

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка растрового графічного редактора»

Виконав студент групи ПЗ-19-1
ПОЛОНЕВИЧ Денис Вадимович

Керівник роботи:
ТОЛСТОЙ Ігор Анатолійович

Рецензент:
ЧИЖМОТРИЯ Олена Геннадіївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПОЛОНЕВИЧ Денис Вадимович**

Керівник роботи: **ТОЛСТОЙ Ігор Анатолійович**

Тема роботи: «Розробка растрового графічного редактора»,
затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: розробка растрового графічного редактора

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Толстой І.А.	16.02.2023р.	16.02.2023р.
2	Толстой І.А.	10.04.2023р.	10.04.2023р.
3	Толстой І.А.	01.05.2023р.	01.05.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	3 лютого 2023 – 14 лютого 2023	Виконано
2	Аналіз моделей аналогів	15 лютого 2023 – 28 лютого 2023	Виконано
3	Постановка технічного завдання	1 березня 2023 – 9 березня 2023	Виконано
4	Вивчення літературних джерел	10 березня 2023 – 14 березня 2023	Виконано
5	Проектування структури	15 березня 2023 – 24 березня 2023	Виконано
6	Написання програмного коду	25 березня 2023 – 17 квітня 2023	Виконано
7	Тестування роботи програми	18 квітня 2023 – 22 квітня 2023	Виконано
8	Аналіз одержаних результатів	23 квітня 2023 – 25 квітня 2023	Виконано
9	Оформлення пояснювальної записки	10 травня 2023 – 11 червня 2023	Виконано
10	Попередній захист	26 квітня 2023	Виконано

РЕФЕРАТ

Випускна робота бакалавра складається з комплексної програмної реалізації растрового графічного редактора, а також пояснювальної записки. Пояснювальна записка до дипломної роботи містить 61 сторінку, 33 ілюстрації, 5 таблиць, 4 додатки та 31 літературних джерел.

Основною метою даної роботи є розробка алгоритмів та реалізація програмного продукту, що надаватиме широкий спектр інструментів для малювання, маніпуляції зображеннями, створення спеціальних ефектів, колажів.

В роботі визначено ключові завдання, що пов'язані з розробкою редактора, проведено аналіз існуючого інструментарію, подібного до розроблюваного, а також наведено сценарії роботи програмного продукту. Представлено огляд загальної структури додатку, описано алгоритми взаємодії між окремими компонентами, а також наведено опис об'єктної структури (діаграму класів) додатку.

В рамках випускної роботи проводились експерименти по оптимізації графічних перетворень в результаті яких були визначені оптимальні налаштування для швидкого редагування зображень.

КЛЮЧОВІ СЛОВА: ЗОБРАЖЕННЯ, РАСТРОВИЙ ГРАФІЧНИЙ РЕДАКТОР, ПІКСЕЛ, ФОРМАТИ ГРАФІЧНИХ ФАЙЛІВ, КОМП'ЮТЕРНА ГРАФІКА.

					ІПЗ.КР.Б – 121 – 23 – ПЗ								
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка растрового графічного редактора				Літ.		Арк.	Аркушів	
Розроб.		Д.В. Полоневич										2	61
Керівник		І.А. Толстой							Житомирська політехніка, група ІПЗ-19-1				
Рецензент		О.Г. Чижморя											
Зав. каф.		А.В. Морозов			Пояснювальна записка								

ABSTRACT

The bachelor's thesis consists of a complex software implementation of a graphic editor, as well as an explanatory note. The explanatory note to the thesis contains 61 pages, 33 illustrations, 5 tables, 4 appendices, and 31 references.

The main purpose of this work is to develop algorithms and implement a software product that will provide a wide range of tools for drawing, image manipulation, creating special effects, collages, etc.

The paper identifies key tasks related to the development of the editor, analyzes existing tools similar to the one being developed, and provides scenarios for the software product. An overview of the general structure of the application is presented, the algorithms of interaction between individual components are described, and the object structure (class diagram) of the application is described.

As part of the graduation work, experiments were conducted to optimize graphic transformations, as a result of which the optimal settings for fast image editing were determined.

KEYWORDS: IMAGE, RASTER GRAPHIC EDITOR, PIXEL, GRAPHIC FILE FORMATS, COMPUTER GRAPHICS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	5
ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	8
1.1 Постановка задачі.....	8
1.2 Аналіз аналогів програмного продукту	10
1.3 Вибір архітектури растрового графічного редактора.....	16
1.4 Обґрунтування вибору інструментальних засобів та вимог до апаратного забезпечення	17
Висновки до розділу 1	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ РАСТРОВОГО ГРАФІЧНОГО РЕДАКТОРА	20
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	20
2.2 Опис структури даних з якими працює растровий графічний редактор	27
2.3 Проектування та реалізація алгоритмів роботи системи	28
2.4 Реалізація функціоналу растрового графічного редактора	35
Висновки до розділу 2	38
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З РАСТРОВИМ ГРАФІЧНИМ РЕДАКТОРОМ	39
3.1 Порядок встановлення та налаштування системи.....	39
3.2 Інтерфейс та порядок роботи.....	41
3.3 Тестування роботи растрового графічного редактора.....	55
Висновки до розділу 3	56
ВИСНОВКИ.....	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	59
ДОДАТКИ.....	62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення;

ОС – Операційна система;

RAM - Random Access Memory (Пам'ять з довільним доступом);

GUI – Graphic User Interface (Графічний інтерфейс користувача);

UML – Unified Modeling Language (Уніфікована мова моделювання);

GVF – Graphics View Framework;

BSP – Binary Space Partitioning (Метод бінарного розбиття простору);

ВСТУП

Актуальність теми. Комп'ютерна графіка відноситься до галузі обчислювальної науки, що займається створенням, обробкою і відображенням графічних зображень і відео за допомогою комп'ютерних систем. Вона включає в себе різні аспекти, включаючи створення 2D та 3D графіки, комп'ютерну анімацію, обробку зображень, віртуальну реальність, розширену реальність та інші.

Без комп'ютерної графіки не обходиться жодна сучасна мультимедійна програма. Робота над графікою становить до 90% робочого часу програмістських колективів, які випускають програми масового використання. Графічні редактори, у свою чергу, як раз і є інструментами, які дозволяють створювати та редагувати графіку, від малюнків та ілюстрацій до дизайну логотипів, брендів, елементів для вебсторінок, та багато іншого. Це і є причиною того, чому програмні забезпечення цього типу продовжують бути важливими і актуальними інструментами для багатьох людей, особливо для тих, хто працює в області дизайну, медіа, маркетингу, реклами та інших суміжних галузях [1], [2].

З широким розповсюдженням Інтернету та зростаючою залежністю від цифрових рішень зростає попит на графічні редактори, які можуть допомогти у вирішенні проблеми у найкоротший проміжок часу. Ці додатки пропонують цілий ряд переваг, включаючи оптимізацію графічних перетворень, зручний та неважкий у освоєнні інтерфейс, збереження та подальше конвертування файлів у різні графічні типи.

Як результат, вони дозволяють творчим професіоналам і любителям виразити свої ідеї та створити вражаючі візуальні роботи.

Метою даної кваліфікаційної роботи є розробка алгоритмів та реалізація растрового графічного редактора з використанням інструментарію розробки програмного забезпечення мовою програмування C++ - Qt. Створення такого редактора має на меті надавати широкий спектр інструментів для малювання,

маніпуляції зображеннями, створення спеціальних ефектів, колажів у найкоротший у застосуванні час [3].

Встановлена мета обумовлює наступні завдання:

- аналіз задач, пов'язаних з користуванням растровим графічним редактором;
- визначення архітектури, узагальненої структури, ключових компонентів додатку;
- обґрунтування та вибір засобів реалізації продукту;
- проектування структурних складових та алгоритмів роботи додатку;
- провадження програмної реалізації;

Об'єктом дослідження є процес користування графічним редактором, а саме растровим.

Предметом дослідження є використання інструментарію розробки програмного забезпечення мовою програмування C++ Qt для програмної реалізації растрового графічного редактора.

В роботі було використано методи об'єктно-орієнтованого програмування та дизайну інтерфейсів.

Розроблений редактор може бути використаний для навчання базовим навичкам користування графічними редакторами або ж просто для розваги.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Основною метою провадження растрового графічного редактора є надання користувачам інструментів для створення, редагування та обробки растрових зображень. Впровадження редактора дозволить користувачам створювати нові зображення з нуля, вибираючи розмір, роздільну здатність, колірну глибину та інші параметри; редагувати файли: зміна наявних растрових зображень шляхом видалення, переміщення окремих елементів, зміни розміру, масштабу, застосування фільтрів, додавання тексту [4].

В кваліфікаційній роботі необхідно розробити додаток для редагування растрових зображень, якомога різноманітнішими шляхами.

