения цвета объекта. | Отобразится палитра цветов. |
| 7 | Нажать кнопку “Залить” | Отображается области с возможной палитрой цветов для заливки объекта. | Отобразится палитра цветов. |
| 8 | Нажать кнопку “Войти” | Отображается index файл главной старицы. | Переход на главную страницу. |
| 9 | Нажать на кнопку “Удалить” | Удаляется объект с рабочей области. | Удалиться объект. |
| 10 | Нажать кнопку “Загрузить из файла” | Отображается проводник с файлами для загрузки. | Откроется проводник. |
| 11 | Нажать кнопку “Сохранить в файл ” | Отображается проводник с файлами для сохранения. | Откроется проводник. |
| 12 | Нажать на кнопку “Услуги” | Отображается таблица с услугами и кнопками редактирования . | Откроется форма с услугами. |

После выполнения тест-кейсов были обнаружены не глобальные ошибки, которые были исправлены с помощью создание дополнительных файлов для исключения дублирование кода .

5 Применение

5.1 Назначение и условия применения

Функциональное назначение: создание примитивов, заливка, измене цвета, трансформация, создание 3d примитивов, анимированные изображения.

В состав технического обеспечения должны входить технические средства с характеристиками не ниже указанных в скобках:

* системный блок рабочей станции (проц. 1.5Ггц, ОЗУ 512 мб, винчестер 140Гб);
* средства ввода информации (клавиатура, мышь, сканер);
* средства отображения информации (монитор, 19 дюймов);
* средства вывода документации на печать (принтер);
* источники бесперебойного питания для сервера и рабочей станции;
* сервер(3 Ггц, ОЗУ 2 Гб, винчестер не менее 320 Гб);
* телефон;
* устройство обеспечивающее доступ к интернету (ADSL модем);
* сетевое оборудование (2 сетевых адаптера, коммутатор);
* линии связи (кабель типа «неэкранированная витая пара», 25 м).

Техническое обеспечение должно быть создано на базе современных ПК и периферийных устройств, обеспечивающих наилучшее соотношение цены и качества, подходящих для использования в небольших офисах.

Использование полного функционала не требует сложной архитектуру технической и программной структуры.

5.2 Руководство пользователя

.

После успешной инсталляции программного обеспечения пользователю отображается рабочая область ,которая показана на рисунке 4.

