Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

РАЗРАБОТКА ГРАФИЧЕСКОГО РЕДАКТОРА

ОТЧЕТ
ПО РЕЗУЛЬТАТАМ
Учебной практики:
(вид практики)
Получение первичных навыков научно-исследовательской работы (рассред.)
(тип практики)

Обучающийся гр. 430-4

_____ П.А. Куминов
(подпись) (И.О. Фамилия)

«___» _____ 2021 г.
(дата)

Руководитель практики
от Университета:
Старший преподаватель каф. АСУ,
(должность, ученая степень, звание)

_____ _____ А. Е. Косова
(оценка) (подпись) (И.О. Фамилия)

«___» _____ 2021 г.
(дата)

Томск 2021

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

УТВЕРЖДАЮ
Зав. кафедрой АСУ
канд. техн. наук, доц.
В. В. Романенко

(подпись)
«___» _____ 2021 г.
(дата)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на Учебную практику:
(вид практики)

Получение первичных навыков научно-исследовательской работы (рассред.)
(тип практики)

студенту гр. 430-4 факультета ФСУ

Куминова Павла Алексеевича
(Ф.И.О. студента)

1. Тема практики: Разработка графического редактора
2. Цель практики: Получить навыки разработки приложений, получить навыки командной разработки, изучить место графических редакторов в современном мире
3. Задачи практики: Описать классы и структуру приложения на UML, реализовать класс фигур, реализовать манипулирование с файлами
4. Сроки практики: с «___» _____ 2021 г. по «___» _____ 2021 г.

Совместный рабочий график (план) проведения практики

№ п/п	Перечень заданий	Сроки выполнения
1	Выбор языка программирования и фреймворка	15.09.2021
2	Определение функционала приложения	21.09.2021
3	Описание классов и структуры приложения на UML	29.09.2021
4	Реализация класса фигур	24.11.2021
5	Реализация функций манипулирования с файлами	26.11.2021
6	Комплексное тестирование приложения	29.11.2021

Дата выдачи задания: «___» сентября 2021 г.

Руководитель практики от Университета

Старший преподаватель каф. АСУ _____ А. Е. Косова
(должность, ученая степень, звание) (подпись) (Ф.И.О.)

Задание принял к исполнению: «___» сентября 2021 г.

Студент гр. 430-4 _____ Куминов П. А.
(подпись) (Ф.И.О.)

Оглавление

Введение

Графические редакторы занимают среди программного обеспечения особое место. Это как раз тот случай, когда требования к программному обеспечению дизайнеров-профессионалов и основной массы пользователей не совпадают. При наборе текстов и профессиональный писатель, и семиклассник, пишущий сочинение, выполняют одни и те же операции и предъявляют к программе сходные требования. Другое дело – графика. Здесь большинству рядовых пользователей требуется совсем иное, нежели профессионалам. Ведь они используют для своей работы, как правило, уже готовые изображения. На первое место выступает возможность быстрого просмотра имеющихся картинок, удобного преобразования из одного формата в другой, простейших, интуитивно понятных манипуляций: масштабирование, яркость, контрастность, резкость, инвертирование, повороты и т. п. Благодаря компьютерной графики можно оформить Интернет-страницу, создать рекламу или просто редактировать собственные домашние фото.

Современная компьютерная графика – это достаточно сложная, основательно проработанная и разнообразная научно-техническая дисциплина. Некоторые ее разделы, такие как геометрические преобразования, способы описания кривых и поверхностей, к настоящему времени уже исследованы достаточно полно. Ряд областей продолжает активно развиваться: методы растрового сканирования, удаление невидимых линий и поверхностей, моделирование цвета и освещенности, текстурирование, создание эффекта прозрачности и полупрозрачности и др.

1 Описание предметной области

Графический редактор – программа (или пакет программ), позволяющая создавать, просматривать, обрабатывать и редактировать цифровые изображения (рисунки, картинки, фотографии) на компьютере[3].

Графические редакторы подразделяются на два типа:

- Векторный графический редактор
- Растровый графический редактор

Растровый графический редактор – специализированная программа, предназначенная для создания и обработки растровых изображений, то есть графики, которая в память компьютера записывается как набор точек, а не как совокупность формул геометрических фигур[4].

Векторные графические редакторы позволяют пользователю создавать и редактировать векторные изображения, объекты которых основаны на математическом описании элементарных геометрических объектов, обычно называемых примитивами[5].

Для некоторого обобщённого графического редактора характерно выполнение следующих функций:

- Создание рисунка: в режиме ручной прорисовки; с использованием панели инструментов (штампов, примитивов).

- Манипулирование рисунком (выделение фрагментов рисунка; проработка мелких деталей рисунка (увеличение фрагментов картины); копирование фрагмента рисунка на новое место экрана (а также возможность вырезать, склеивать, удалять фрагменты изображения); закраска отдельных частей рисунка ровным слоем или узором, возможность применять для рисования произвольные "краски", "кисти" и "напыление"; масштабирование изображения; перемещение изображения; поворот изображения.

- Ввод в изображение текста (выбор шрифта; выбор символов (курсив, подчёркивание, применения полужирного начертания)).

- Работа с цветами: создание своей палитры цветов; создание своего узора (штампа) для закраски.

- Работа с внешними устройствами (диски, принтер, сканер и др.): запись рисунка на диск (дискету) в виде файла стандартного формата (pcx, bmp, tif, gif, jpg, png и др.); чтение файла с диска (дискеты); печать рисунка; сканирование рисунка[6].

2 Обзор программного обеспечения

2.1 Язык программирования C++

C++ – компилируемый, статически типизированный язык программирования общего назначения.

Тот факт, что C++ – компилируемый язык, автоматически делает его более эффективным по сравнению с интерпретируемыми языками, инструкции которых переводятся в двоичный код непосредственно во время исполнения программы.

C++ является статически типизированным языком, т.е. тип каждой сущности (например, объекта, значения, имени или выражения) должен быть известен компилятору в точке использования. Тип объекта определяет набор применимых к нему операций. Таким образом, избегаются ошибки, когда объекту, подразумевающему под собой один конкретный тип, присваивается значение объекта другого типа.

Этот язык поддерживает различные парадигмы программирования: процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, функциональное программирование.

Как уже говорилось, C++ — язык общего назначения. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр.

C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. На C++ можно написать процедурный низкоуровневый код, обращаясь к памяти напрямую, или использовать высокоуровневые концепции вроде объектно-ориентированного и функционального программирования.

2.2 Фреймворк Qt

2.2.1 Общая информация о фреймворке

Qt – фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Отличительная особенность — использование метаобъектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Для ускорения и упрощения создания пользовательских интерфейсов Qt предоставляет программу Qt Designer, позволяющую делать это в интерактивном режиме. Очень сильно повысить скорость создания пользовательских интерфейсов можно так же и при помощи технологии Qt Quick, модули и инструменты которой являются неотъемлемой частью Qt.

Одним из преимуществ фреймворка Qt - подробная документация, сопровождающаяся большим количеством примеров. Исходный код примеров содержит подробные комментарии и описание, что также упрощает изучение Qt.

2.2.2 Описание используемых классов фреймворка

2.2.2.1 Класс QGraphicsItem

Класс QGraphicsItem обеспечивает легкую основу для написания ваших собственных пользовательских элементов. Этот класс включает в себя определение геометрии элемента, обнаружение столкновений, реализацию его рисования и взаимодействие элемента через его обработчики событий. QGraphicsItem является частью Graphics View Framework.

Чтобы написать свой собственный графический элемент, вы сначала создаете подкласс QGraphicsItem, а затем начинаете с реализации его двух чистых виртуальных общедоступных функций: boundingRect(), который возвращает размер области, нарисованного элемента, и paint(), который реализует собственно рисование элемента.

QGraphicsView использует boundingRect(), как для отбраковки невидимых элементов, так и для определения области, которая должна быть перекомпонована при рисовании перекрывающихся элементов.

2.2.2.2 Класс QLinearGradient

Класс QLinearGradient используется в сочетании с классом QBrush для задания кисти линейного градиента.

Линейные градиенты интерполируют цвета между начальной и конечной точками. Вне этих точек градиент либо дополняется, либо отражается, либо повторяется в зависимости от текущего установленного метода распространения.

Цвета градиента определяются с помощью точек остановки типа QGradientStop, то есть позиции и цвета. Функция QGradient::setColorAt(), QGradient::setStops() используются для определения точек остановки. Это полный набор точек остановки градиента, который описывает, как должна быть заполнена область градиента. Если точки остановки не указаны, используется градиент от черного в точке 0 до белого в точке 1.

2.2.2.3 Класс QImage

Класс QImage обеспечивает аппаратно-независимое представление изображения, которое обеспечивает прямой доступ к данным пикселей и может использоваться в качестве устройства рисования.

Класс QImage поддерживает несколько форматов изображений: монохромные, 8-битные, 32-битные и альфа-смешанные.

QImage предоставляет несколько способов загрузки файла изображения: файл можно загрузить при создании объекта QImage или с помощью функций load() или loadFromData() позже. QImage также предоставляет статическую функцию fromData(), конструирующую QImage из заданных данных. При загрузке изображения имя файла может относиться либо к реальному файлу на диске, либо к одному из встроенных ресурсов приложения.

Вызова функции save() досаточно, чтобы сохранить объект QImage.

2.3 Qt Creator

Qt Creator – кроссплатформенная свободная IDE для разработки на C, C++, JavaScript и QML. Разработана Trolltech для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому среди возможностей, присущих любой среде разработки, есть и специфичные, такие как отладка приложений на QML и отображение в отладчике данных из контейнеров Qt, встроенный дизайнер интерфейсов: как на QML, так и на QtWidgets.

В Qt Creator реализовано автодополнение, в том числе ключевых слов, введённых в стандарте C++11, подсветка кода. Также, начиная с версии 2.4,

есть возможность задания стиля выравнивания, отступов и постановки скобок.

Реализован ряд возможностей при работе с сигнатурами методов, а именно:

- автогенерация пустого тела метода после его обновления;

- возможность автоматически изменить сигнатуру метода в определении, если она была изменена в объявлении и наоборот;

- возможность автоматически поменять порядок следования аргументов.

При навигации по коду доступно переключение между определением и объявлением метода, переход к объявлению метода, переименование метода как в отдельном проекте, так и во всех открытых. Также есть возможность вызвать справку согласно текущему контексту.

2.4 GitHub

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc. Сервис бесплатен для проектов с открытым исходным кодом и – с 2019 года – небольших частных проектов, предоставляя им все возможности, а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

2.5 diagrams.net

diagrams.net – это бесплатное онлайн-программное обеспечение для создания диаграмм. Его можно использовать в качестве средства создания блок-схем, программного обеспечения для создания сетевых диаграмм, для создания UML в Интернете, в качестве инструмента для создания диаграмм ER, для разработки схемы базы данных, для создания BPMN в Интернете, в

качестве средства создания схем и т.д. draw.io может импортировать файлы .vsdx, Gliffy и Lucidchart.

Сервис предоставляет возможность хранить диаграммы на различных платформах. Для этого существует множество различных интеграций с другими платформами и приложениями, включая Atlassian Confluence Cloud и Jira Cloud, приложения Google, GitHub и приложения Microsoft. Также доступны неофициальные интеграции для многих других платформ и инструментов.