Для реалізації поставленого завдання, основними етапами є:

1. Аналіз проекту та збір вимог:
 - 1.1. проведення ретельного аналізу практики використання растрових графічних редакторів;
 - 1.2. визначення основних характеристик і функціональних можливостей редактора;
 - 1.3. аналіз аналогів програмного продукту;
2. Вибір та інтеграція технологій:
 - 2.1. дослідження та вибір платформи розробки;
 - 2.2. інтеграція стандартних інструментів та фреймворків, на яких буде працювати редактор;
 - 2.3. впровадження розгортання програмного продукту на будь-яку Windows-машину;
3. Розробка та впровадження системи:
 - 3.1. розробка інтуїтивно зрозумілого та зручного інтерфейсу, який включає різні інструменти малювання, панелі інструментів, меню,

полотно для відображення зображення та інші елементи інтерфейсу;

3.2. реалізація основних функцій: такі як малювання, обрізання, переміщення, зміна розміру, застосування кольорів, фільтрів та інших ефектів до зображень. Це включає обробку користувацьких дій, таких як кліки мишею, рухи та введення з клавіатури;

3.3. реалізація можливостей до редагування: відміна та повтор дій, копіювання, вставка, видалення, групування та шарування об'єктів на зображенні;

3.4. обробка зображень: коригування кольорів, яскравості, контрастності, застосування фільтрів;

3.5. реалізація алгоритмів, що впливають на окремі елементи зображення, наприклад, зміна розміру, обертання;

4. Тестування та забезпечення якості:

4.1. проведення ретельного тестування, щоб гарантувати, що програмне рішення працює за призначенням і відповідає вимогам;

4.2. впровадження процесів забезпечення якості для виявлення та вирішення будь-яких проблем або помилок, які можуть виникнути на етапах розробки та тестування;

5. Розгортання та підтримка:

5.1 розгортання програмного рішення через windeployqt;

Для досягнення поставленого завдання проект зосередиться на вищезазначених етапах [5], [6].

Успішна розробка та впровадження цього програмного забезпечення дасть користувачам доступ до потужного інструменту, який має на меті надавати широкий спектр інструментів для малювання, маніпуляції зображеннями, створення спеціальних ефектів, колажів у найкоротший у застосуванні час.

Таким чином, розробка дозволить створювати та редагувати графіку, від малюнків та ілюстрацій до дизайну логотипів, брендів, елементів для вебсторінок, та багато іншого.

1.2 Аналіз аналогів програмного продукту

Існує багато растрових графічних редакторів, які мають свої унікальні особливості та функціональні можливості. У цьому розділі ми проведемо аналіз трьох популярних інструментів: Microsoft Paint, Adobe Photoshop та Krita.

Microsoft Paint - (також відомий як MS Paint або Paint) - це растровий графічний редактор, який входить у стандартний набір програм Windows. (рис.1.1). Він має простий інтерфейс та базові функції для малювання, редагування та обробки растрових зображень. Основні функції включають в себе малювання фігур, маніпуляції з зображеннями: кольори, розміри, вирізання та копіювання, також є підтримка роботи з шарами та збереження зображень у різних форматах, таких як BMP, JPEG і PNG [7].

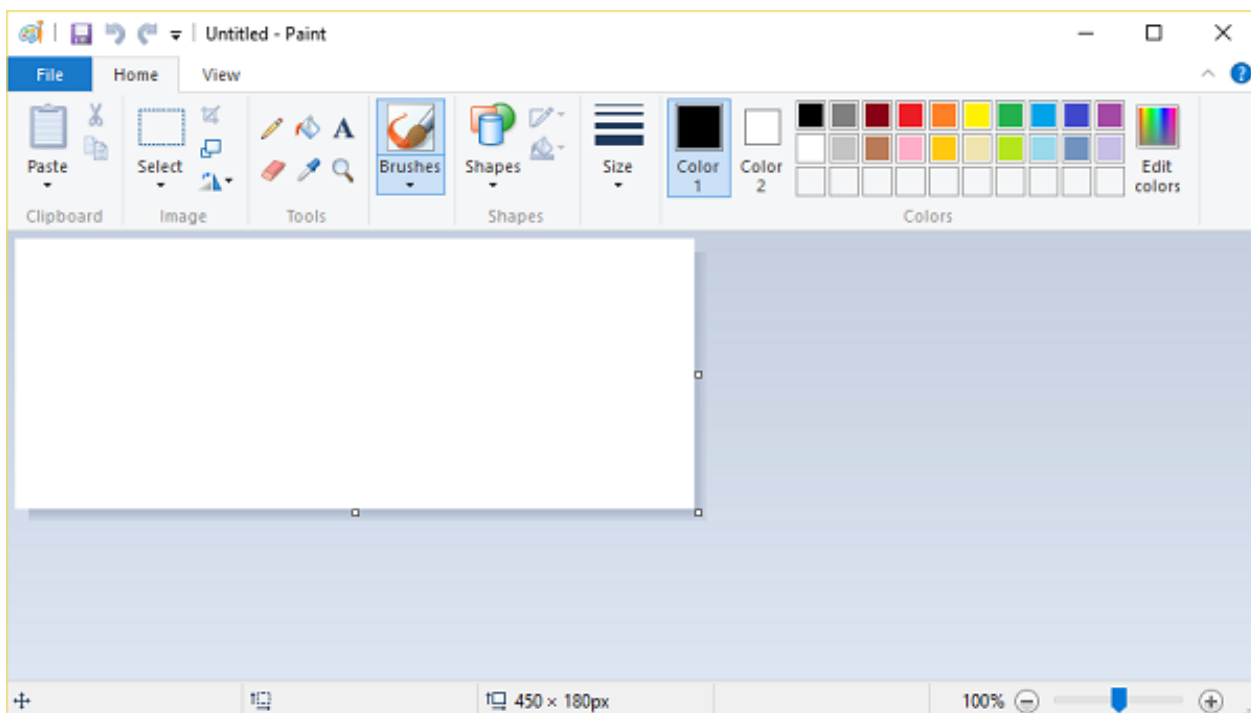


Рисунок 1.1 – Інтерфейс Microsoft Paint.

Переваги:

- зрозумілий і простий у використанні інтерфейс;
- не потребує значних обчислювальних ресурсів і може працювати на більшості комп'ютерів без будь-яких проблем;
- базові функції малювання (пензель, олівець, кисть та випромінювання);
- зручне копіювання та вставка;
- підтримка основних форматів зображень (BMP, JPEG, PNG);
- вбудовані розміри масштабування;

Недоліки:

- відсутність таких потужних функцій, як шари, маски, редакція тексту, інструменти для ретуші та реставрації зображень, що можуть бути необхідними для професійної роботи;
- відсутність продвинутих функцій обробки зображень (корекція кольору, ретуш, редагування шуму);
- не підтримує роботу з шарами;
- відсутність версії для інших операційних систем окрім Windows;

Adobe Photoshop - графічний редактор, розроблений і поширюваний фірмою Adobe Systems (рис.1.2). Розроблений компанією Adobe, Photoshop надає безліч функцій для створення, редагування та обробки растрових зображень [8].

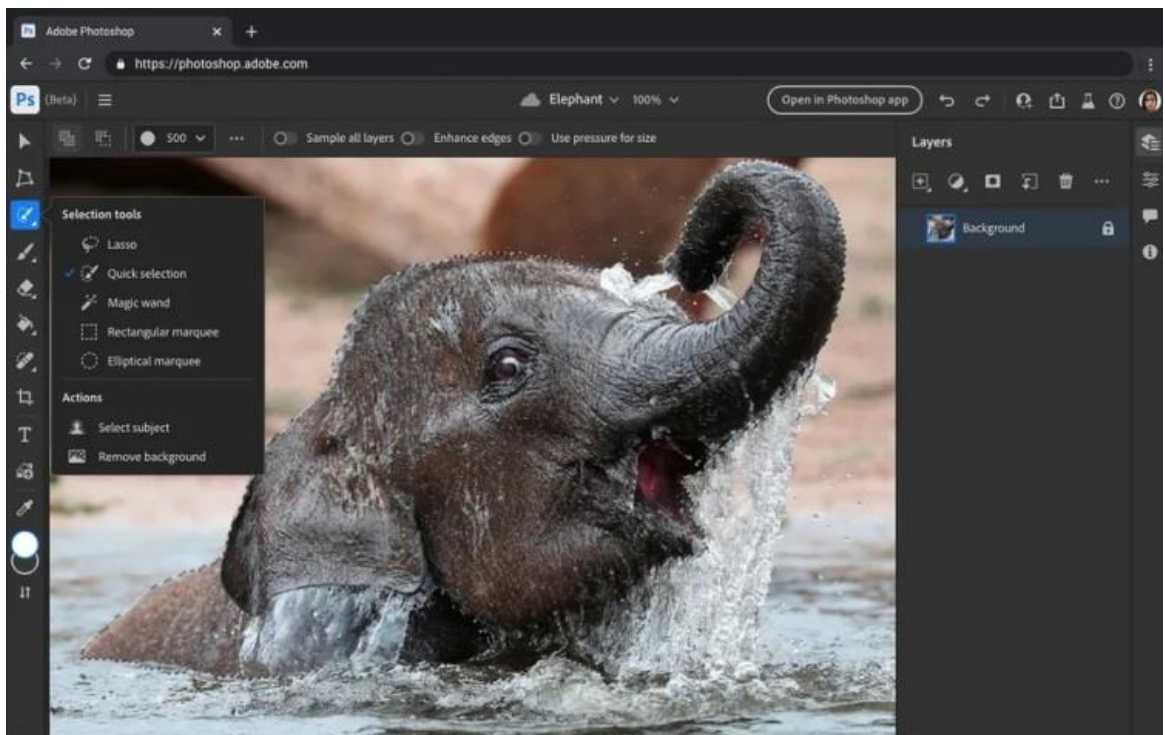


Рисунок 1.2 – Інтерфейс Adobe Photoshop.