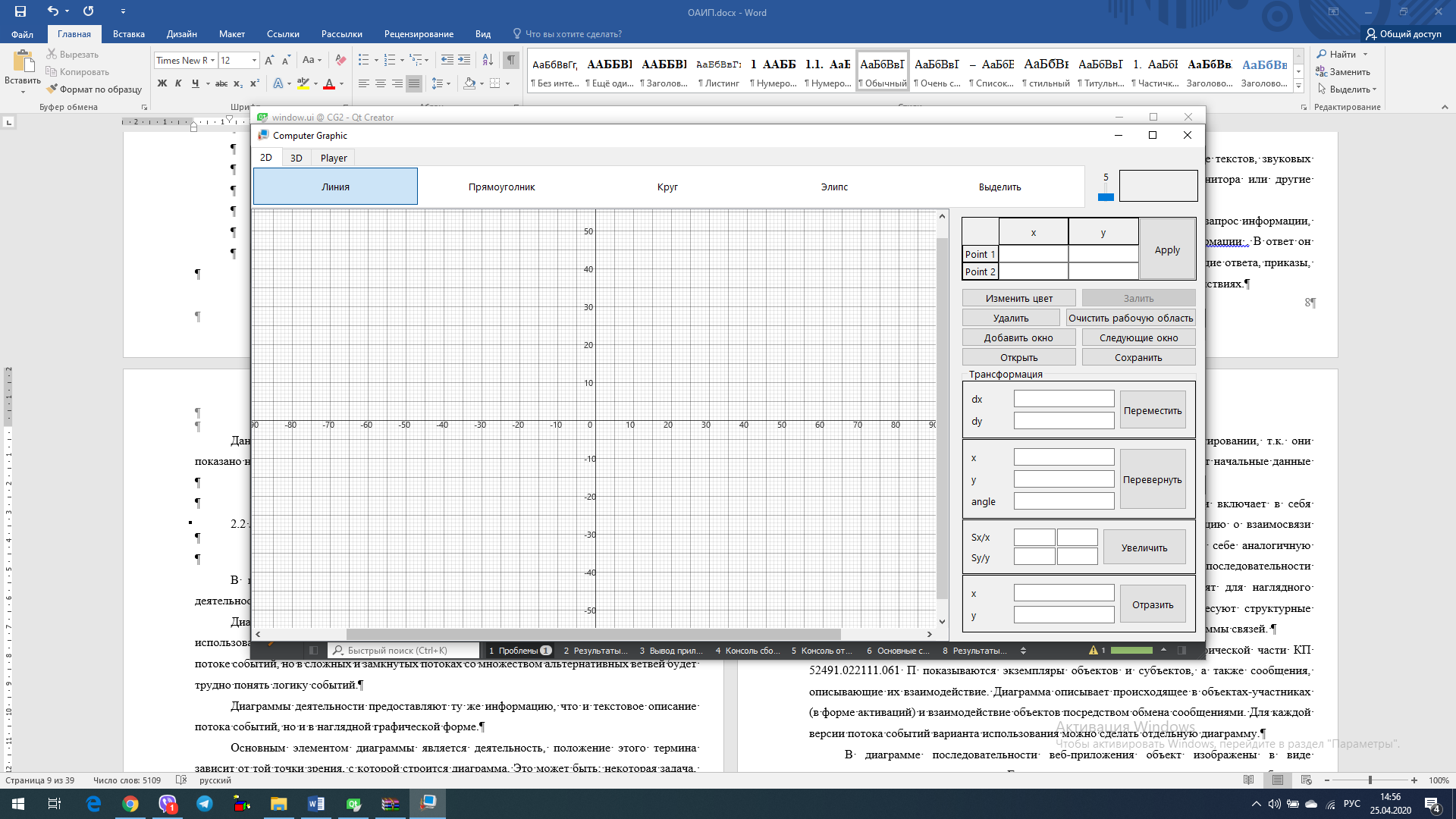


Рисунок 4 –Рабочая область

В верхней части веб-приложения располагается навигационная панель программного средства показано на рисунке 5, где можно выбрать интересующий пользователя раздел.



Рисунок 5–Навигационная панель

В разделе "2D находиться панель инструментов, показана на рисунке 6

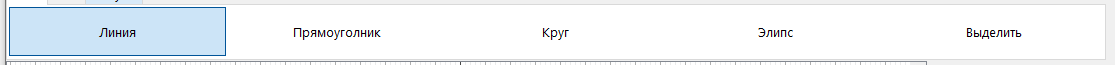


Рисунок 6 – Панель инструментов

В разделе "2D находиться панель для редактирования элементов, показана на рисунке 7

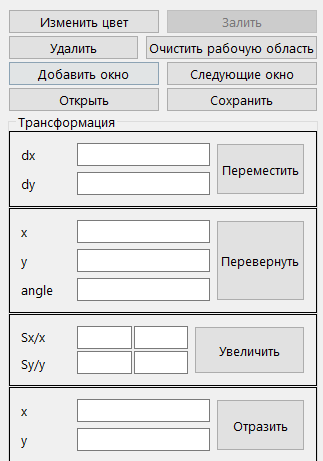


Рисунок 7– Панель для редактирования элментов

В разделе "3D находиться панель инструментов, показана на рисунке 8

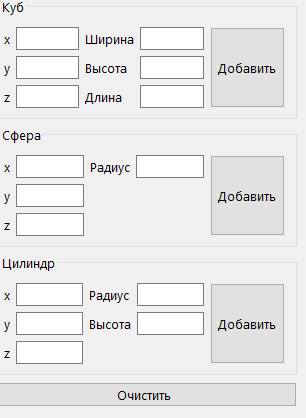


Рисунок 6 – Панель инструментов

В случае возникновения ошибок при работе с программным средством, не описанных ниже в данном разделе, необходимо обращаться к сотруднику подразделения технической поддержки либо к ответственному администратору.