3 Обзор нормативной документации

3.1 Унифицированный язык моделирования

3.1.1 Что такое UML?

Унифицированный язык моделирования (Unified Modeling Language, UML) – это универсальный язык визуального моделирования систем.

Хотя чаще всего UML ассоциируется с моделированием объектно-ориентированных программных систем, он имеет намного более широкое применение благодаря свойственной ему расширяемости.

Основная идея UML – возможность моделировать программное обеспечение и другие системы как наборы взаимодействующих объектов. Это, конечно же, замечательно подходит для объектно-ориентированных программных систем и языков программирования, но также очень хорошо работает и для бизнес-процессов и других прикладных задач.

В UML модели есть два аспекта:

- Статическая структура – описывает, какие типы объектов важны для моделирования системы и как они взаимосвязаны.

- Динамическое поведение – описывает жизненные циклы этих объектов и то, как они взаимодействуют друг с другом для обеспечения требуемой функциональности системы.

3.1.2 Сущности

Сущности – это существительные UML- модели.

Все UML-сущности можно разделить на:

- структурные сущности – существительные UML-модели, такие как класс, интерфейс, кооперация, прецедент, активный класс, компонент, узел;

- поведенческие сущности – глаголы UML-модели, такие как взаимодействия, деятельности, автоматы;

- группирующая сущность – пакет, используемый для группировки семантически связанных элементов модели в образующие единое целое модули;

- аннотационная сущность – примечание, которое может быть добавлено к модели для записи специальной информации.

3.1.3 Отношения

Отношения позволяют показать взаимодействие в пределах модели двух или более сущностей. Отношения в модели UML позволяют зафиксировать значимые (семантические) связи между сущностями. На рис. 3.1 представлены UML-отношения, применяемые к структурным и группирующим сущностям модели.

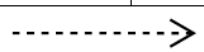
Тип отношения	UML-синтаксис		Краткая семантика
	источник	цель	
Зависимость			Исходный элемент зависит от целевого элемента и изменение последнего может повлиять на первый.
Ассоциация			Описание набора связей между объектами.
Агрегация			Целевой элемент является частью исходного элемента.
Композиция			Строгая (более ограниченная) форма агрегирования.
Включение			Исходный элемент содержит целевой элемент.
Обобщение			Исходный элемент является специализацией более обобщенного целевого элемента и может замещать его.
Реализация			Исходный элемент гарантированно выполняет контракт, определенный целевым элементом.

Рисунок 3.1 – Типы отношений

3.1.4 Объекты

Объект – отдельная сущность с явно выраженными границами, которая инкапсулирует состояние и поведение; экземпляр класса.

Объекты объединяют данные и функциональность в единый блок. Объект можно представить как единый пакет данных и функциональности. Как правило, единственный путь добраться до данных объекта – вызвать одну из предоставляемых им функций.

Свойства, присущие всем объектам:

- Идентификатор – это определение существования и единственности объекта во времени и пространстве. Это то, что отличает его от всех остальных объектов.

- Состояние – определяется значениями атрибутов объекта и его отношениями с другими объектами в конкретный момент времени.

- Поведение – конкретные действия, которые может выполнять объект.

Операция – это описание части поведения. Реализация этого поведения называется методом.

Вызов операции объекта всегда приводит к изменению значений одного или более его атрибутов или отношений с другими объектами. Это может обусловить переход состояний – целенаправленный переход объекта из одного состояния в другое.

3.1.5 Сообщения

Объекты кооперируются для осуществления функций системы. Это означает, что они устанавливают связи с другими объектами и обмениваются сообщениями по этим связям. Когда объект получает сообщение, он проверяет набор своих операций в поиске той, сигнатура которой соответствует сигнатуре сообщения. Если таковая имеется, он инициирует эту операцию. В сигнатуру входят имя сообщения (или операции), типы параметров и возвращаемое значение.

Пример обмена сообщениями показан на рис. 3.2.

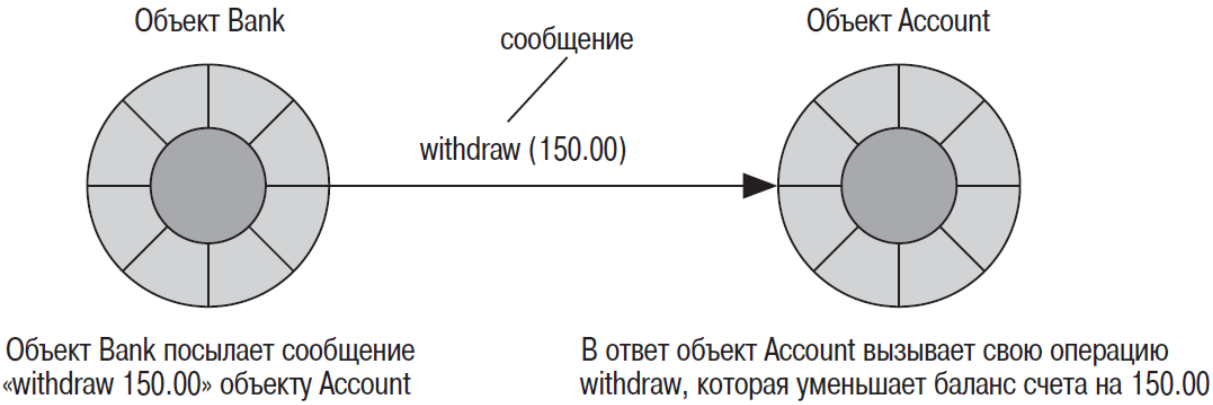


Рисунок 3.2 – Пример обмена сообщениями

3.1.6 Нотация объектов

Пиктограмма объекта в UML – это прямоугольник с двумя ячейками. В верхней ячейке размещается идентификатор объекта, который всегда подчеркивается.

UML очень гибок относительно представления объектов на диаграммах объектов. Идентификатор объекта может включать следующие элементы:

- Только имя класса. Это означает, что имеется анонимный объект или экземпляр данного класса. Анонимные объекты обычно используются, когда на данной диаграмме присутствует только один объект этого конкретного класса. Если необходимо показать два объекта одного и того же класса, каждому из них должно быть присвоено уникальное имя, чтобы можно было их различать.

- Только имя объекта. Обозначен конкретный объект, но не указано, какому классу он принадлежит. Данное обозначение может быть полезным на самых ранних стадиях анализа, когда еще не выявлены все классы.

- Если указываются и имя объекта, и имя класса, они разделяются двоеточием. Двоеточие может читаться как «является экземпляром класса».

Имена объектов обычно записываются заглавными и строчными буквами вперемежку, начиная со строчной буквы. Следует избегать специальных символов, таких как пробелы и подчеркивания. Такой стиль записи называют lowerCamelCase.

Поскольку все объекты одного класса имеют совершенно одинаковый набор операций, они перечисляются в пиктограмме класса, а не в пиктограмме объекта.

Атрибуты по выбору могут быть приведены в нижней ячейке пиктограммы объекта. Тем атрибутам, которые решено вынести на диаграмму, должны быть присвоены имена. Их тип и значение указывать необязательно. Имена атрибутов также записываются в стиле lowerCamelCase.

3.1.7 Нотация классов

Чтобы синтаксис был управляемым, в UML существует понятие необязательных дополнений. Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны.

На рис. 3.3 показана нотация класса в UML со всеми ячейками и дополнениями.

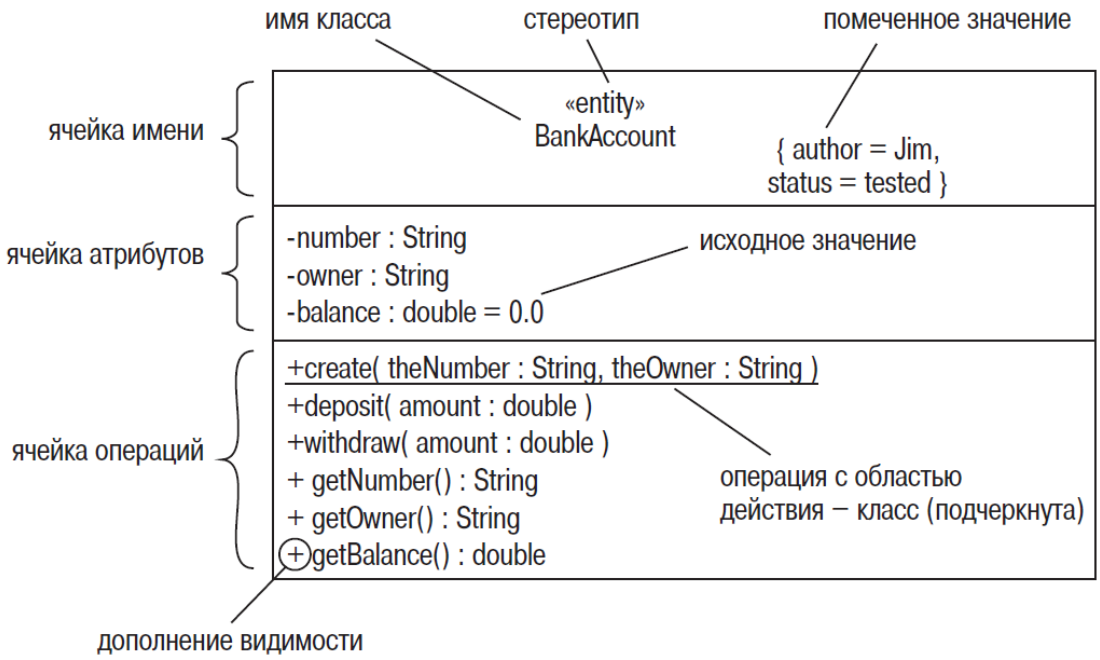


Рисунок 3.3 – Нотация классов в UML

В аналитических моделях обычно необходимо показывать только:

- имя класса;

- ключевые атрибуты;

- ключевые операции;

- стереотипы (если они приносят пользу делу).

Обычно не показывают следующее:

- помеченные значения;

- параметры операций;

- видимость;

- исходные значения (если только они не значимы для дела).

Хотя UML не определяет для классов никаких обязательных соглашений о присваивании имен, существует общепринятое соглашение:

- Имя класса записывается в стиле UpperCamelCase: имя и все слова, его образующие, пишутся с заглавной буквы. Специальные символы, такие как знаки препинания, тире, подчеркивание, «&», «#» и слэш, никогда не используются.

- Во что бы то ни стало необходимо избегать сокращений. Имена классов всегда должны отражать имена реальных сущностей без сокращений.

- Если есть специальные акронимы данной предметной области, широко используемые и понятные всем пользователям модели, они могут применяться.

Единственной обязательной частью UML-синтаксиса для атрибутов является имя атрибута. Имена атрибутов записываются в стиле lowerCamelCase. Обычно в качестве имен атрибутов используются имена существительные или именные группы, потому что атрибуты указывают на некоторую «сущность». Необходимо избегать специальных символов и сокращений.

Дополнение видимости применяется к атрибутам и операциям класса.

Видимость контролирует доступ к свойствам класса.

Все дополнения видимости указаны в таблице 3.1.