Переваги:

- широкий набір інструментів, які дозволяють ретушувати, коригувати та творити малюнки (пензлі, кисті, виправлення дефектів, редакцію кольору, маски, фільтри);
- дозволяє працювати з шарами, що дозволяє створювати складні композиції та редагувати окремі елементи незалежно один від одного;
- потужні інструменти для редагування фотографій, включаючи корекцію експозиції, кольорову корекцію, видалення червоних очей, ретушування шкіри, зміну фону;
- робота з текстом;
- підтримка широкого спектру форматів файлів;

Недоліки:

- дорога ліцензія;
- великі системні вимоги: вимагає значних обчислювальних ресурсів і потужного комп'ютера для оптимальної роботи;

- складність для новачків: Photoshop має великий набір функцій і складний інтерфейс, що може зробити його складним для освоєння для новачків;
- великий обсяг файлів при використанні багатьох шарів і високої роздільної здатності;

Krita - це відкрите програмне забезпечення для растрової графіки, яке призначене для малювання, редагування та обробки зображень (рис.1.3). Він доступний безкоштовно і підтримується спільнотою розробників [9].

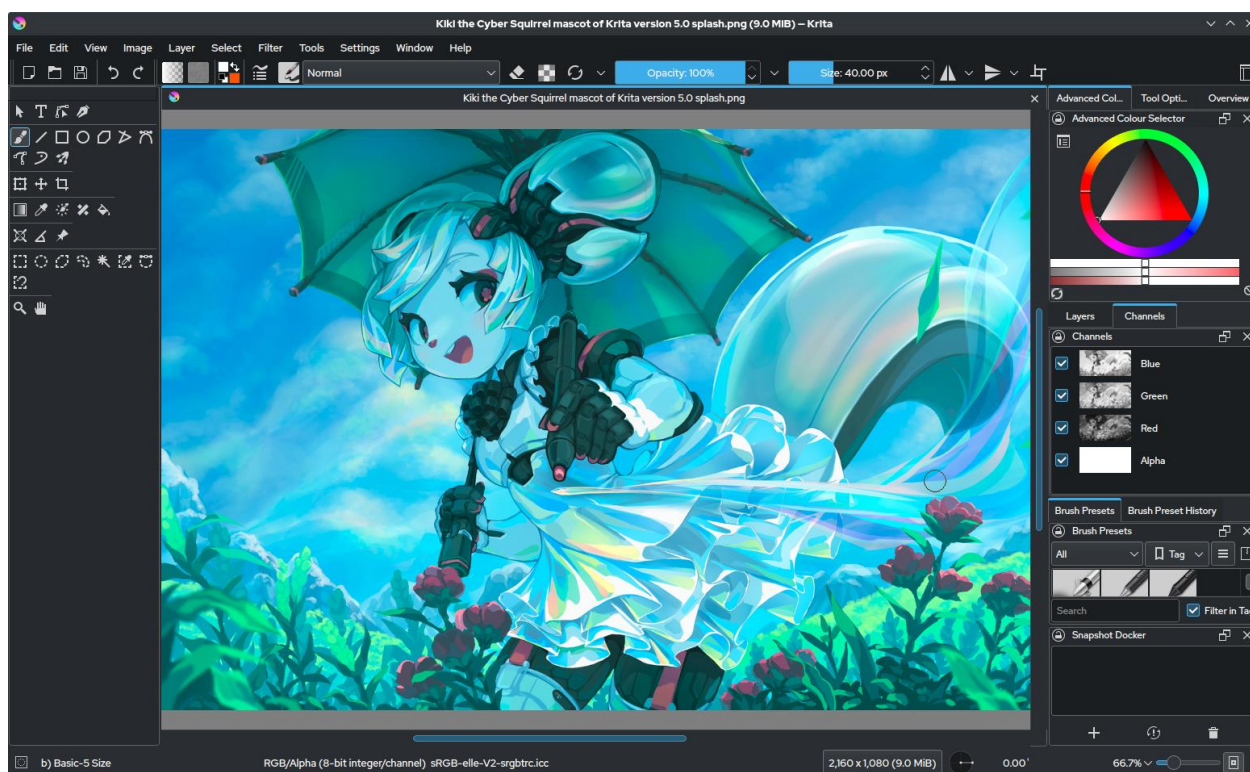


Рисунок 1.3 – Інтерфейс Krita.

Переваги:

- широкий набір пензлів та інструментів для малювання, включаючи різноманітність типів пензлів, текстур та налаштувань, що дозволяє художникам створювати різноманітність стилів і ефектів;
- має інструменти для редагування зображень, такі як обрізка, зміна розміру, корекція кольору, режими змішування, фільтри та багато інших, що дозволяють виправляти та поліпшувати фотографії та інші растрові зображення;

- підтримує роботу з шарами, що дозволяє організовувати зображення на окремих шарах і редагувати їх незалежно один від одного;
- підтримує використання масок, що дозволяє точно контролювати видимість і редагування областей зображення;
- можливість створення анімації;

Недоліки:

- Відсутність підтримки СМΥΚ: Krita зосереджений на роботі з растровими зображеннями і не підтримує кольоровий простір СМΥΚ, який використовується в друку;
- менша спільнота користувачів;
- потреба в обчислювальних ресурсах: Krita може вимагати значних обчислювальних ресурсів, особливо при роботі з великими файлами або складними ефектами;
- обмежена сумісність з іншими програмами;

На основні аналізу аналогів програмного продукту було складено порівняльну таблицю:

Таблиця 1.1

Порівняння існуючих растрових графічних редакторів

	MS Paint	Adobe Photoshop	Krita
Простий інтерфейс	+	-	+
Можливості налаштування	+	+	+
Операційна система	Windows	Microsoft Windows, macOS і iPadOS	Linux, Microsoft Windows, macOS, FreeBSD, OpenBSD
Доступність (ціна)	+	-	+
Функціонал	-	+	+

Швидкість та продуктивність	+	+	+
Розширення та плагіни	-	+	+
Підтримка форматів файлів	+	+	+

Беручи до уваги всі аспекти, ми можемо ретельно проаналізувати та включити найкращі функції та можливості MS Paint, Adobe Photoshop та Krita у наш власний продукт. Такий інтегрований підхід гарантує, що наш продукт не лише задовольняє різноманітні потреби користувачів, але й використовує сильні сторони цих визнаних растрових графічних редакторів.

Завдяки комплексному підходу наш продукт гарантує користувачам доступ до потужного та зручного інструменту, що надаватиме широкий спектр інструментів для малювання, маніпуляції зображеннями, створення спеціальних ефектів, колажів у найкоротший у застосуванні час [10].

1.3 Вибір архітектури растрового графічного редактора

При виборі архітектури графічного редактора на основі інструментарію розробки мовою програмування C++ – Qt важливо пам'ятати про варіанти використання, а саме: GUI, обробка зображень, рендерінг та відображення, архітектура модулів та кросплатформеність. Оптимальним для реалізації редактора буде використання Qt Paint System. Paint System у Qt дозволяє малювати на екрані та пристроях друку, використовуючи той самий API, і в основному базується на класах QPainter, QPaintDevice та QPaintEngine [11], [12].

QPainter використовується для виконання операцій малювання, QPaintDevice є абстракцією двовимірного простору, на якому можна малювати за допомогою QPainter, а QPaintEngine надає інтерфейс, який використовується художником для малювання на різних типах пристроїв. Клас QPaintEngine використовується внутрішньо QPainter та QPaintDevice і є прихованим від прикладних програмістів, доки вони не створять свій власний тип пристрою.

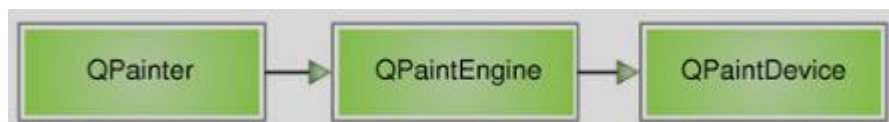


Рисунок 1.4 – Архітектура Qt Paint System.

Основна перевага цього підходу полягає в тому, що всі малювання виконуються в одному конвеєрі малювання, що полегшує додавання підтримки нових функцій і забезпечує реалізацію за замовчуванням для непідтримуваних функцій.

Таким чином, вибір архітектури розроблюваного додатку є дуже важливим, оскільки він має вплив на багато аспектів проекту і може визначити його майбутні можливості, розширюваність, продуктивність та зручність у використанні.

1.4 Обґрунтування вибору інструментальних засобів та вимог до апаратного забезпечення

Вибір інструментальних засобів для розробки додатку є важливим завданням, яке може впливати на ефективність, продуктивність та якість проекту. Цей розділ містить стислий порівняльний опис архітектурних патернів та огляд обраної архітектури, яка реалізована в додатку.

Для реалізації растрового графічного редактора обрано мову програмування C++ та крос-платформовий інструментарій розробки програмного забезпечення Qt в поєднанні зі зручним візуальним редактором коду Qt Creator.