Заключение

Данная курсовая работа потребовала изучения большого количества литературного материала, в результате чего были изучены основные типы графических редакторов, их достоинства и недостатки. Познакомившись с наиболее популярными в настоящее время программами для просмотра графических изображений, была достигнута поставленная цель курсовой работы: создан графический редактор.

В ходе выполнения курсового проекта закреплены понятие объектно-ориентированного программирования, объектно-ориентированного анализа и объектно-ориентированного подхода, навыки по программированию в объектно ориентированной среде.

Программа, описанная в курсовом проекте разработана в соответствии с постановкой задачи на курсовую работу по теме “Разработка графического редактора” по дисциплине “Основы алгоритмизации и программирования”

Интерфейс созданной программы удобен, прост, наглядно отображает ее возможности. Главное меню редактора содержит команды работы с файлами, команды занесения рисунка в файл и вывода рисунка из файла.

Тестирование подтвердило, что программа корректно выполняет обработку данных и демонстрацию результатов.

Всё это свидетельствует о работоспособности программы и позволяет сделать вывод о ее пригодности для создания и редактирования графических изображений.

Литература:

1. Campbell Parallel Programming with Microsoft® Visual C++® / Campbell. - Москва: Гостехиздат, 2011. - 784 c.
2. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - Москва: Высшая школа, 2015. - 882 c.
3. Балена, Франческо Современная практика программирования на Microsoft Visual Basic и Visual C# / Франческо Балена , Джузеппе Димауро. - М.: Русская Редакция, 2015. - 640 c.
4. Боровский, А. C++ и Pascal в Kylix 3. Разработка интернет-приложений и СУБД / А. Боровский. - М.: БХВ-Петербург, 2015. - 544 c.

Приложение А

(обязательное)