4 Описание структуры разрабатываемой программы

4.1 Функциональность приложения

Приложение включает в себя функции манипулирования с файлами, различные функции рисования на холсте, их настройку, вставку текста на холст, очистку холста, логирование сообщений от пользователей, руководство пользователя.

Манипулирование с файлами включает в себя: сохранение, открытие и создание файла.

Приложение содержит следующие инструменты для создания и редактирования изображения: рисование кистью, рисование геометрических примитивов, заливку холста и примитивов, разлиновку.

Инструменты имеют следующие изменяемые параметры: размер кисти, цвет, стиль штриха, вид заливки, дублирование.

В приложении реализовано логирование отзывов – генерирование файла с данными, который пользователь может отправить на почту разработчикам, указанную в мануале.

Каждая функция в приложении имеет клавишу быстрого доступа.

4.2 Общая схема приложения

Общая схема приложения показана на рис. 4.1.

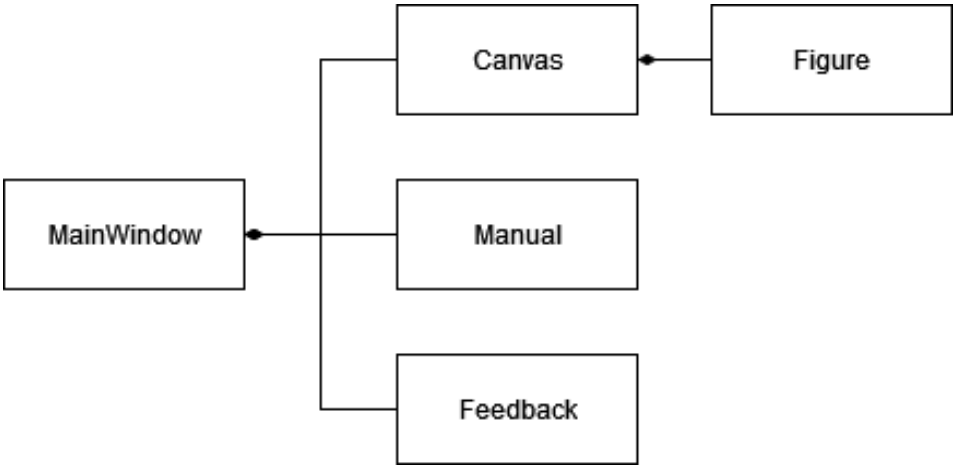


Рисунок 4.1 – UML-диаграмма классов приложения.

Экземпляры классов Canvas, Manual и Feedback создаются вместе с MainWindow и существуют до тех пор, пока существует MainWindow.

Экземпляры класса Figure создаются при использовании инструментов «Фигуры», «Заливка» и существуют до тех пор, пока существует класс Canvas.

4.3 Описание класса MainWindow

Класс MainWindow является основным классом приложения и представляет из себя главное окно приложения, содержащее панель с выпадающими меню «Меню», «Вид» и «Сведения», панель «Инструменты» и окно для рисования – холст.

Панель «Меню» содержит такие кнопки, как «Создать», «Открыть», «Сохранить». При нажатии на любую из кнопок сначала вызывается диалоговое окно для работы с файлами, а затем происходит обращение к методам класса Canvas, поскольку вся работа с содержимым холста лежит на нем. В методы в качестве параметра передается путь к файлу, сформированный пользователем в диалоговом окне.

Класс содержит кнопки, которые вызывают методы настройки кисти, установки текущего инструмента – в которых также вызываются метода класса Canvas.

UML-диаграмма класса представлена в приложении Д

4.4 Описание класса Canvas

4.4.1 О классе

Класс Canvas отвечает за холст: рисование на нем, очистку, хранит информацию о всех объектах, которые на нем находятся.

Canvas содержит методы для настройки размера, цвета и штриха кисти; установки выбранного инструмента в данный момент.

К Canvas прикреплены три MouseEventListener’a – обработчика событий мыши – MousePressEvent, MouseMoveEvent, MouseReleaseEvent. Они срабатывают при нажатии ЛКМ, перемещении мыши и отпускании ЛКМ соответственно.

UML-диаграмма класса представлена в приложении Е

4.4.2 Реализация функции «Сохранить»

Функция на вход получает значение типа QString, которое является полным именем сохраняемого файла.

Затем определяем значения высоты и ширины сохраняемой картинки. Значение ширины и высоты определяется не размером холста, а размером заполненной на нем области. Т.е. если холст заполнен не полностью, а лишь, допустим, квадратная область размером 100x100 пикселей, то сохраненная картинка будет иметь именно такой размер.

Область холста сохраняется в переменную типа QImage с моделью ARGB32 – трехканальной цветовой моделью RGB, дополненной четвертым альфа-каналом.

Затем QImage-переменная сохраняется по указанному пути.

4.4.2 Реализация функции «Открыть»

Функция на вход получает значение типа QString, которое является полным именем открываемого файла.

Устанавливается offset – количество пикселей, на которое предполагается смещение изображения при позиционировании относительно других изображений – в центр изображения.

Само изображение размещается на холсте в точке (0, 0).

4.5 Описание класса Figure

4.5.1 О классе

Класс Figure предназначен для рисования геометрических примитивов и задания заднего фона.

Для рисования предоставлены следующие примитивы: прямая, четырехугольник, эллипс, равнобедренный треугольник, прямоугольный треугольник, ромб, трапеция, пятиугольник.

Так как класс является наследником класса QGraphicsItem, то в будущем можно будет добавить возможность перемещения и изменения объектов, находящихся на холсте.

4.5.2 О формулах

В каждый момент времени нахождения курсора мыши на холсте мы можем определить его текущее положение, получив значение по оси абсцисс и по оси ординат – или проще говоря – пару точек (x, y). Таким образом, у нас имеется одна точка.

Узнать начальное положение курсора тоже довольно нетрудно – необходимо записать координаты курсора в момент нажатия кнопки мыши. Логично, что точка, в которой пользователь нажал на холсте, будет являться началом, из которого будет происходить рисование фигуры.

Конечное положение курсора узнается в момент отжатия кнопки мыши.

Построить прямую по двум точкам просто – достаточно указать начальные и конечные координаты.

А вот при построении фигур, требующих большее количество точек, появляются некоторые осложнения.

Для того чтобы нарисовать четырехугольник нам нужны две пары попарно параллельных и перпендикулярных прямых. Тогда из имеющихся пар

(x_1, y_1) и (x_2, y_2) нам необходимо составить следующие пары: (x_1, y_1) , (x_2, y_1), (x_2, y_2), (x_1, y_2). А затем между двумя соседними парами провести прямые.

Построение прямоугольного треугольника происходит по трем точкам и исключает из вышеуказанного набора пару точек (x_2, y_2). Порядок следования оставшихся точек сохраняется.

Чтобы построить равнобедренный треугольник комбинированием имеющихся «иксов» и «игреков» не обойтись, поскольку вершина по оси Ox не лежит ни на x_1, ни на x_2.

Начнем с построения стороны, где нам все известно – основание. Оно строится по точкам (x_1, y_2) и (x_2, y_2).

Из курса школьной математики нам известно, что медиана, проведенная из вершины равнобедренного треугольника к его основанию, делит это основание пополам. Т.е. вершина находится над серединой основания. Тогда середину высчитаем следующим образом:

$$\frac{x_1+x_2}{2} \ (1).$$

Таким образом, для построения равнобедренного треугольника необходимо провести следующие прямые:

Первая прямая с точками: (x_1, y_2), ($\frac{x_1+x_2}{2}$, y_1);

Вторая прямая с точками: (x_2, y_2), ($\frac{x_1+x_2}{2}$, y_1);

Третья прямая с точками: (x_1, y_2), (x_2, y_2).

Для построения трапеции требуется четыре точки. Точки для проведения нижнего основания нам известны – (x_1, y_2) и (x_2, y_2). Верхнее основание параллельно нижнему, но оно его короче.

Поэтому для определения точек оси Ox мы введем некоторый коэффициент, который будет сужать верхнее основание. Чем больше коэффициент, тем короче прямая. При коэффициенте равном 0, длина верхней прямой прямой будет равна длине нижней.

Тогда определение длины отступа от x_1 и x_2 будем определять прибавляя и отнимание соответственно некоторое число, определенное по следующей формуле: len = (abs(x1)+abs(x2)) * coeff;

$$len = (|x_1| \pm |x_2|) * coeff \ (2),$$

где len – число, прибавляемое или отнимаемое от координаты оси Ox; coeff – некоторый коэффициент; значение, заданное в приложении – 0,25. Между x_1 и x_2 ставится знак плюс, если они находятся по разные стороны оси Oy и знак минус – в противном случае.

У ромба каждая его вершина является центром воображаемого четырехугольника, построенного по точкам (x_1, y_1) и (x_2, y_2).

Тогда для определения середины линия составим следующую формулу:

$$cW = \frac{|x_1| \pm |x_2|}{2} + x_1 \ (3),$$

где cW (center width) – значение середины ширины по Ox. Знак суммы или разности определяется также, как и в (2).

Для cH (center height) – значения середины ширины по Oy – формула определяется также, но с заменой «иксов» на «игреки».

Основание пятиугольника короче, прямоугольной области, в пределах которой он строится, а вершина является серединой верхней стороны этой области.

Тогда для определения вершины пятиугольника нам понадобится значение из (3). Значение сдвига по Ox для основания и прямых по Oy, исходящих из вершины пятиугольника, определяется по (2), со значениями коэффициента 0,25 и 0,35 соответственно. Обозначим длины отступов как shift и uShift соответственно.

Таким образом, для построения пятиугольника необходимо провести следующие прямые:

Первая прямая с точками: (x_1 + shift, y_2), (x_2 - shift, y_2);

Вторая прямая с точками: (x_2 - shift, y_2), (x_2, y_1 + uShift);

Третья прямая с точками: (x_2, y_1 + uShift), (cW, y_1);

Четвертая прямая с точками: (cW, y_1), (x_1, y_1 + uShift);

Пятая прямая с точками: (x_1, y_1 + uShift), (x_1 + shift, y_2).

4.5.3 Возникшие ошибки

Изначально, при выводе формул рассматривалось, что точка (x_1, y_1) будет являться точкой верхнего левого угла прямоугольника, в рамках которого происходит построение фигуры, а точка (x_2, y_2) – нижнего правого.

Однако пользователю об этом ничего не известно, и он может рисовать фигуры в любую сторону.

Когда x_2 становится больше x_1 и/или y_2 становится больше y_1, тогда прямые некоторых фигур начинают путаться и пересекаться.

Данный недочет был решен следующим образом: перед построением фигуры сравнивались значения начальной и конечной точек по осям Ox и Oy. Если значение x_2 или y_2 было больше, чем значение x_1 или y_1 соответственно, то происходил обмен значений между этими двумя переменными.