Використання мови програмування C++ для розробки графічних редакторів має свої переваги і обґрунтування:

- Швидкодія: C++ є мовою з низьким рівнем абстракції, що дозволяє прямий доступ до ресурсів комп'ютера і ефективне використання пам'яті. Це особливо важливо для графічних редакторів, які працюють з зображеннями та потребують швидкого оброблення та відображення графіки [13];
- Кросплатформовість: C++ є платформонезалежною мовою, що дозволяє розробляти графічні редактори, які можуть працювати на різних операційних системах, таких як Windows, macOS і Linux. Можна використовувати бібліотеки, такі як Qt, які надають кросплатформову підтримку для розробки графічних інтерфейсів [14];
- Високий рівень контролю: C++ надає високий рівень контролю над системою та ресурсами. Це дозволяє оптимізувати роботу з пам'яттю, оптимізувати алгоритми та реалізувати власні оптимізації для підвищення продуктивності редактора;
- Розширюваність: C++ надає можливість створювати модульну архітектуру та легко розширювати функціональність графічного редактора;

- Завдяки об'єктно-орієнтованому програмуванню і можливості використання патернів проектування, можна побудувати гнучкі системи, які легко розширюються і підтримують нові функції та плагіни [15], [16];

Використання **Qt** для написання графічного редактору є досить популярним вибором та має свої переваги і обґрунтування:

- обробка подій та сигналів: Qt має потужну систему обробки подій та сигналів, яка дозволяє легко відстежувати взаємодію користувача з програмою. Можна легко реагувати на події, такі як натискання кнопок, переміщення миші або зміна значень полів. Це дозволяє створювати відзивчиві та інтерактивні графічні редактори;
- готові компоненти та інструменти: Qt надає широкий набір готових компонентів, таких як кнопки, тексти, меню, діалогові вікна та багато інших;
- потужні інструменти для маніпулювання зображеннями, роботи з графікою та обробки подій;
- підтримка роботи з графікою: Qt надає багато можливостей для роботи з графікою. Можна маніпулювати зображеннями, малювати фігури, використовувати растрову графіку, застосовувати ефекти до зображень;
- велика спільнота та документація: Qt має велику спільноту розробників, що активно підтримує цей фреймворк разом з високоякісною документацією з прикладами коду та поясненнями, як використовувати різні функції фреймворку [17];

На завершення, обґрунтування вибору інструментів та вимог до апаратного забезпечення для проекту надає чітке обґрунтування вибору ресурсів, які ефективно підтримуватимуть розробку та експлуатацію програмного забезпечення.

Ретельно зваживши потреби проекту, технічні міркування та організаційні вимоги, обрані інструменти та апаратне забезпечення були адаптовані для оптимізації ефективності, продуктивності та рентабельності.

Висновки до розділу 1

Перший розділ розпочався з чіткої постановки задачі, виокремлюючи мету розробки растрового графічного редактора.

Далі було проведено глибокий аналіз аналогів програмного продукту, зосередившись на трьох відомих растрових графічних редакторах, а саме: MS Paint, Adobe Photoshop та Krita. Цей аналіз висвітлив особливості, функціональність, переваги та недоліки кожного інструменту, що дозволило краще зрозуміти існуючі рішення на ринку, щоб розробити максимально зручний і корисний продукт.

Крім того, розділ заглибився у важливий аспект вибору архітектури для розробки растрового графічного редактора. Була розглянута технологія Qt. Обґрунтування вибору інструментів та вимог до апаратного забезпечення базувалося на їх сумісності, масштабованості, продуктивності та кращих галузевих практиках.

Було описано обрану архітектуру редактора на основі інструментарію розробки Qt, спираючись на головні варіанти використання, а саме: GUI, обробка зображень, рендерінг та відображення, архітектура модулів та кросплатформеність.

В цілому, цей розділ заклав основу для розробки редактора, що надаватиме широкий функціонал та залишатиметься швидким та простим у використанні. Аналіз аналогів і ретельний вибір інструментів та архітектури проклали шлях до наступних розділів, де детально розглянуто проектування та реалізацію програмного забезпечення.

РОЗДІЛ 2. ПРОЕКТУВАННЯ РАСТРОВОГО ГРАФІЧНОГО РЕДАКТОРА

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури

У цьому розділі представлено аналіз вимог до програмного продукту з акцентом на виявленні функціональних та нефункціональних вимог. Крім того, визначено вимоги до часових характеристик реакції додатку на дії користувача, а також окреслено різні варіанти використання. Для забезпечення чіткого розуміння структури додатково включено узагальнену діаграму елементів (сутностей системи), їх структури та методів з описом [18].

Аналіз вимог передбачає докладне вивчення потреб та очікувань користувачів щодо функціональності та характеристик графічного редактора. Основною метою аналізу вимог є зрозуміти, що потрібно втілити в програмному забезпеченні для задоволення потреб користувачів. За допомогою інтерв'ю з користувачами, опитувань та зворотного зв'язку були визначені функціональні вимоги, які окреслюють специфічні особливості та можливості, якими повинен володіти програмний продукт. На додаток були зібрані нефункціональні вимоги, які стосуються таких аспектів, як продуктивність, стабільність роботи і зручність використання [19].

Функціональні вимоги:

- малювання та редагування зображень: можливість створювати нові зображення з нуля або відкривати наявні зображення для редагування; інструменти малювання, такі як олівці, пензлі, заливка кольором; зміна розміру, обрізка та обертання зображень;
- обробка зображень: фільтри та ефекти для покращення якості зображень, такі як розмиття, насиченість;
- робота з текстом: додавання та редагування текстових елементів на зображенні; вибір шрифту, розміру, кольору; розташування тексту на зображенні;

- збереження та експорт зображень: збереження зображень у різних форматах, таких як JPEG, PNG, GIF, TIFF;
- базові налаштування додатку: розмір робочого полотна, авто збереження, історія дій, мова інтерфейсу;

Нефункціональні вимоги:

- система повинна швидко реагувати на дії користувача, забезпечуючи безперебійну та чуйну роботу;
- ефективне використання ресурсів: редактор повинен ефективно використовувати пам'ять та процесор, щоб забезпечити швидку та безперебійну роботу, навіть з великими зображеннями;
- відновлення стану: при падінні або несподіваному закритті редактора, важливо мати можливість відновити збережені дані та відновити останній відкритий проект;
- сумісність з форматами зображень: редактор повинен підтримувати різні формати зображень, щоб користувачі могли легко імпортувати та експортувати свої проекти;
- захист даних: редактор повинен забезпечувати безпеку та конфіденційність збережених зображень та проектів;
- інтуїтивний інтерфейс: редактор повинен мати зрозумілий та легкий у використанні інтерфейс;

Часові характеристики реакції системи:

- час відгуку на дії користувача: швидкий час відгуку забезпечує миттєву відповідь на дії користувача, що поліпшує загальний досвід роботи з редактором;
- час завантаження та збереження зображень: редактор повинен ефективно завантажувати великі файли зображень та зберігати їх після редагування; оптимізація цих процесів допомагає забезпечити швидку обробку та збереження зображень;
- час виконання операцій: деякі операції, такі як фільтри, ефекти або розмиття, можуть займати багато часу, особливо при роботі з

великими зображеннями, тому швидкість виконання цих операцій впливає на плавність та продуктивність роботи з редактором;

- переміщення та масштабування зображень: коли користувач переміщає або масштабує зображення на полотні редактора, важливо, щоб ця операція відбувалася безперервно та без лагів; плавність переміщення та масштабування дозволяє користувачеві зручно працювати з зображенням;
- відкриття та закриття програми: швидке відкриття програми дозволяє користувачеві швидко почати працювати, а швидке закриття забезпечує збереження часу та енергії користувача;

Важливо враховувати, що часові характеристики реакції графічного редактора можуть залежати від різних факторів, таких як обладнання комп'ютера, розмір та складність зображення, а також оптимізація самої програми [20].

За допомогою тестування продуктивності визначаються прийнятні часові межі для різних дій та процесів у системі. Ці часові характеристики є важливими для задоволення очікувань користувачів та уникнення розчарування, спричиненого затримками або відсутністю реакції [21].

Варіанти використання. Щоб задовольнити різноманітні потреби користувачів, система пропонує різні варіанти використання. Забезпечуючи гнучкість у використанні системи, користувачі можуть адаптувати програмний продукт до своїх конкретних вимог і робочих процесів, підвищуючи свій загальний досвід і продуктивність.

Діаграма використання растрового графічного редактора відображає основні функціональні можливості редактора та взаємодію користувача з системою. Основний актор на діаграмі - це користувач редактора, такий як артист, дизайнер, фотограф тощо. Він взаємодіє з системою через різні використовувані випадки використання [22].

Ми створили діаграму варіантів використання (рис.2.1) - візуальне представлення взаємодії між користувачем системи (актором) і самою системою та специфікацію прецедентів (дод.А).



Рисунок 2.1 – Діаграма варіантів використання.

Користувач:

- створення нового проекту: обрати формат та розмір робочої поверхні або відкрити існуючий проект;
- загальні налаштування редактора: мова інтерфейсу, розмір полотна при завантаженні додатку, глибина історію, авто збереження;
- відкриття та збереження зображень: відкрити або імпортувати існуюче зображення; зберегти редаговане зображення у вибраному форматі;
- друк готового зображення;
- модальні вікна виходу та про програмний продукт;

- редагування зображень: застосовування різних інструментів, трансформацій та ефектів до зображення;

Визначені функціональні можливості для користувача створюють комплексне рішення, яке задовольняє різноманітні графічні потреби.

Загалом, діаграма варіантів використання забезпечує комплексне і наочне представлення функціональних можливостей. Визначаючи та документуючи варіанти використання, діаграма слугує цінним інструментом для аналізу системи, збору вимог та проектування.

Об'єктно-орієнтована структура. Структура дозволяє модульний дизайн системи, де різні елементи представлені як об'єкти з визначеними атрибутами та методами. Завдяки інкапсуляції функціональності в об'єкти, система сприяє повторному використанню, модульності та розширюваності.

Наведено узагальнену діаграму об'єктно-орієнтованої структури системи, яка ілюструє взаємозв'язки та взаємодію між різними об'єктами системи (рис.2.2). Кожна сутність представляє певний компонент або модуль програмного продукту, з власними атрибутами та методами. Діаграма слугує візуальним представленням архітектури системи, сприяючи кращому розумінню її загальної структури [23].