Текст программы

Файл sceneanimation.h

1. #include "sceneanimation.h"
2. #include <iostream>
3. #include "window.h"
4. SceneAnimation::SceneAnimation(QWidget \*parent)
5. :Scene (parent)
6. {
7. // setThickness(3);
8. setItemIndexMethod(QGraphicsScene::ItemIndexMethod::NoIndex);
9. parser = new FrameParser(this);
10. connect(&timer, SIGNAL(timeout()), this, SLOT(doAnimation()));
11. }
12. void SceneAnimation::open(const QString &fileName)
13. {
14. if (fileName.isEmpty()) return;
15. if (!parser->setInputFile(fileName)){
16. QMessageBox(QMessageBox::Icon::Critical, "ERROR", "BUG BUG BUG").exec();
17. return;
18. }
19. window->setOpenFileName(fileName.split("/").back());
20. }
21. void SceneAnimation::play()
22. {
23. parser->reset();
24. this->clear();
25. timer.start(window->getDelay());
26. }
27. void SceneAnimation::doAnimation()
28. {
29. if (!parser->nextFrame(window->isStopAtEachFrame())) {
30. timer.stop();
31. if (!window->isPauseAtEnd()) clear();
32. parser->reset();
33. return;
34. }
35. }
36. void SceneAnimation::drawBackground(QPainter \*painter, const QRectF &rect)
37. {
38. if (window->isRulerOn()){
39. Scene::drawBackground(painter, rect);
40. }
41. // else {
42. // int halfHeight = static\_cast<int>(this->height()) / 2;
43. // int halfWidth = static\_cast<int>(this->width()) / 2;
44. // const int halfThick = thickness / 2; //make it center
45. // // Draw ox and oy
46. // painter->setPen(QPen(QBrush(Qt::black), 1));
47. // painter->setOpacity(0.6);
48. // painter->drawLine(0, halfHeight + halfThick, static\_cast<int>(this->width()), halfHeight + halfThick);
49. // painter->drawLine(halfWidth + halfThick, 0, halfWidth + halfThick, static\_cast<int>(this->height()));
50. // }
51. }
52. void SceneAnimation::wheelEvent(QGraphicsSceneWheelEvent \*wheelEvent)
53. {
54. QGraphicsScene::wheelEvent(wheelEvent);
55. #include "scene2d.h"
56. #include "item.h"
57. #include <QDebug>
58. #include <QThread>
59. #include <QTimer>
60. #include <QTextItem>
61. #include <QPoint>
62. #include <QFileDialog>
63. #include <QCoreApplication>
64. #include "window.h"
65. #include "frameparser.h"
66. Scene2D::Scene2D(QWidget \*parent) : Scene(parent)
67. {
68. parser = new FrameParser(this);
69. lineInfo = new LineInfo();
70. rectInfo = new RectInfo();
71. circleInfo = new CircleInfo();
72. ellipseInfo = new EllipseInfo();
73. setItemIndexMethod(QGraphicsScene::ItemIndexMethod::NoIndex);
74. }
75. Scene2D::~Scene2D()
76. {
77. delete parser;
78. }
79. void Scene2D::readTextFile(const QString &fileName)
80. {
81. this->clear();
82. parser->setInputFile(fileName);
83. }
84. bool Scene2D::nextFrame()
85. {
86. return parser->nextFrame();
87. }
88. void Scene2D::toTextFile()
89. {
90. QString outFileName = QFileDialog(window, "Save to file",
91. QCoreApplication::applicationDirPath()).getSaveFileName();
92. QFile outFile(outFileName);
93. if (outFile.open(QIODevice::WriteOnly)){
94. outFile.write(parser->getOutPut().toLatin1());
95. } else {
96. QMessageBox(QMessageBox::Icon::Critical, "Error", "Can't not save file: " + outFileName).exec();
97. }
98. }
99. void Scene2D::doChangeColor(const QColor &color)
100. {
101. for(auto &it : this->selectedItems()){
102. Item \*item = static\_cast<Item\*>(it);
103. parser->outPutChangeColor(item, color);
104. changeColor(item, color);
105. }
106. }
107. void Scene2D::doFillColor(const QColor &color)
108. {
109. for(auto &it : this->selectedItems()){
110. Item \*item = static\_cast<Item\*>(it);
111. parser->outPutFillColor(item, color);
112. changeFillColor(item, color);
113. }
114. }
115. void Scene2D::deleteItem()
116. {
117. auto list = this->selectedItems();
118. for(auto &it : list){
119. Item \*item = static\_cast<Item\*>(it);
120. parser->outPutDeletion(item);
121. parser->removeObj(item);
122. }
123. update();
124. if (!this->items().isEmpty()) {
125. Item \*item = static\_cast<Item\*>(this->items().first());
126. item->setSelected(true);
127. window->setEnableFillButton(item->getType() != Item::Type::LINE);