4.5.4 Реализация класса

Как видно на диаграмме класса (рис. 4.2) на вход в конструктор поступают следующие значения:

x_1, y_1, x_2, y_2 – начальные конечные значения области рисования фигуры. Имеют тип qreal, являющийся просто переопределением типа double.

chosenFigure – целочисленная переменная, которая в методе отрисовки определяет то, какая фигура будет нарисована.

pen – переменная типа QPen. Она содержит в себе всю информацию о кисти, которой будет рисоваться фигура: размер, цвет, стиль линии (штриховка).

gradient – булева переменная, определяющая будет ли фигура иметь градиентый контур и/или заливку.

fillPath – булева переменная, определяющая будет ли заливка у нарисованной фигуры.

linearGrad – переменная типа QLinearGradient. Определяет стиль линейного градиента.

Описание переменных shift, uShift, cW, cH см. в п. 4.5.2 отчета.

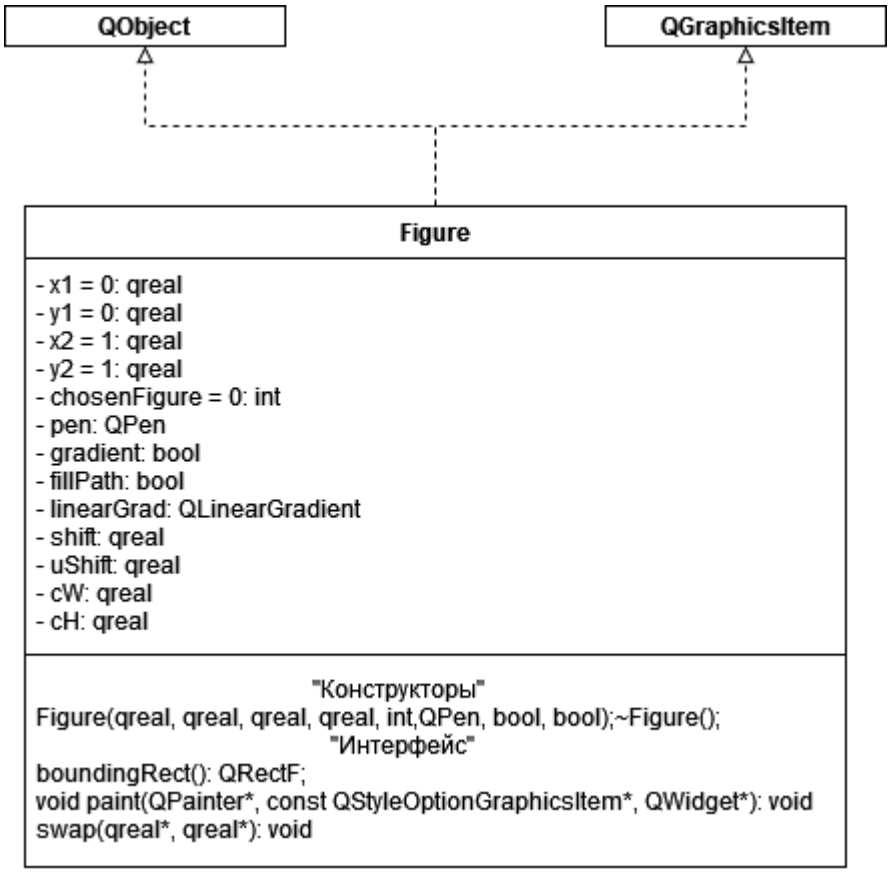


Рисунок 4.2 – Диаграмма класса Figure

Метод boundingRect() определяет область фигуры, в которой она рисуется. При отрисовке фигуры на холсте обновляется именно та область, которая задана этим методом.

4.6 Описание класса Manual

Класс Manual содержит в себе обозреватель текста, который отображает HTML-документ, представляющий из себя руководство пользователя. Руководство содержит в себе описание функциональной составляющей редактора и контакты разработчиков.

UML-диаграмма класса представлена в приложении Ж.

4.7 Описание класса Feedback

Класс Feedback предназначен для поддержки обратной связи. Сообщения в данный момент просто сохраняются в txt-файл, но т.к. в Руководстве пользователя есть почта разработчиков, пользователь всегда может отправить свой отзыв на нее.

UML-диаграмма класса представлена в приложении И.

5 Описание команды разработчиков и их функций в проекте

Разработка программного обеспечения в команде позволяет людям создавать большие проекты, которые одному человеку создать непосильно, или разработка которых будет занимать невероятно большое количество времени, по истечении которого проект скорее всего будет уже не актуален.

Наша команда состоит из трех человек.

Куминов Павел Алексеевич, студент каф. АСУ, гр. 430-4. Занимался проектированием приложения, разработкой класса Figure, реализацией функций манипулирования с файлами.

Завзятов Владислав Сергеевич, студент каф. АСУ, гр. 430-4. Занимался разработкой классов MainWindow, Manual, Feedback.

Лузинсан Анастасия Александровна, студентка каф. АСУ, гр. 430-2. Занималась разработкой класса Canvas. Проводила маркетинговые исследования.

6 Тестирование

6.1 Тестирование инструмента «Фигуры»

6.1.1 Тест №1

Цель теста: убедиться в правильности рисования фигур с различными размерами и цветами кисти.

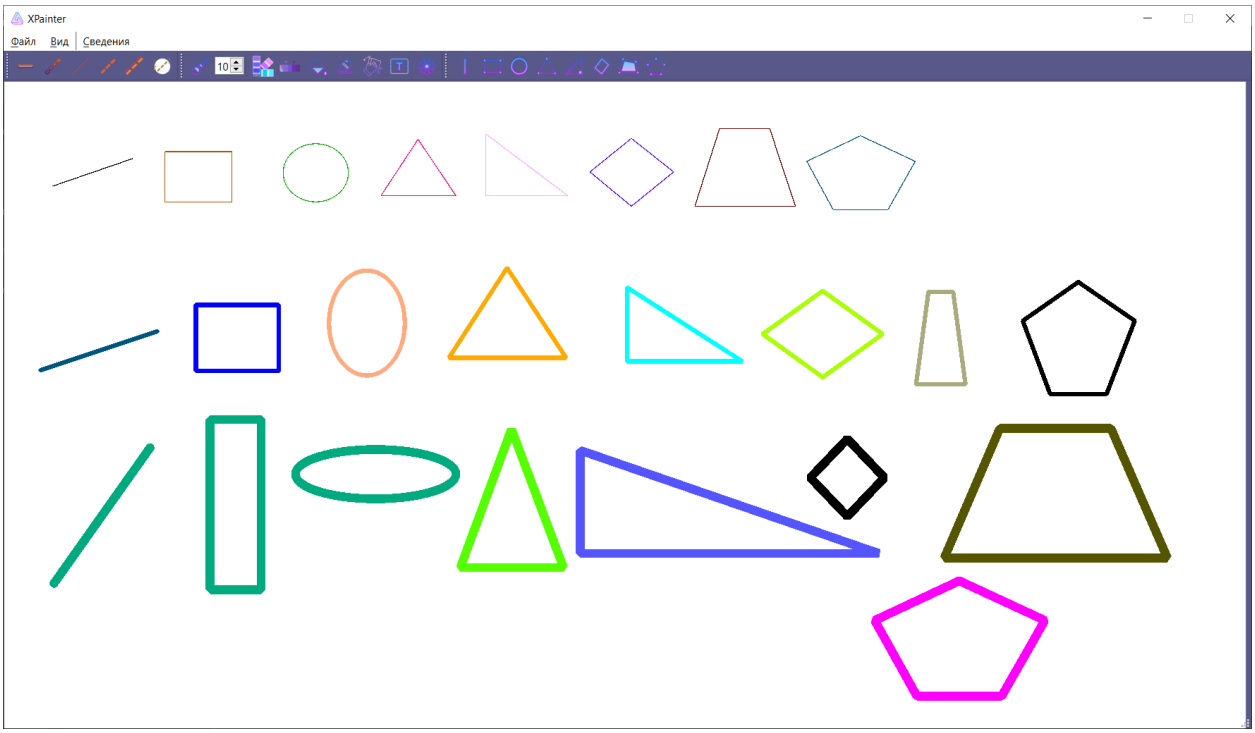


Рисунок 6.1 – Результат теста №1

6.1.2 Тест №2

Цель теста: убедиться в правильности рисования фигур с использованием инструмента «заполнение».

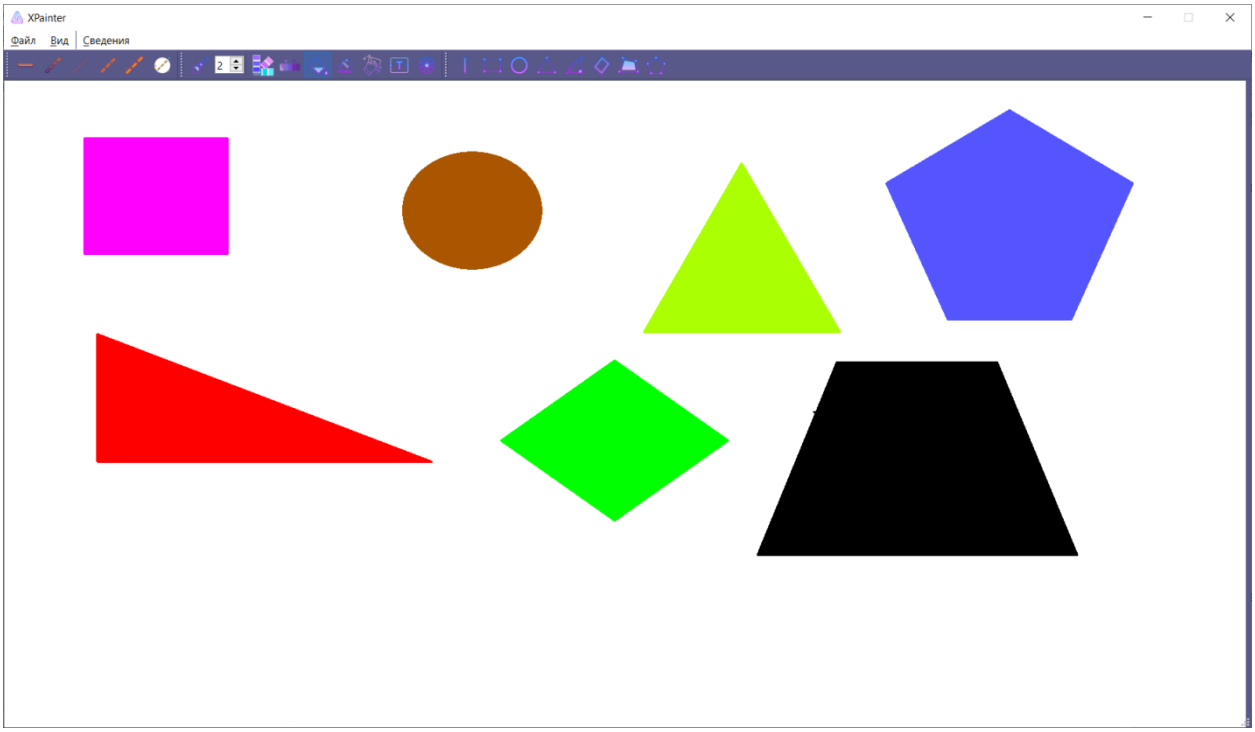


Рисунок 6.2 – Результат теста №2

6.2 Тестирование инструмента «Заливка»

Цель теста: убедиться в работоспособности инструмента «Задний фон».