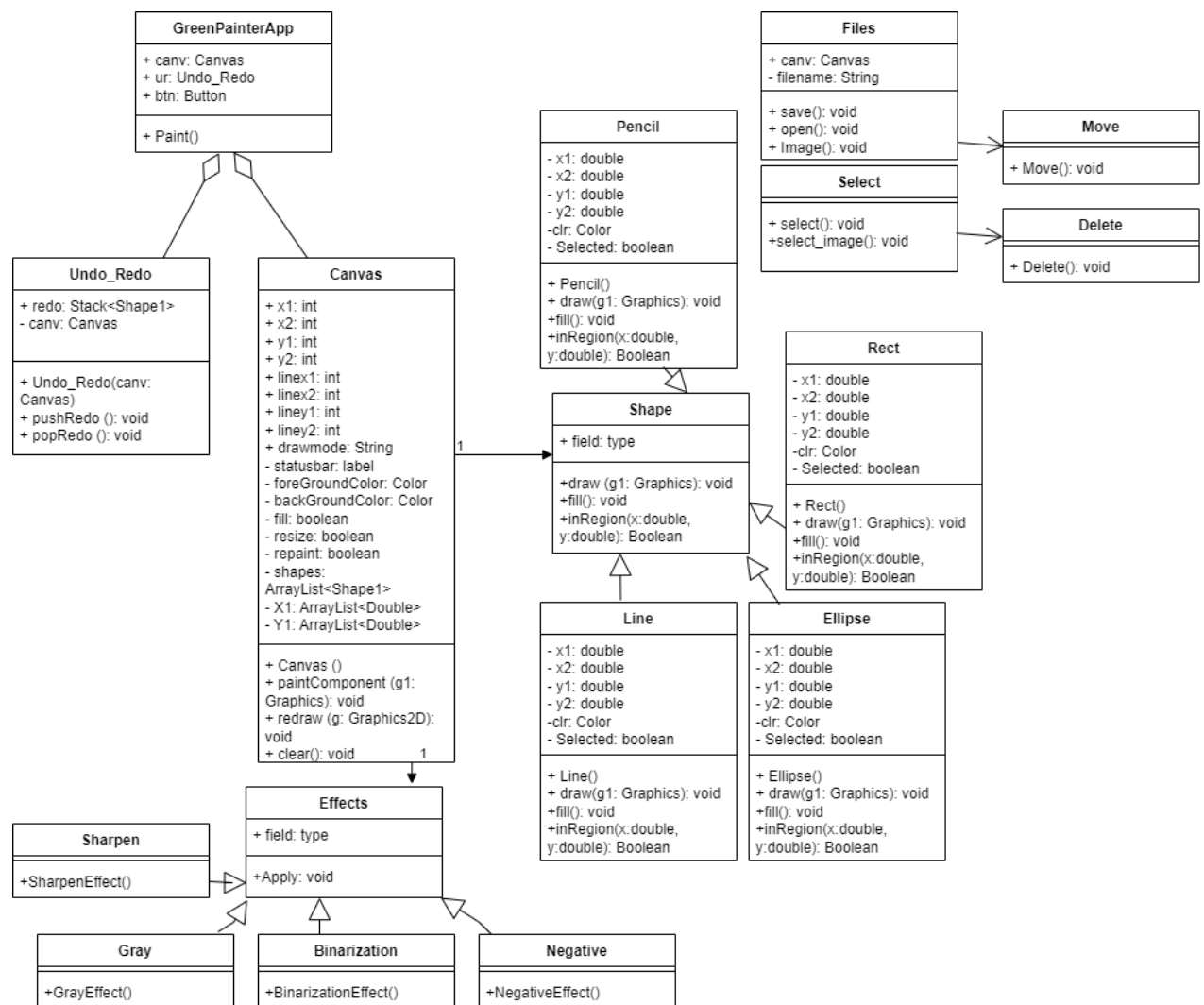


Рисунок 2.2 – Діаграма класів.

В діаграмі можна виділити класи, які спроектовано, використовуючи шаблон “Фабричний метод” [24].

Фабричний метод – породжуючий шаблон проектування, що надає підкласу можливість створення екземплярів деякого класу. В момент створення підкласи можуть визначити, який клас створювати. Іншими словами, даний шаблон надає право створення об’єктів батьківському класу. Це дозволяє використовувати в коді програм не специфічні класи, а маніпулювати абстрактними об’єктами на більш високому рівні. Таким чином, маємо сутності:

- GreenPainterApp - клас ініціалізації. Клас має конструктор який приймає розмір робочої поверхні та наступні методи: Paint – для створення та відображення полотна на екрані;
- Undo_Redo – абстрактний клас для роботи з подіями від користувача. Клас

має конструктор, що приймає полотно, на якому працює користувач та лист з завданням. Методи: Do – повторити дію, Redo – повернути дію;

- Canvas – клас ініціалізації робочої поверхні. Методи: PaintComponent – промальовування робочої поверхні з зазначеними параметрами. Redraw() – метод малювання. Clear() – метод очищення;

- Effect – абстрактний клас для роботи з графічними елементами зображень. Клас має конструктор, який приймає назву обраного користувачем ефекту;

- Gray, Sharpen, Binarization, Negative – класи графічного перетворення зображення;

- Shape – клас для роботи з фігурами. Клас має конструктор який приймає назву обраного до побудови фігури. Методи: draw() – намалювати, fill() – заливка, inRegion() – положення на полотні;

- Line, Rect, Ellipse, Pencil – класи фігур та інструментів для роботи. Методи: Draw(), inRegion();

- Files – робота з файлами, методи: Open(), Save(), Image();

Таким чином, дана система реалізує функціонал графічного редактора.

2.2 Опис структури даних з якими працює растровий графічний редактор

Qt надає розширений функціонал для роботи з зображеннями різних типів. Основним класом для роботи з зображеннями в Qt є клас QImage. Клас дозволяє завантажувати, зберігати, маніпулювати та відображати зображення [25].

Основні функції та можливості класу QImage включають:

- Завантаження та збереження зображення: можна завантажувати зображення з файлу або зберігати зображення в файлі. Це можна зробити за допомогою методів load() та save();
- Створення зображення: можна створити новий об'єкт QImage за допомогою конструктора, який приймає розміри зображення та формат пікселів;
- Маніпулювання зображенням: QImage надає різноманітні методи для маніпулювання зображенням, такі як зміна розміру (scaled()), обрізання (copy()), зміна яскравості та контрастності (adjustBrightness(), adjustContrast()) та багато інших;
- Отримання даних пікселів: можна отримати доступ до пікселів зображення за допомогою методів bits() або scanLine(). Це дозволяє працювати безпосередньо з пікселями та їх значеннями;
- Відображення зображення: QImage можна відображати на віджетах Qt, таких як QLabel або QGraphicsView, або використовувати для малювання на QPainter-об'єктах;

Qt також надає клас QPixmap, який є підкласом класу QImage і забезпечує додаткові можливості для роботи з зображеннями, зокрема використання в інтерфейсі користувача.

2.3 Проектування та реалізація алгоритмів роботи системи

Розробка та реалізація системних алгоритмів відповідають за виконання різних обчислювальних завдань, обробку даних і прийняття рішень в системі. Ми створили декілька діаграм активності для візуалізації та представлення потоку діяльності в системі. Вони пропонують чіткий спосіб зображення послідовних і паралельних дій, які відбуваються під час виконання системних процесів. Створюючи діаграми активності, ми прагнемо забезпечити всебічне розуміння алгоритмічного дизайну системи та її реалізації [26].

Діаграма активності для створення нового проекту (рис.2.3): ця діаграма показує послідовність дій користувача, таких як вибір розміру полотна, що в подальшому перейде у створення нового об'єкту зображення та його ініціалізацію.



Рисунок 2.3 - Діаграма активності для створення нового проекту.

На основі діаграми активності для створення нового проекту було створено наступну таблицю специфікації:

Таблиця 2.1

Ім'я варіанту використання: створення нового проекту		Рівень: високий
Основний актор: користувач		
Короткий опис:		Створення робочого полотна для редагування зображення
Передумови:		Запустити редактор
Відносини:		
	Включення:	Ініціалізація
Основний потік:		Користувач може створити нове робоче полотно у поточному чи новому вікні. Також є можливість вибору розміру та прапорець автозбереження
Альтернативні/виключні потоки:		Повідомлення помилки

Діаграма активності для редагування зображення (рис.2.4): Ця діаграма може показати послідовність дій користувача для редагування зображення, таких як вибір інструмента, застосування фільтрів, малювання тощо.