128. updateInfo(item);
129. } else {
130. lineInfo->setLine(nullptr);
131. rectInfo->setRect(nullptr);
132. circleInfo->setCircle(nullptr);
133. ellipseInfo->setEllipse(nullptr);
134. }
135. }
136. void Scene2D::clearAll()
137. {
138. parser->outPutClear();
139. for(auto &it : this->items()){
140. removeItem(it);
141. delete it;
142. it = nullptr;
143. }
144. this->clear();
145. lineInfo->setLine(nullptr);
146. rectInfo->setRect(nullptr);
147. circleInfo->setCircle(nullptr);
148. ellipseInfo->setEllipse(nullptr);
149. }
150. void Scene2D::doTranslation()
151. {
152. for(auto &it : selectedItems()){
153. Item \*selectedItem = static\_cast<Item\*>(it);
154. if (selectedItem){
155. parser->outPutTranslation(selectedItem, window->getDxTrans(), window->getDyTrans());
156. translateItem(selectedItem, window->getDxTrans(), window->getDyTrans());
157. updateInfo(selectedItem);
158. }
159. }
160. }
161. void Scene2D::doRotation()
162. {
163. for(auto &it : selectedItems()){
164. Item \*selectedItem = static\_cast<Item\*>(it);
165. if (selectedItem){
166. parser->outPutRotation(selectedItem, window->getXRotate(), window->getYRotate(), window->getAngleRotate());
167. rotateItem(selectedItem, window->getXRotate(), window->getYRotate(), window->getAngleRotate());
168. updateInfo(selectedItem);
169. }
170. }
171. }
172. void Scene2D::doScaling()
173. {
174. for(auto &it : selectedItems()){
175. Item \*selectedItem = static\_cast<Item\*>(it);
176. if (selectedItem){
177. parser->outPutScaling(selectedItem, window->getSXScale(), window->getSYScale());
178. // scaleItem(selectedItem, window->getSXScale(), window->getSYScale());
179. scaleItem(selectedItem, window->getSXScale(), window->getSYScale(),
180. window->getXScale(), window->getYScale());
181. updateInfo(selectedItem);
182. }
183. }
184. }
185. void Scene2D::doReflection()
186. {
187. for(auto &it : selectedItems()){
188. Item \*selectedItem = static\_cast<Item\*>(it);
189. if (selectedItem){
190. parser->outPutReflection(selectedItem, window->getXReflection(), window->getYReflection());
191. reflectItem(selectedItem, window->getXReflection(), window->getYReflection());
192. updateInfo(selectedItem);
193. }
194. }
195. }
196. void Scene2D::updateInfo(Item \*item)
197. {
198. if (!item) return;
199. switch (item->getType()) {
200. case Item::Type::LINE:
201. window->setMode(Window::Mode::DRAW\_LINE);
202. lineInfo->setLine(static\_cast<Line\*>(item));
203. break;
204. case Item::Type::RECT:
205. window->setMode(Window::Mode::DRAW\_RECT);
206. rectInfo->setRect(static\_cast<Rectangle\*>(item));
207. break;
208. case Item::Type::CIRCLE:
209. window->setMode(Window::Mode::DRAW\_CIRCLE);
210. circleInfo->setCircle(static\_cast<Circle\*>(item));
211. break;
212. case Item::Type::ELLIPSE:
213. window->setMode(Window::Mode::DRAW\_ELLIPSE);
214. ellipseInfo->setEllipse(static\_cast<Ellipse\*>(item));
215. break;
216. default:
217. break;
218. }
219. }
220. void Scene2D::setThickness(int value)
221. {
222. Scene::setThickness(value);
223. for(auto &item : this->items()){
224. if (!item) continue;
225. switch (static\_cast<Item\*>(item)->getType()) {
226. case Item::Type::LINE:
227. static\_cast<Line\*>(item)->reDraw();
228. break;
229. case Item::Type::RECT:
230. static\_cast<Rectangle\*>(item)->reDraw();
231. break;
232. case Item::Type::CIRCLE:
233. static\_cast<Circle\*>(item)->reDraw();
234. break;
235. case Item::Type::ELLIPSE:
236. static\_cast<Ellipse\*>(item)->reDraw();
237. break;
238. default: break;
239. }
240. }
241. update();
242. }
243. void Scene2D::addScene()
244. {
245. parser->outPutAddition();
246. }
247. void Scene2D::addItemFrom2Points(const QPoint &p1, const QPoint &p2)
248. {
249. if (tmpItem) {
250. removeItem(tmpItem);
251. delete tmpItem;
252. tmpItem = nullptr;
253. }
254. switch (window->getMode()){
255. case Window::Mode::DRAW\_LINE :
256. tmpItem = new Line(p1, p2, this);
257. break;
258. case Window::Mode::DRAW\_RECT :