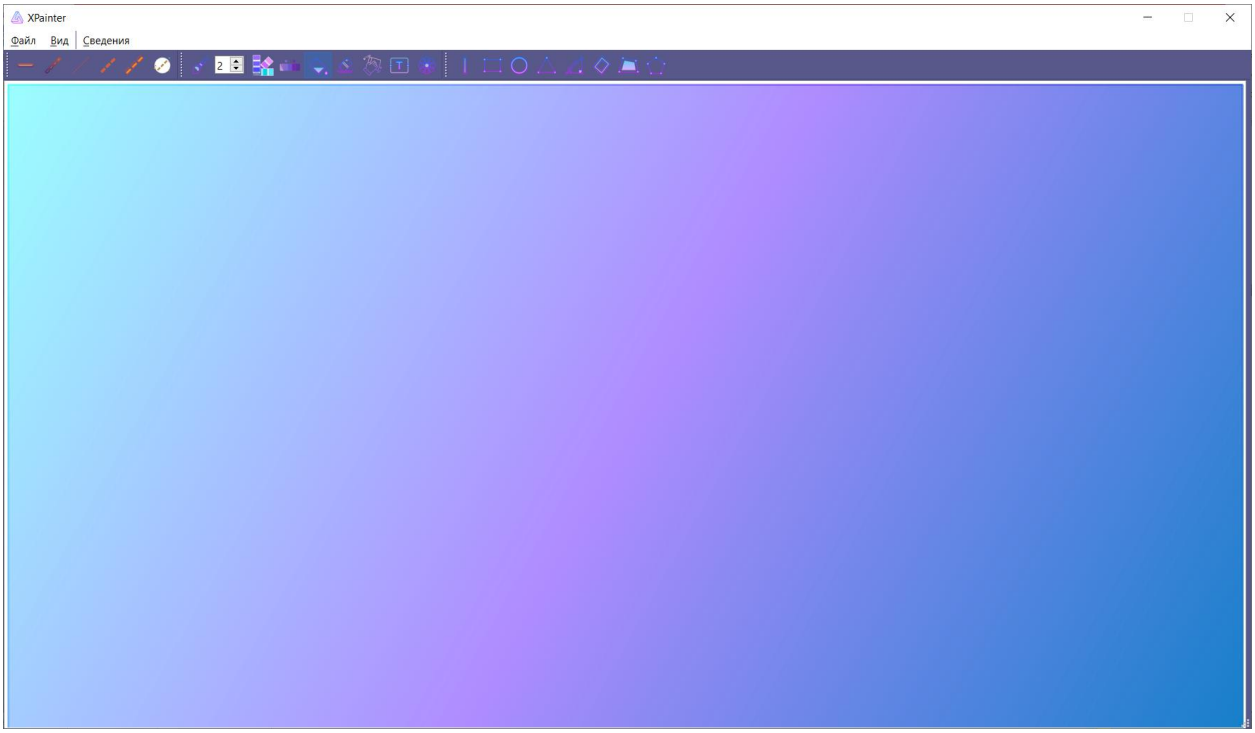


Рисунок 6.3 – Результат теста №3

6.3 Тестирование функции сохранения файла

Цель теста: убедиться в работоспособности функции сохранения файла.