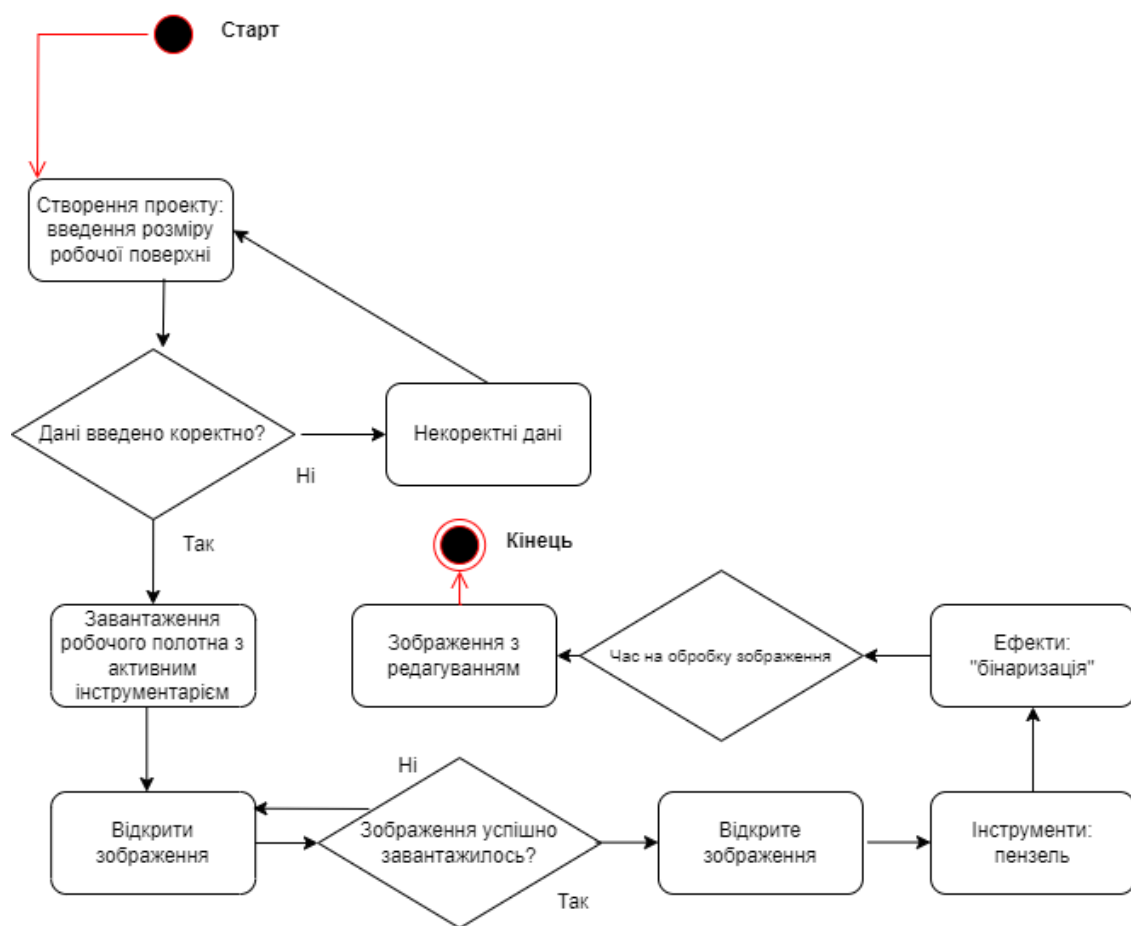


Рисунок 2.4 - Діаграма активності для редагування зображення.

На основі діаграми активності для редагування зображення було створено наступну таблицю специфікації:

Таблиця 2.2

Ім'я варіанту використання: редагування зображення		Рівень: високий
Основний актор: користувач		
Короткий опис:	Процес редагування зображення	
Передумови:	Запустити редактор, створити робочий проект	
Відносини:		
	Включення:	Ініціалізація
Основний потік:	Користувач має створити нове робоче полотно у поточному чи новому вікні, зазначивши потрібні розміри. Далі йде процес імпорту зображення та його подальше редагування з використанням інструментарію редактора.	
Альтернативні/виключні потоки:	Повідомлення помилки	

Діаграма активності для відміни або повтору дій (рис.2.5): ця діаграма показує, як користувач може виконати відміну або повторити певні дії в графічному редакторі.

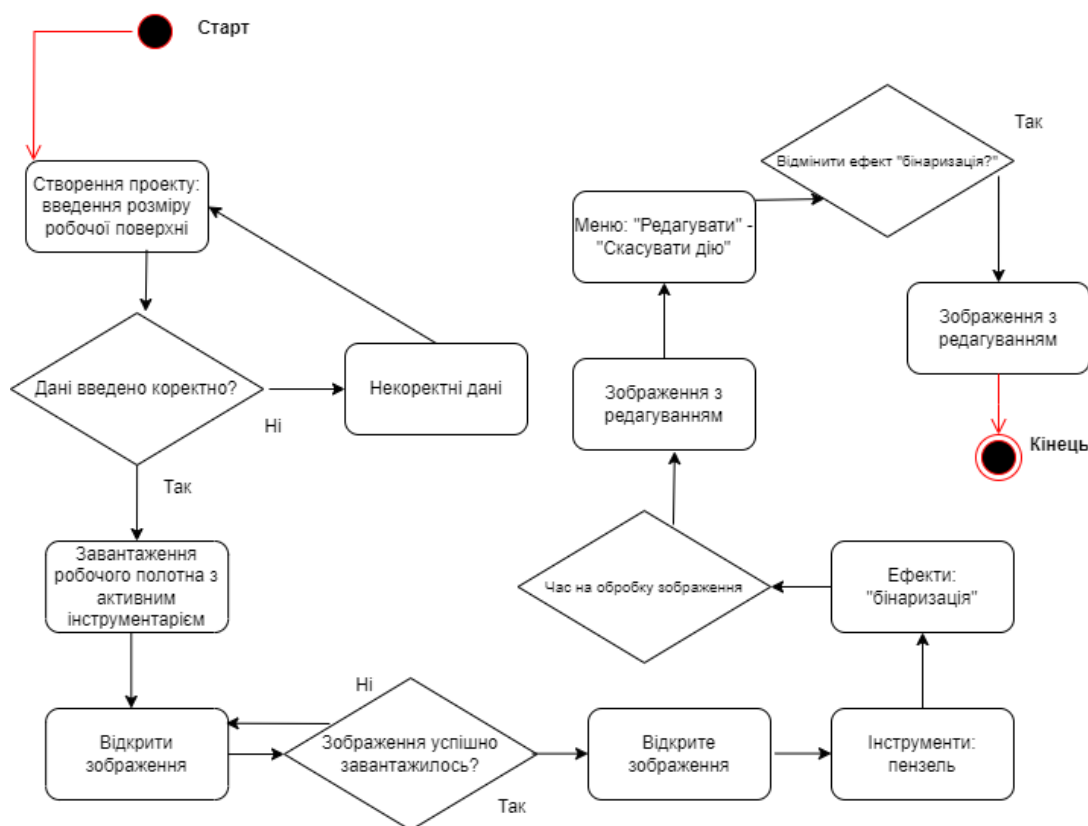


Рисунок 2.5 - Діаграма активності для відміни дій.

На основі діаграми активності для відміни дій було створено наступну таблицю специфікації:

Таблиця 2.3

Ім'я варіанту використання: відміна дії		Рівень: високий	
Основний актор: користувач			
Короткий опис:		Процес відміни дії	
Передумови:		Запустити редактор, створити робочий проект, виконати хоча б одну дію за допомогою інструментарію. Також важливо мати активний прапорець у полі 'Історія'. Значення цього поля має бути більшим 1.	
Відносини:			
	Включення:	Ініціалізація	
Основний потік:		Користувач має створити нове робоче полотно у поточному чи новому вікні, зазначивши потрібні розміри. Далі йде	

	процес імпорту зображення та його подальше редагування з використанням інструментарію редактора. При потребі користувач може корегувати свої дії з можливістю переходу по історії своїх дій.
Альтернативні/виключні потоки:	Повідомлення помилки

Діаграма активності для імпортування фото та застосування інструменту ‘Текст’ (рис.2.6): ця діаграма може показати, як користувач може імпортувати зображення та створити текстовий об’єкт.