259. tmpItem = new Rectangle(QPoint(std::min(p1.x(), p2.x()), std::max(p1.y(), p2.y())),
260. QSize(std::abs(p2.x() - p1.x()), std::abs(p2.y() - p1.y())), this);
261. break;
262. case Window::Mode::DRAW\_CIRCLE :
263. tmpItem = new Circle(p1.x(), p1.y(),
264. Drawer::dist(p1, p2), this);
265. break;
266. case Window::Mode::DRAW\_ELLIPSE :
267. tmpItem = new Ellipse(p1.x(), p1.y(),
268. std::abs(p2.x() - p1.x()),
269. std::abs(p2.y() - p1.y()), this);
270. break;
271. default:
272. break;
273. }
274. tmpItem->setSelected(true);
275. addItem(tmpItem);
276. tmpItem->update();
277. updateInfo(tmpItem);
278. window->setEnableFillButton(tmpItem->getType() != Item::Type::LINE);
279. }
280. void Scene2D::addBoundingRect(const QPoint &p1, const QPoint &p2)
281. {
282. QPoint sceneP1 = toScenePos(p1);
283. QPoint sceneP2 = toScenePos(p2);
284. if (tmpSelected){
285. removeItem(tmpSelected);
286. delete tmpSelected;
287. tmpSelected = nullptr;
288. }
289. tmpSelected = new QGraphicsRectItem(
290. std::min(sceneP1.x(), sceneP2.x()),
291. std::min(sceneP1.y(), sceneP2.y()),
292. std::abs(sceneP1.x() - sceneP2.x()),
293. std::abs(sceneP1.y() - sceneP2.y())
294. );
295. tmpSelected->setPen(QPen(Qt::black, 1, Qt::DashLine));
296. setSelectionArea(tmpSelected->shape());
297. addItem(tmpSelected);
298. tmpSelected->setSelected(false);
299. window->setEnableFillButton(true);
300. }
301. void Scene2D::mousePressEvent(QGraphicsSceneMouseEvent \*mouseEvent)
302. {
303. points.clear(); // TODO: add polygon
304. window->showMousePos(toUserCoordinate(mouseEvent->scenePos()));
305. QGraphicsScene::mousePressEvent(mouseEvent);
306. if (mouseEvent->button() == Qt::LeftButton){
307. const QPoint userCoor = toUserCoordinate(mouseEvent->scenePos());
308. window->showMousePos(userCoor);
309. points.emplace\_back(userCoor);
310. points.emplace\_back(userCoor);
311. if (window->getMode() == Window::Mode::SELECT\_ITEMS){
312. isDrawing = false;
313. QGraphicsItem \*selectedItem = (this->itemAt(mouseEvent->scenePos(), QTransform()));
314. if (selectedItem){
315. Item \*item = static\_cast<Item\*>(selectedItem);
316. updateInfo(item);
317. window->setEnableFillButton(item->getType() != Item::Type::LINE);
318. if (!(mouseEvent->modifiers() & Qt::ControlModifier)){
319. clearSelection();
320. }
321. selectedItem->setSelected(true);
322. }
323. } else {
324. isDrawing = true;
325. clearSelection();
326. addItemFrom2Points(points.front(), points.back());
327. }
328. }
329. }
330. void Scene2D::mouseReleaseEvent(QGraphicsSceneMouseEvent \*mouseEvent)
331. {
332. Q\_UNUSED(mouseEvent);
333. if (window->getMode() == Window::Mode::SELECT\_ITEMS){
334. if (tmpSelected) {
335. removeItem(tmpSelected);
336. delete tmpSelected;
337. tmpSelected = nullptr;
338. }
339. } else {
340. parser->outPutItem(tmpItem);
341. tmpItem = nullptr;
342. }
343. isDrawing = false;
344. points.clear();
345. // QGraphicsScene::mouseReleaseEvent(mouseEvent);
346. }
347. void Scene2D::mouseMoveEvent(QGraphicsSceneMouseEvent \*mouseEvent)
348. {
349. window->showMousePos(toUserCoordinate(mouseEvent->scenePos()));
350. if (isDrawing){
351. points.pop\_back();
352. points.emplace\_back(toUserCoordinate(mouseEvent->scenePos()));
353. addItemFrom2Points(points.front(), points.back());
354. } else if (window->getMode() == Window::Mode::SELECT\_ITEMS){
355. if (!points.empty()) points.pop\_back();
356. points.emplace\_back(toUserCoordinate(mouseEvent->scenePos()));
357. addBoundingRect(points.front(), points.back());
358. }
359. QGraphicsScene::mouseMoveEvent(mouseEvent);
360. }
361. EllipseInfo \*Scene2D::getEllipseInfo() const
362. {
363. return ellipseInfo;
364. }
365. CircleInfo \*Scene2D::getCircleInfo() const
366. {
367. return circleInfo;
368. }
369. LineInfo \*Scene2D::getLineInfo() const
370. {
371. return lineInfo;
372. }
373. RectInfo \*Scene2D::getRectInfo() const
374. {
375. return rectInfo;
376. }

}

Приложение Б

Блок-схема алгоритма отмены последних действий