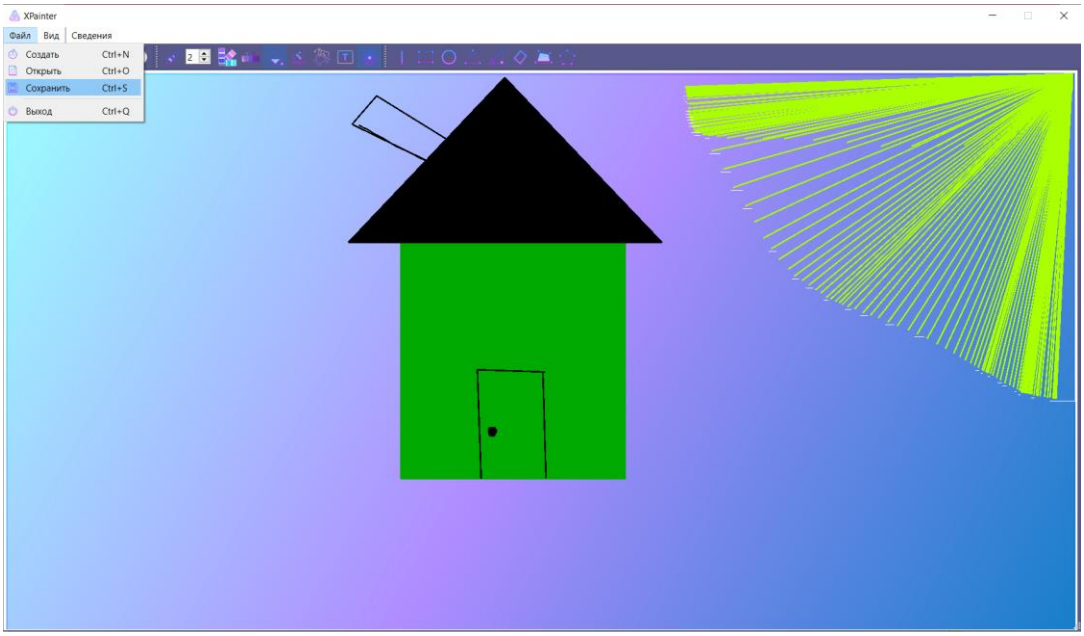


Рисунок 6.4 – Картинка, нарисованная в XPainter

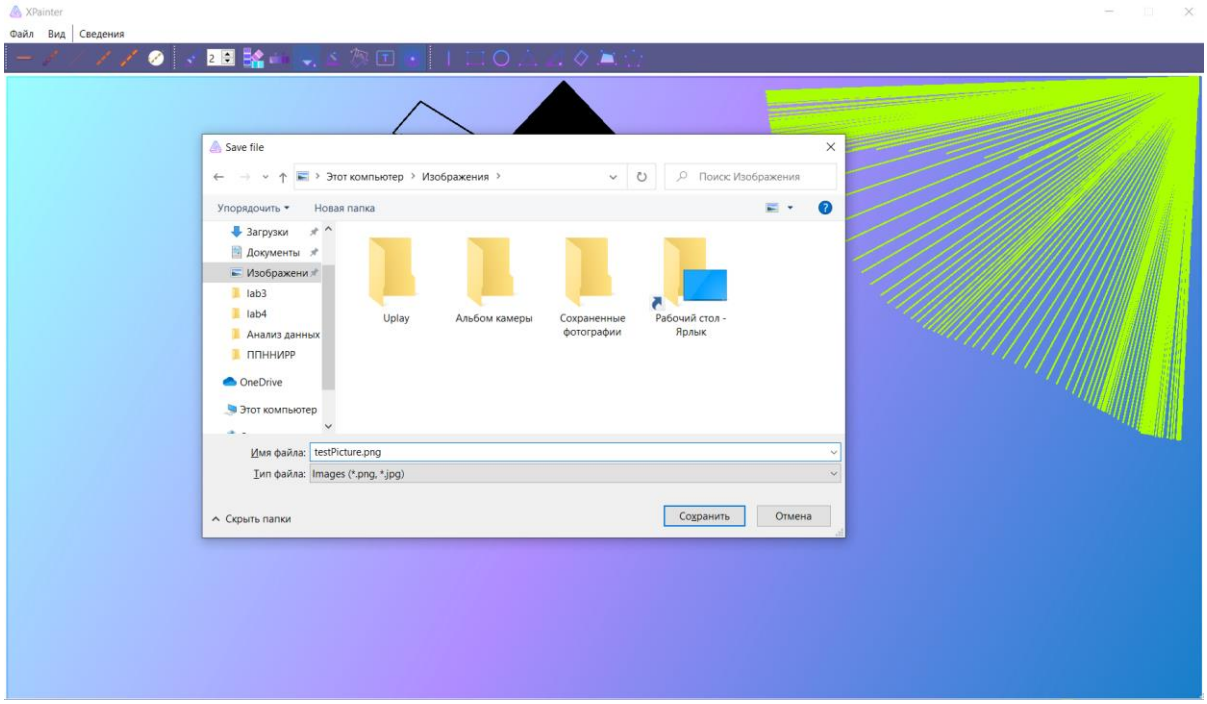


Рисунок 6.5 – Диалоговое окно для сохранения картинки

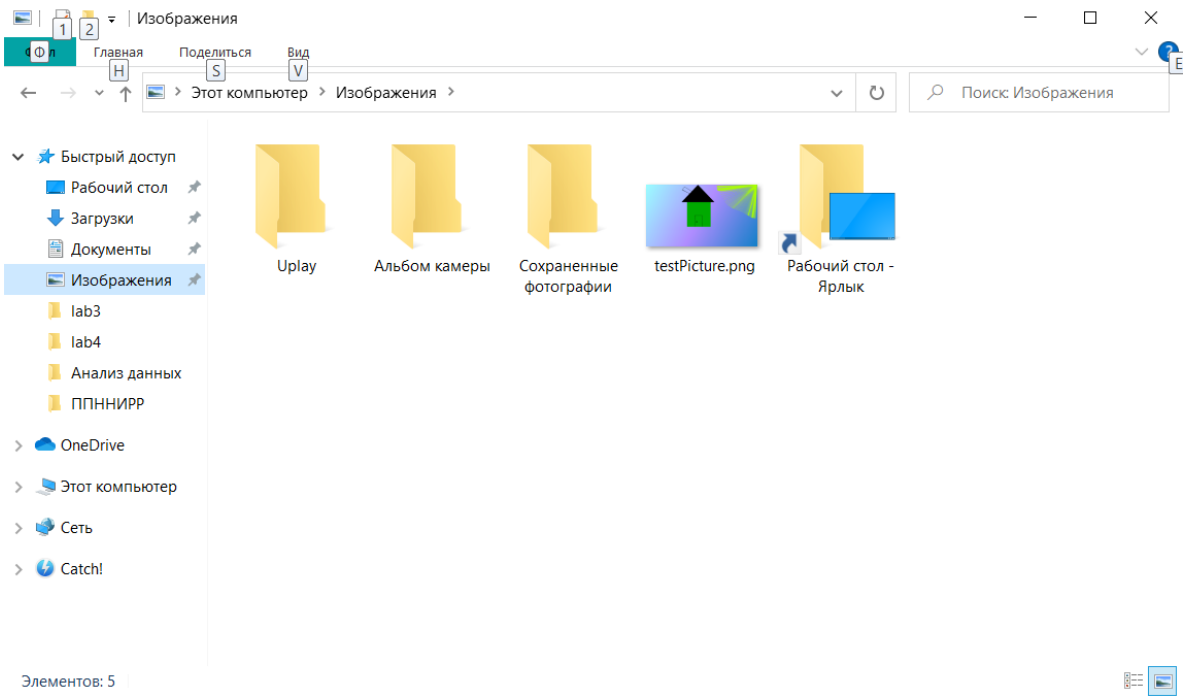


Рисунок 6.6 – Папка с сохраненной картинкой

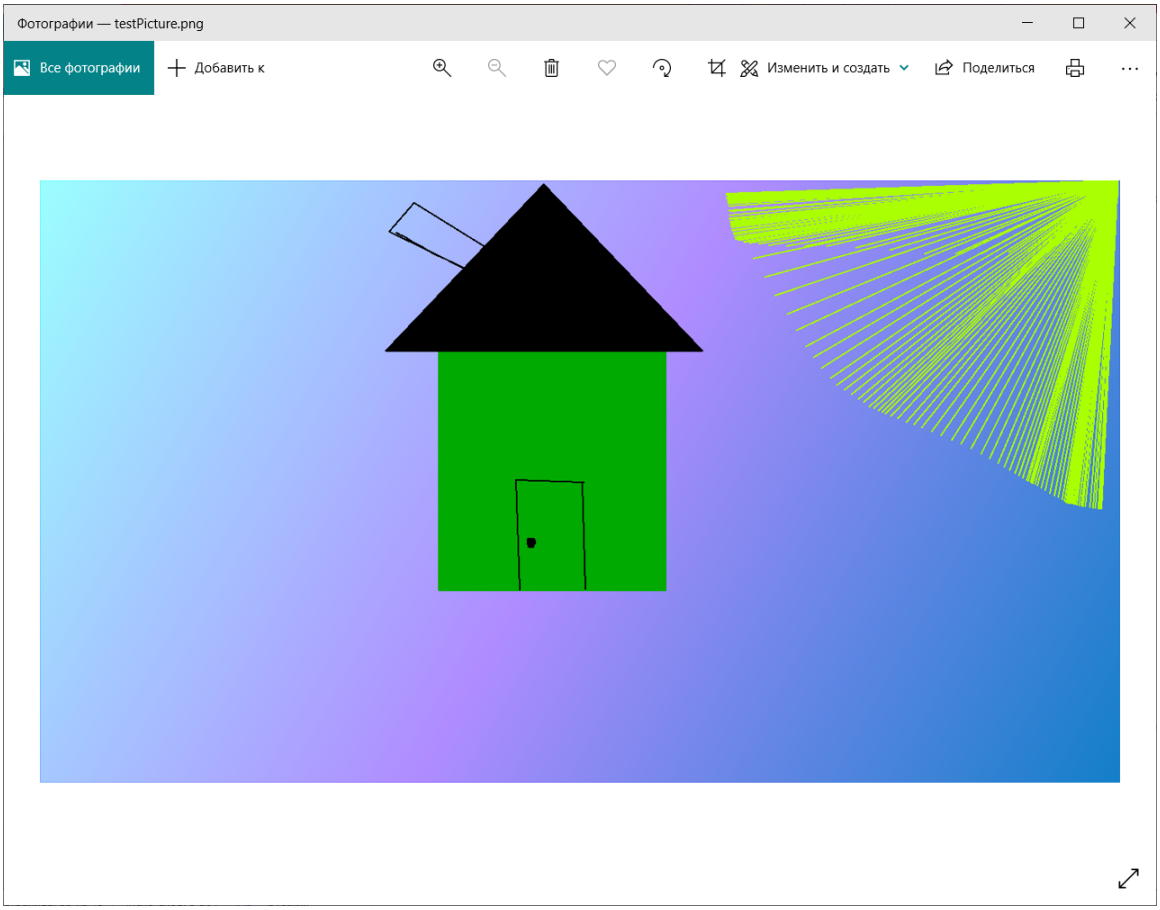


Рисунок 6.7 – Картинка, открытая в стандартном обозревателе фотографий ОС Windows 10

6.4 Тестирование функции открытия файла

Цель теста: убедиться в работоспособности функции открытия файла.

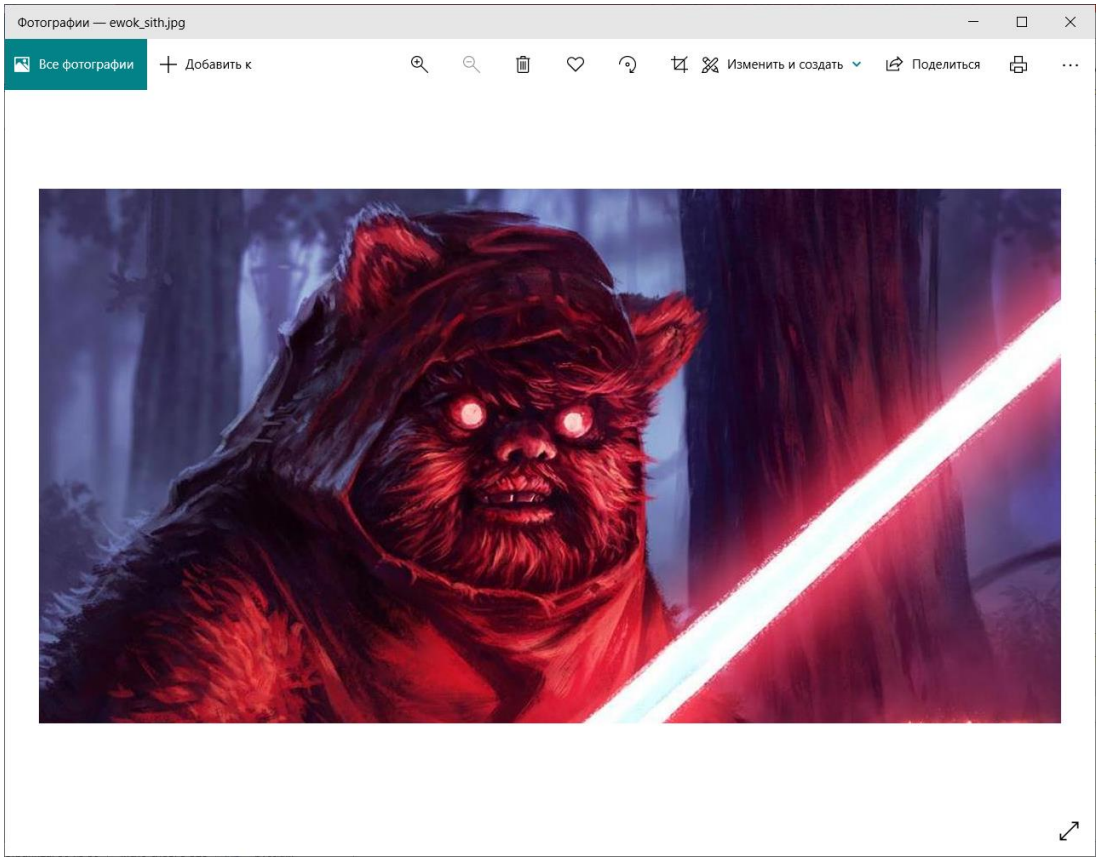


Рисунок 6.8 – Вид картинки в обозревателе фотографий

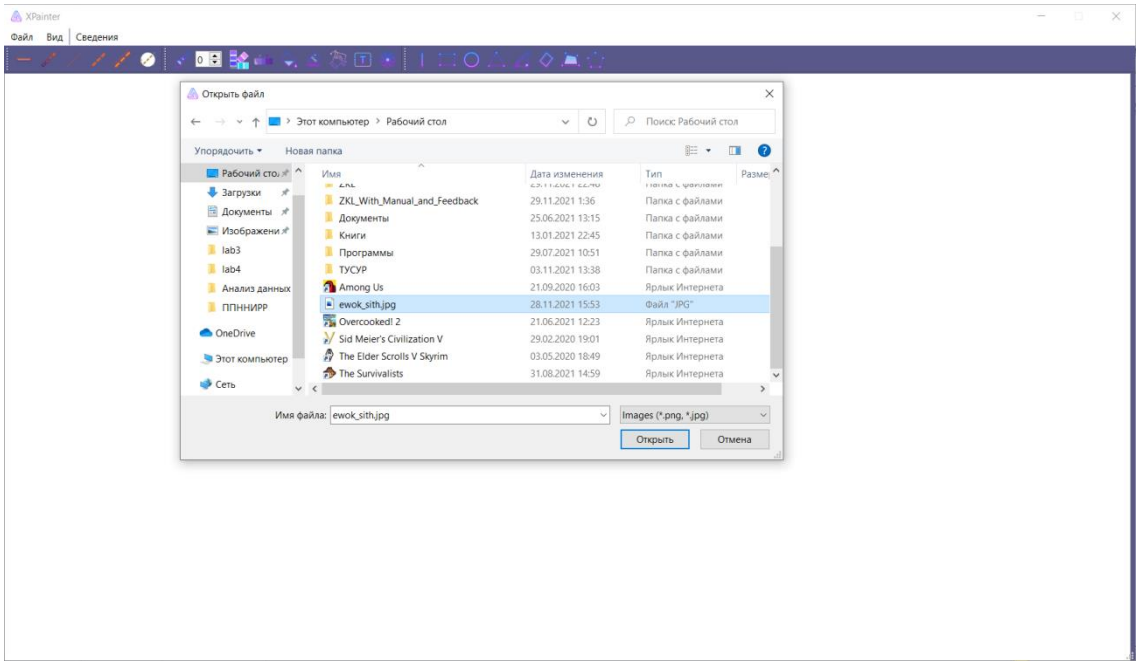


Рисунок 6.9 – Диалоговое окно для открытия файла

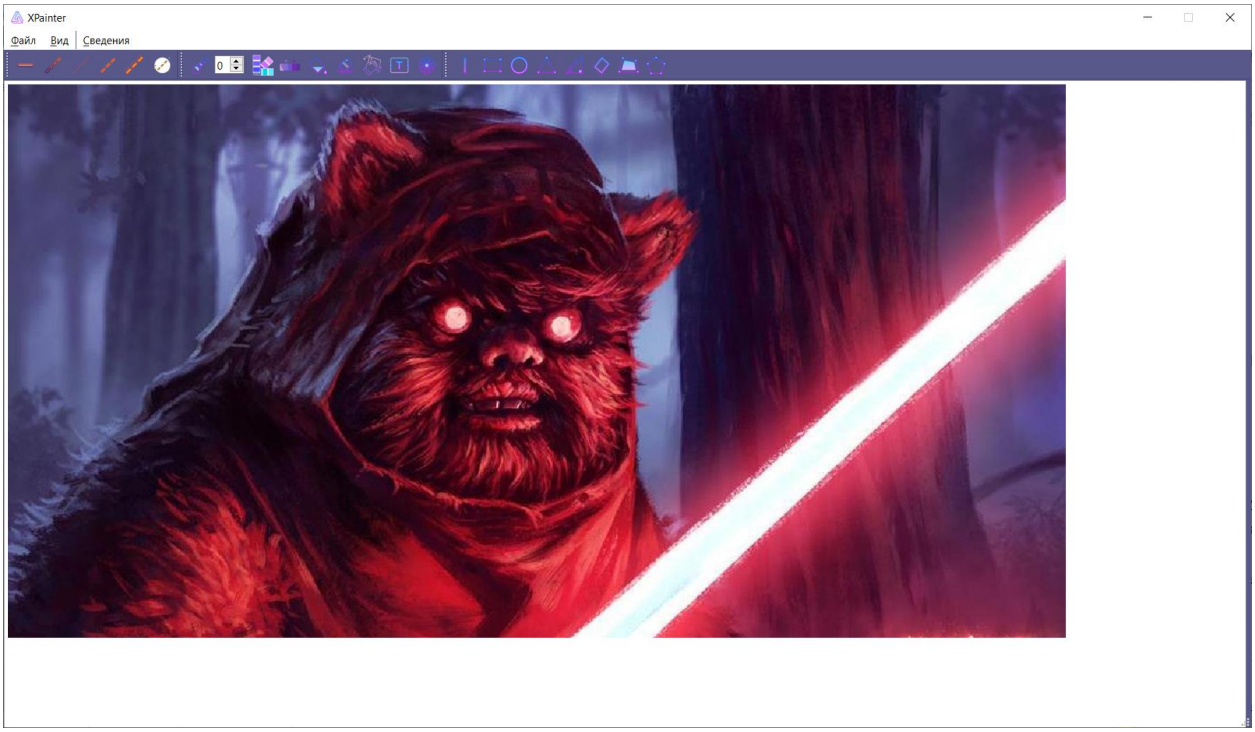


Рисунок 6.10 – Картинка, открытая в редакторе

Заключение

В ходе прохождения этой практики я получил навыки командной разработки приложений. Научился распределять задачи между членами команды. Получил навыки работы с GitHub, позволившим удобно проводить слияние и контроль версий приложения.