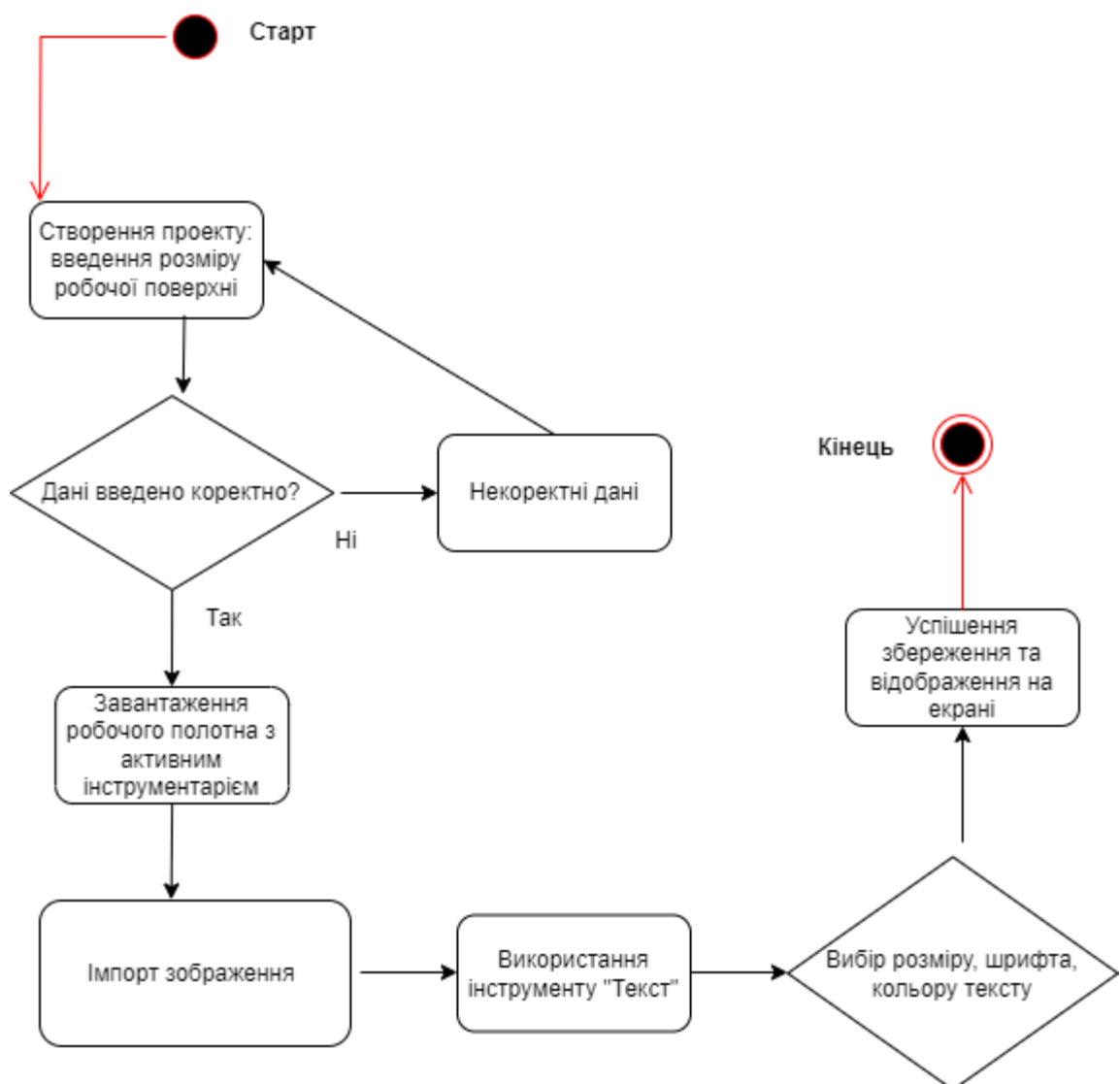


Рисунок 2.6 - Діаграма активності для імпортування фото та застосування інструменту ‘Текст’.

На основі діаграми активності для імпортування фото та застосування інструменту ‘Текст’ було створено наступну таблицю специфікації:

Таблиця 2.4

Ім'я варіанту використання: імпорт зображення та застосування інструменту ‘Текст’		Рівень: високий
Основний актор: користувач		
Короткий опис:	Застосування інструментарію редактора під час роботи з зображенням	
Передумови:	Запустити редактор, створити робочий проект	
Відносини:		
	Включення:	Ініціалізація
Основний потік:	Користувач може імпортування зображення будь-якого файлового типу та використати інструмент ‘Текст’ для створення, наприклад, заголовку.	
Альтернативні/виключні потоки:	Повідомлення помилки	

Ці діаграми активності та таблиці специфікації допоможуть уявити та візуалізувати послідовність дій та взаємодію користувача з графічним редактором.

Діаграма послідовності в графічному редакторі може показати взаємодію між різними об'єктами та компонентами системи під час виконання певної функціональності. Основними елементами діаграми послідовності будуть об'єкти (класи, модулі), повідомлення (методи, функції) та часова послідовність виконання [27].

Діаграма послідовності, що описує дії користувача, який взаємодіє з полотном. (рис.2.7)

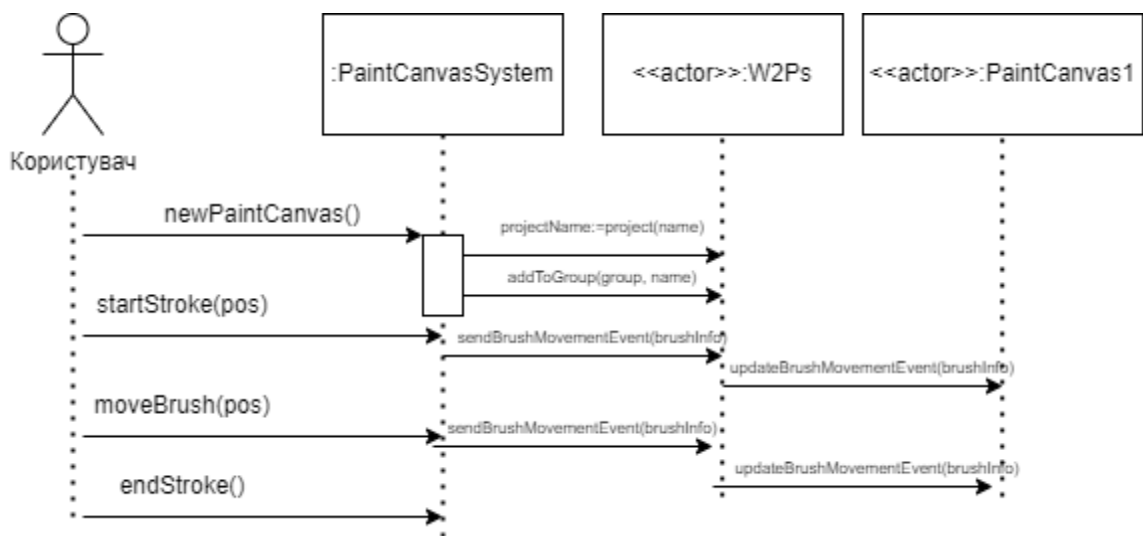


Рисунок 2.7 - Діаграма послідовності взаємодії з робочим полотном.

Загалом, діаграми послідовності пропонують стисле і наочне представлення динамічної взаємодії та потоку повідомлень у системі. Вони відіграють важливу роль у розумінні послідовності подій та обміну інформацією, дозволяючи краще аналізувати та розуміти поведінку системи.

Ефективно розробляючи та впроваджуючи ці системні алгоритми, ми можемо забезпечити безперебійну роботу системи, надаючи користувачам ефективне управління завданнями, безперебійну комунікацію, інтеграцію із зовнішніми інструментами, глибокі звіти.

2.4 Реалізація функціоналу растрового графічного редактора

У цьому розділі ми обговоримо етапи процесу впровадження та висвітлимо ключові аспекти розробки додатку.

Алгоритм вирішення задач за допомогою графічного редактора може варіюватися в залежності від конкретної задачі, яку ви намагаєтеся вирішити, але однією з найголовніших функцій редактора – є використання ефектів.

При виборі інструментарії розробки було зазначено, що пріоритетною причиною вибору C++ - це можливість оптимізації графічних перетворень.

Оптимізація, наприклад, ефекту бінаризації у графічних редакторах може бути досягнута за допомогою кількох підходів. В нашому випадку був обраний алгоритм Otsu.

Алгоритм Otsu є одним з найпоширеніших методів бінаризації зображень. Він автоматично визначає порогове значення для розділення зображення на області переднього плану і тла. Основна ідея алгоритму полягає в тому, щоб максимізувати міжкласову дисперсію пікселів, тобто розділити зображення таким чином, щоб дисперсія між областями була максимальною [28].

```
void BinarizationEffect::makeBinarization(ImageArea& imageArea, int coeff1, int
coeff2)
{
    for (int x(0); x < imageArea.getImage()->width(); x++)
    {
        for (int y(0); y < imageArea.getImage()->height(); y++)
        {
            QRgb pixel = imageArea.getImage()->pixel(x, y);
            int r = (int)qRed(pixel);
            if (r >= coeff1)
                r = 0;
            else if (r >= coeff2 && r < coeff1)
                r = 255;
            else
                r = 0;
            pixel = qRgb(r, r, r);
            imageArea.getImage()->setPixel(x, y, pixel);
        }
    }
}
```

Оптимізація чорно-білого ефекту може бути досягнута за допомогою кількох підходів, наприклад інтерполяції при зміні розмірів: якщо необхідно змінити розміри зображення, треба використовувати оптимізовані алгоритми

інтерполяції, такі як білінійна інтерполяція. Це дозволить зберегти якість зображення та покращити продуктивність обробки [29].

```
void GrayEffect::applyEffect(ImageArea& imageArea) {
    imageArea.clearSelection();    makeUndoCommand(imageArea);
    for (int i(0); i < imageArea.getImage()->width(); i++)
    {
        for (int y(0); y < imageArea.getImage()->height(); y++)
        {
            QRgb pixel(imageArea.getImage()->pixel(i, y));
            int rgb = (int)(0.299 * qRed(pixel) + 0.587 * qGreen(pixel) + 0.114 *
qBlue(pixel));
            pixel = qRgb(rgb, rgb, rgb);
            imageArea.getImage()->setPixel(i, y, pixel);
        }
        imageArea.setEdited(true);
        imageArea.update();
    }
}
```

Оптимізація гаусового розмиття може бути досягнута за допомогою різних підходів і методів, наприклад використання властивостей симетрії: Гаусове ядро є симетричним відносно центра, що означає, що ваги пікселів знаходяться на однаковій відстані від центру. Це дозволяє застосувати оптимізації, такі як збереження та повторне використання ваг, що дозволяє зменшити кількість обчислень [30].

```
GaussianBlurFilterSettings::GaussianBlurFilterSettings(QWidget* parent) :
    AbstractEffectSettings(parent)
{
    QLabel* label = new QLabel(tr("Інтенсивність"), this);

    mIntensitySlider = new QSlider(Qt::Horizontal, this);
    mIntensitySlider->setTickPosition(QSlider::TicksBothSides);
    mIntensitySlider->setMinimum(1);
    mIntensitySlider->setMaximum(10);
    mIntensitySlider->setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Fixed);
    connect(mIntensitySlider, &QAbstractSlider::valueChanged, this,
&AbstractEffectSettings::matrixChanged);

    QVBoxLayout* vLayout = new QVBoxLayout();

    vLayout->addWidget(label);
    vLayout->addWidget(mIntensitySlider);
    vLayout->addStretch();
    setLayout(vLayout);
}
```

Оптимізація ефекту негатива в програмі на мові C++ може бути досягнута за допомогою використання оптимізованих функцій та оптимізація пам'яті.

```
void NegativeEffect::applyEffect(ImageArea& imageArea)
{
    imageArea.clearSelection();
    makeUndoCommand(imageArea);

    imageArea.getImage()->invertPixels(QImage::InvertRgb);
    imageArea.setEdited(true);
    imageArea.update();
}
```

Вирішенням цього займається метод `invertPixels()`.

```
void QImage::invertPixels(QImage::InvertMode mode = InvertRgb)
```

Він інвертує всі значення пікселів на зображенні. Цей режим інвертування має сенс лише тоді, коли глибина зображення дорівнює 32. За замовчуванням використовується режим `InvertRgb`, який залишає альфа-канал без змін. Якщо вибрано режим `InvertRgba`, альфа-біти також інвертуються.