Провел декомпозицию предметной области. Изучил UML-нотации классов и научился строить диаграммы.

Изучил основы фреймворка Qt и некоторые классы и методы, работающие с графикой.

Я провел анализ места графических редакторов в современном мире и пришел к выводу, что современная компьютерная графика – это достаточно сложная, основательно проработанная и разнообразная научно-техническая дисциплина. Некоторые ее разделы, такие как геометрические преобразования, способы описания кривых и поверхностей, к настоящему времени уже исследованы достаточно полно. Ряд областей продолжает активно развиваться: методы растрового сканирования, удаление невидимых линий и поверхностей, моделирование цвета и освещенности, текстурирование, создание эффекта прозрачности и полупрозрачности и др.

В наше время компьютерная графика – как двумерная, так и трехмерная – используется повсеместно. Она используется в различных отраслях для визуализации и отображения накопленной информации в виде графиков и диаграмм.

В проектировании активно используются САПР для разработки схем, чертежей и инженерных расчетов.

В медицине в настоящее время широко используются методы диагностики, использующие компьютерную визуализацию внутренних органов человека. Томография (в частности, ультразвуковое исследование) позволяет получить трехмерную информацию, которая затем подвергается математиче-

ской обработке и выводится на экран. Помимо этого применяется и двумерная графика: энцефалограммы, миограммы, выводимые на экран компьютера или графопостроитель.

Графика используется и для имитации различного рода ситуаций, возникающих, например, при полете самолета или космического аппарата, движении автомобиля.

Из этого можно сделать вывод, что системы позволяющие работать с компьютерной графикой очень важны и специалист, разбирающийся в вопросах конструирования и разработки таких систем будет востребован на рынке.

При разработке нашего графического редактора мы рассмотрели самые простые вопросы при работе с графикой, но этот проект может стать отправной точкой для более глубокого изучения подобных систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. НОУ ИНТУИТ | Лекция | Общее введение в компьютерную графику. [Электронный ресурс]. Режим доступа: https://intuit.ru/studies/courses/70/70/lecture/2092

2. Анализ и исследование графических редакторов. [Электронный ресурс]. Режим доступа: https://www.stud24.ru/programming-computer/analiz-i-issledovanie-graficheskih-redaktorov/380651-1211331-page1.html

3. Графический редактор — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Графический_редактор

4. Растровый графический редактор — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Растровый_графический_редактор

5. Векторный графический редактор — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Векторный_графический_редактор

6. Графический редактор. – Информатика. [Электронный ресурс]. Режим доступа: http://psk68.ru/files/metod/uchebnik_Informatika/graf.html

7. C++ — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/C++

8. Qt — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Qt

9. Qt Creator — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Qt_Creator

10. Qt Documentation | Home. [Электронный ресурс]. Режим доступа: https://doc.qt.io/

11. Уроки программирования на языке C++ | Ravesli. [Электронный ресурс]. Режим доступа: https://ravesli.com/uroki-cpp/

12. Qt 5.3. Профессиональное программирование на C++. / М. Шлее – СПб.: БХВ-Петербург, 2015. – 928 с.: ил. – (в подлиннике).

13. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е изд., пер. с англ. / Д. Арлоу, И. Нейштадт – СПб: Символ Плюс, 2007. – 624 с., ил.

14. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд., пер. с англ. / Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон – М.: ООО “И.Д, Вильямс”, 2008. – 720 с.

15. Diagram Software and Flowchart Maker. [Электронный ресурс]. Режим доступа: https://www.diagrams.net/

16. GitHub — Википедия. [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/GitHub

17. Features | GitHub. [Электронный ресурс]. Режим доступа: https://github.com/features

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

(ОБЯЗАТЕЛЬНОЕ)

ЛИСТИНГ ЗАГОЛОВОЧНОГО ФАЙЛА КЛАССА FIGURE

```
#ifndef FIGURE_H
#define FIGURE_H
#include <QObject>
#include <QGraphicsItem>
#include <QPen>
#include <QRectF>
#include <QPainter>
#include <QStyleOptionGraphicsItem>
#include <QGraphicsSceneMouseEvent>

class Figure :  public QObject, public QGraphicsItem
{
   Q_OBJECT
public:
    Figure(qreal x1=0, qreal y1=0, qreal x2=0, qreal y2=0, int chosenFigure=0,QPen pen = QPen(Qt::black), bool gradient = false, bool fillPath = false);
    Q_INTERFACES(QGraphicsItem);
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
   ~Figure();
private:
   qreal x1=0,y1=0,x2=1,y2=1; int chosenFigure = 0;
   QPen pen; bool gradient, fillPath; QLinearGradient linearGrad;
   void swap(qreal *a, qreal *b);
   };
#endif // FIGURE_H
```

ПРИЛОЖЕНИЕ Б

(ОБЯЗАТЕЛЬНОЕ)

ЛИСТИНГ ФАЙЛА С РЕАЛИЗАЦИЕЙ КЛАССА FIGURE

```
#include "figure.h"
#include <QLinearGradient>
#include <QBrush>
#include <iostream>

Figure::Figure(qreal _x1, qreal _y1, qreal _x2, qreal _y2, int _chosenFigure, QPen _pen, bool _gradient, bool _fillPath)
   : x1{_x1}, y1{_y1}, x2{_x2}, y2{_y2}, chosenFigure{_chosenFigure}, pen{_pen}, gradient{_gradient}, fillPath{_fillPath} {}
Figure::~Figure(){}

QRectF Figure::boundingRect() const
{
   qreal penWidth = this->pen.widthF()+3;
   QRectF rect(x1, y1, x2-x1, y2-y1);
   if(rect.width()>0 && rect.height()>0)
      rect.setRect(x1-penWidth,
                   y1-penWidth,
                   x2-x1+2*penWidth,
                   y2-y1+2*penWidth);
   else if(rect.width()>0)
      rect.adjust(-penWidth, +penWidth, penWidth, -penWidth );
   else if(rect.height()>0)
      rect.adjust(penWidth, -penWidth, -penWidth, penWidth );
   else
      rect.adjust(penWidth, penWidth, -penWidth, -penWidth );
   return  rect.normalized();
}
```

```
void Figure::swap(qreal *a, qreal *b)
{
   qreal temp = *a;
   *a = *b;
   *b = temp;
}

void Figure::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
{
   Q_UNUSED(option);
   Q_UNUSED(widget);

   if(gradient)
   {
      linearGrad = QLinearGradient(x1,y1,x2,y2);
      linearGrad.setColorAt(0, QColor(0,255,255,100));
      linearGrad.setColorAt(0.5, QColor(120,60,255,150));
      linearGrad.setColorAt(1, QColor(17,127,200,255));
      linearGrad.setSpread(QGradient::ReflectSpread);
   }

   if(fillPath)
      if(gradient)
      {
         painter->setBrush(linearGrad);
        // painter->setPen(Qt::black);
         painter->setPen(QPen(QBrush(linearGrad), pen.width(), pen.style(), pen.capStyle() ));
      }
      else
      {
         painter->setBrush(pen.brush());
         painter->setPen(pen);
```

```
    }
  else
    if(gradient) painter->setPen(QPen(QBrush(linearGrad), pen.width(), pen.style(),
pen.capStyle() ));
    else painter->setPen(pen);

  switch(chosenFigure)
  {
  case 1:// линия
    painter->drawLine(x1, y1, x2, y2);
    break;
  case 2: // прямоугольник
    painter->drawRect(x1, y1, x2-x1, y2-y1);
    //painter->fillRect(rect, pen.color());
    break;
  case 3: // круг/окружность
    painter->drawEllipse(x1, y1, x2-x1, y2-y1);
    break;
  case 4:// треугольник
  {
    QPointF points[3] = {QPoint(x1,y2), QPoint((x1 + x2) / 2, y1), QPoint(x2, y2)};
    painter->drawPolygon(points, 3);
    break;
  }
  case 5:// прямоугольный треугольник
  {
    QPointF points[3] = {QPoint(x1, y1), QPoint(x1, y2), QPoint(x2, y2)};
    painter->drawPolygon(points, 3);
    break;
  }
  case 6:// ромб
  {
    if (x2 < x1 && y2 < y1)
    {
```

```
            swap(&x1, &x2);
            swap(&y1, &y2);
        }
        else if (x2 < x1)
            swap(&x1, &x2);
        else if (y2 < y1)
            swap(&y2, &y1);

        qreal cW, cH;
        if ((x1 >= 0  && x2 <= 0) || (x1 <= 0 && x2 >= 0))
            cW = abs((abs(x1) + abs(x2))) / 2 + x1;
        else
            cW = abs((abs(x2) - abs(x1))) / 2 + x1;
        if ((y1 >= 0  && y2 <= 0) || (y1 <= 0 && y2 >= 0))
            cH = abs((abs(y1) + abs(y2))) / 2 + y1;
        else
            cH = abs((abs(y2) - abs(y1))) / 2 + y1;

        QPointF points[4] = {QPoint(x1, cH), QPoint(cW, y1), QPoint(x2, cH), QPoint(cW, y2)};
        painter->drawPolygon(points, 4);
        break;
    }
    case 7: // трапеция
    {
        qreal len, coeff = 0.25;
        if ((x1 >= 0  && x2 <= 0) || (x1 <= 0 && x2 >= 0))
            len = (abs(x1)+abs(x2)) * coeff;
        else
            len = (abs(x2) - abs(x1)) * coeff;

        QPointF points[4] = {QPoint(x1, y2), QPoint(x1 + len, y1), QPoint(x2 - len, y1),
QPoint(x2, y2)};
        painter->drawPolygon(points, 4);
        break;
```

```
    }
    case 8: // пятиугольник
    {
        if (x2 < x1 && y2 < y1)
        {
            swap(&x1, &x2);
            swap(&y1, &y2);
        }
        else if (x2 < x1)
            swap(&x1, &x2);
        else if (y2 < y1)
            swap(&y2, &y1);

        qreal shift, uShift, coeff = 0.25, uCoeff = 0.35, cW;

        if ((x1 >= 0  && x2 <= 0) || (x1 <= 0 && x2 >= 0))
            cW = abs((abs(x1) + abs(x2))) / 2 + x1;
        else
            cW = abs((abs(x2) - abs(x1))) / 2 + x1;
        if ((x1 >= 0  && x2 <= 0) || (x1 <= 0 && x2 >= 0))
            shift = ((abs(x1) + abs(x2))) * coeff;
        else shift = abs((abs(x2) - abs(x1))) * coeff;
        if ((y1 >= 0  && y2 <= 0) || (y1 <= 0 && y2 >= 0))
            uShift = abs((abs(y1) + abs(y2))) * uCoeff;
        else uShift = abs((abs(y2) - abs(y1))) * uCoeff;

        QPointF points[5] = {QPoint(x1 + shift, y2), QPoint(x2 - shift, y2), QPoint(x2, y1 +
uShift),
                             QPoint(cW, y1), QPoint(x1, y1 + uShift)};
        painter->drawPolygon(points, 5);
        break;
    }
    }
}
```

ПРИЛОЖЕНИЕ Г

(СПРАВОЧНОЕ)

ТАБЛИЦА ДОПОЛНЕНИЙ ВИДИМОСТИ В UML

Таблица Г.1

Дополнение	Видимость	Семантика
+	Public (Открытый)	Любой элемент, который имеет доступ к классу, имеет доступ к любой из его возможностей, видимость которой public.
-	Private (Закрытый)	Только операции класса имеют доступ к возможностям, имеющим видимость private.
#	Protected (Защищенный)	Только операции класса или потомка класса имеют доступ к возможностям, имеющим видимость protected.
~	Package (Пакетный)	Любой элемент, находящийся в одном пакете с классом или во вложенном подпакете, имеет доступ ко всем его возможностям, видимость которых package.

ПРИЛОЖЕНИЕ Д

(СПРАВОЧНОЕ)

UML-ДИАГРАММА КЛАССА MAINWINDOW

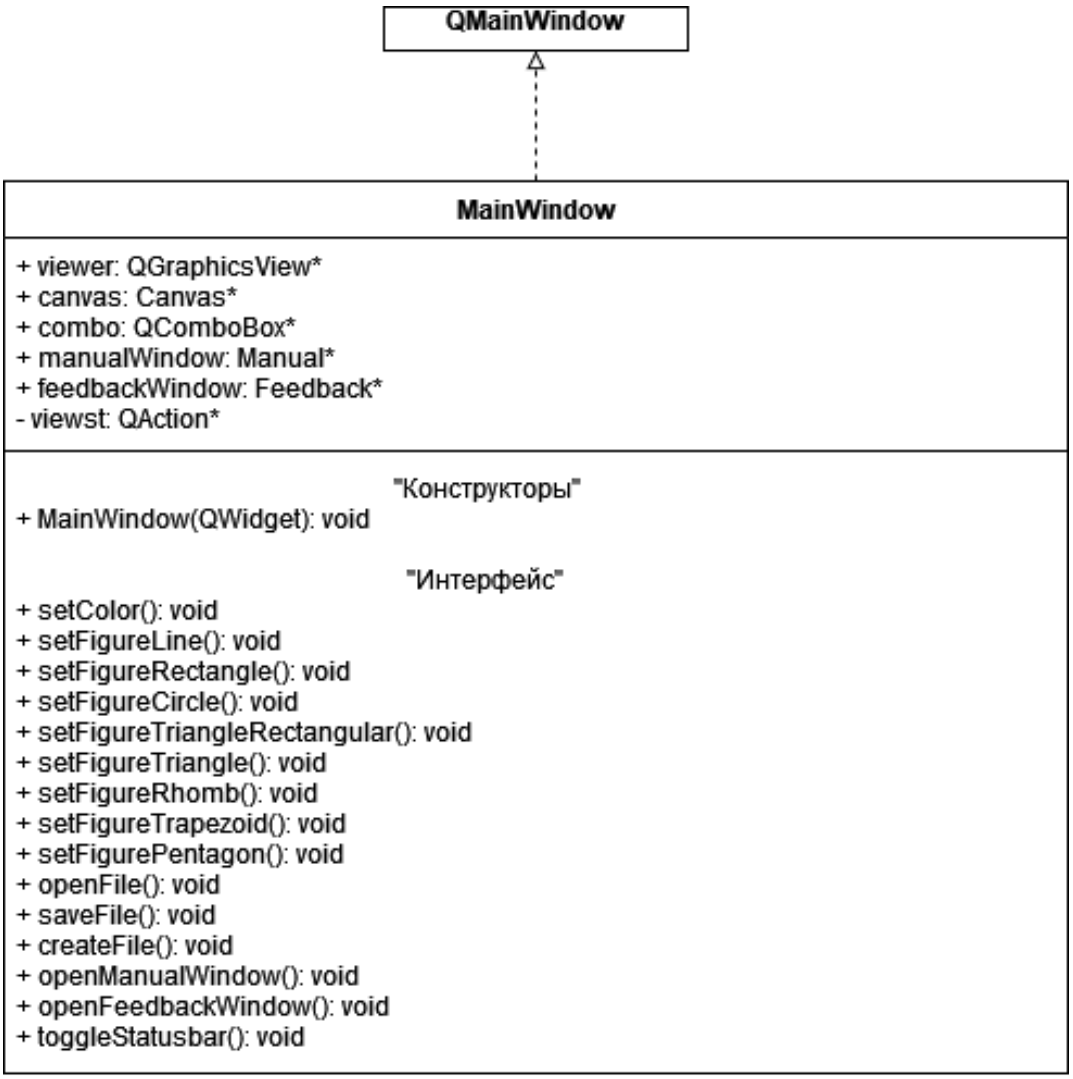


Рисунок Д.1

ПРИЛОЖЕНИЕ Е

(СПРАВОЧНОЕ)

UML-ДИАГРАММА КЛАССА CANVAS

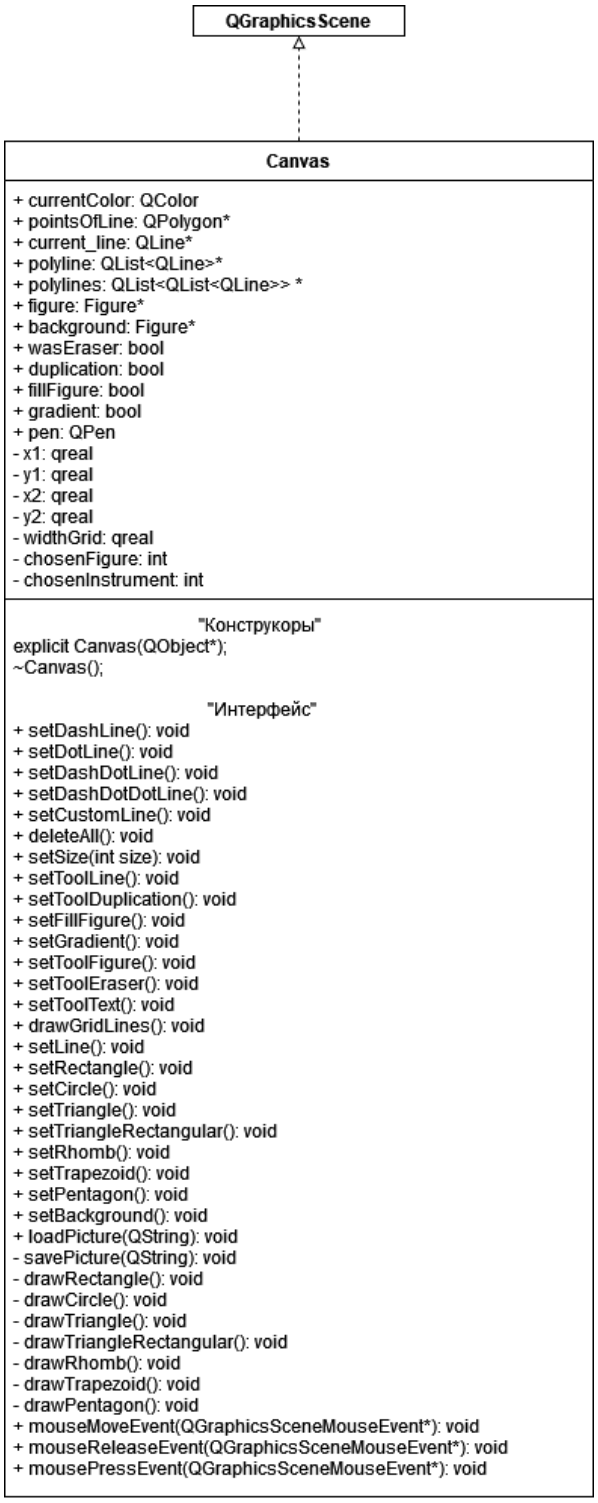


Рисунок Е.1

ПРИЛОЖЕНИЕ Ж

(СПРАВОЧНОЕ)

UML-ДИАГРАММА КЛАССА MANUAL

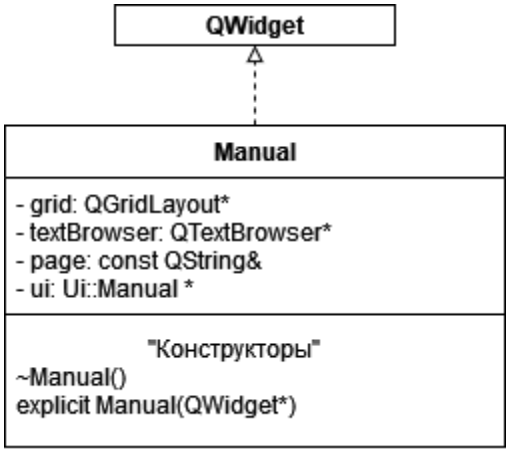


Рисунок Ж.1

ПРИЛОЖЕНИЕ И

(СПРАВОЧНОЕ)

UML-ДИАГРАММА КЛАССА FEEDBACK

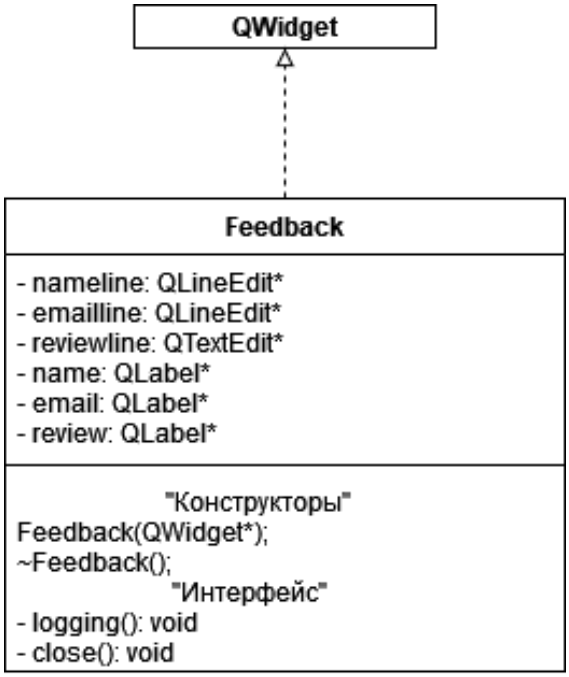


Рисунок И.1