Інвертування 8-бітового зображення означає заміну всіх пікселів з колірним індексом i на пікселі з колірним індексом $255 - i$. Те саме відбувається для 1-бітового зображення.

Якщо зображення має альфа-канал з попереднім множенням, зображення спочатку конвертується у формат зображення без попереднього множення для інвертування, а потім конвертується назад.

Оптимізація гамма ефекту в програмі досягнута за допомогою наступного підходу: якщо гамма ефект застосовується до кожного пікселя окремо, можна розглянути попереднє обчислення та збереження значень гамма-корекції для кожного можливого вхідного значення.

```
void GammaEffect::makeGamma(ImageArea& imageArea, float modifier)
{
    for (int x(0); x < imageArea.getImage()->width(); x++)
    {
        for (int y(0); y < imageArea.getImage()->height(); y++)
        {
            QRgb pixel = imageArea.getImage()->pixel(x, y);
            float r = qRed(pixel);
            r = 255 * pow(r / 255, modifier);
            float g = qGreen(pixel);
            g = 255 * pow(g / 255, modifier);
            float b = qBlue(pixel);
            b = 255 * pow(b / 255, modifier);
            pixel = qRgb(r, g, b);
            imageArea.getImage()->setPixel(x, y, pixel);
        }
    }
}
```

Висновки до розділу 2

У цьому розділі ми заглибилися у розробку та реалізацію графічного редактора. По-перше, ми визначили варіанти використання та створили об'єктно-орієнтовану структуру редактора. Шляхом ретельного аналізу ми визначили ключові функціональні можливості, необхідні для того, щоб додаток задовольняв потреби різних користувачів.

Далі ми зосередилися на опису структури даних з якими працює редактор. Було розглянуто класи, які надає для роботи з зображеннями платформа Qt, а саме QImage та QPixmap.

Потім ми приступили до розробки та впровадження алгоритмів системи. Це передбачало розробку ефективних та дієвих алгоритмів для оптимізації графічних перетворень. Використовуючи відповідні алгоритми та найкращі практики, ми прагнули підвищити продуктивність системи та забезпечити надійну роботу.

Ми перетворили дизайн системи та алгоритми на повнофункціональний додаток. Протягом усього процесу реалізації ми надавали пріоритет зручності використання та масштабованості, щоб забезпечити надійне та зручне для користувача рішення.

Загалом, цей розділ був присвячений різним етапам розробки та впровадження графічного редактора, включаючи визначення варіантів використання, об'єктно-орієнтовану структуру. Впроваджуючи найкращі галузеві практики та використовуючи відповідні технології, ми мали на меті створити комплексне рішення, яке задовольнить потреби людей у швидкому та потужному редакторі.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З РАСТРОВИМ ГРАФІЧНИМ РЕДАКТОРОМ

3.1 Порядок встановлення та налаштування системи

Растровий графічний редактор це додаток, що було створено через інструментарій розробки мовою програмування C++ Qt. Засіб розгортання windeployqt призначено для автоматизації процесу створення теки для розгортання, яка містить залежності, пов'язані з Qt (бібліотеки, імпорт QML, плагіни та переклади), необхідні для запуску програми з цієї теки. Він створює дерево встановлення для десктопних програм Windows, яке може бути легко включено до інсталяційного пакунка.

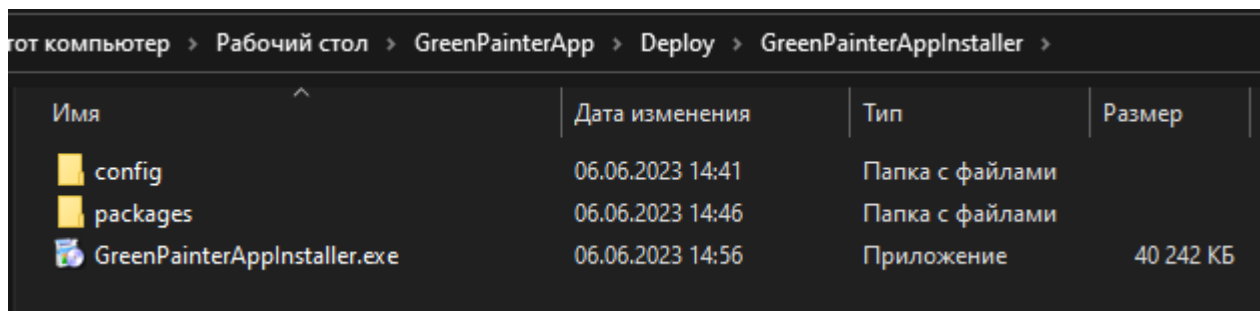


Рисунок 3.1 – Створення теки з залежностями.

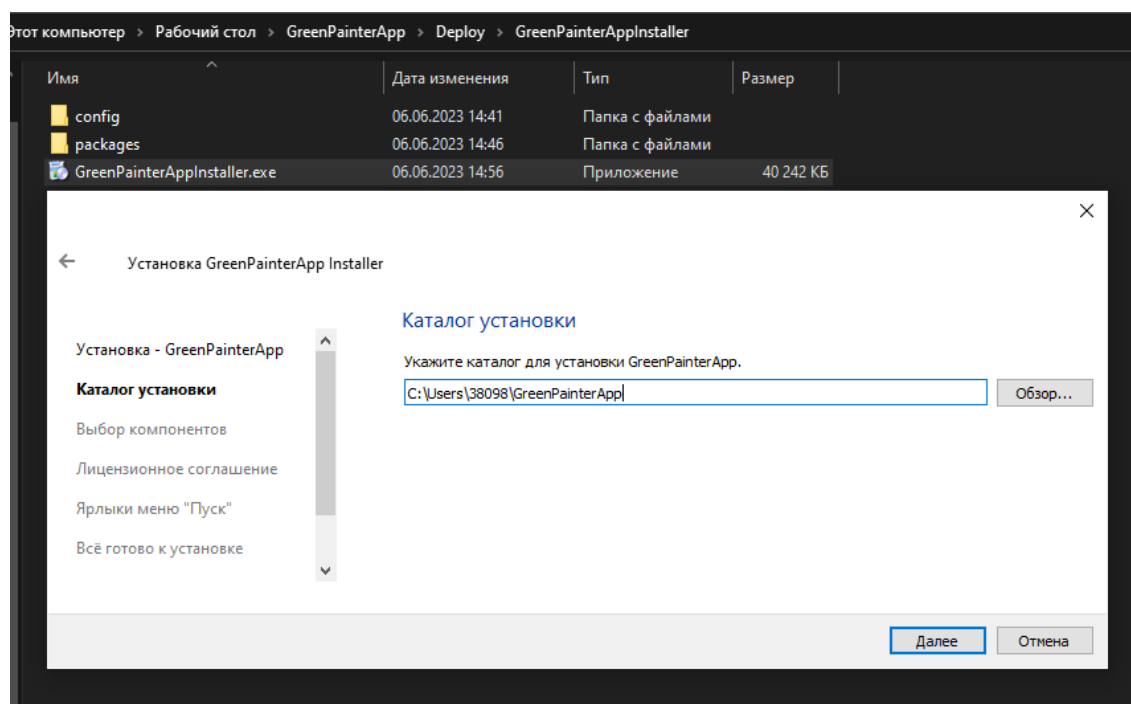


Рисунок 3.2 – Встановлення додатку.

Після побудови релізної версії систему можна переносити на будь-який девайс, що відповідає цільовій операційній системі збірки, адже інструментарій має підтримку багатьох платформ [31].

Якщо збірку було виконано під Windows, редактор комфортно запускатися на будь-якій системі, конфігурація якої задовольняє наступним мінімальним вимогам:

- процесор з тактовою частотою 2.4 ГГц або вище;
- 4 RAM;
- 1 ГБ вільного місця на жорсткому диску;
- Графічний пристрій з підтримкою DirectX 9 (бажано DirectX 11);

Таким чином, розроблена система матиме достатню потужність для виконання усіх цілей у найшвидший час.

3.2 Інтерфейс та порядок роботи

Перейдемо до огляду інтерфейсу та робочого процесу нашого растрового графічного редактора. Ми розглянемо різні скріншоти і надамо короткі описи до кожного з них, підкреслюючи ключові особливості та функціональні можливості програмного продукту. Ці скріншоти демонструють елементи користувацького інтерфейсу, з якими користувачі взаємодітимуть під час роботи.

Відкриття редактора: користувач запускає графічний редактор на своєму комп'ютері. (рис.3.3). При відкритті графічного редактора користувач зазвичай бачить інтерфейс програми, який дозволяє йому взаємодіяти з редактором і працювати з зображеннями.

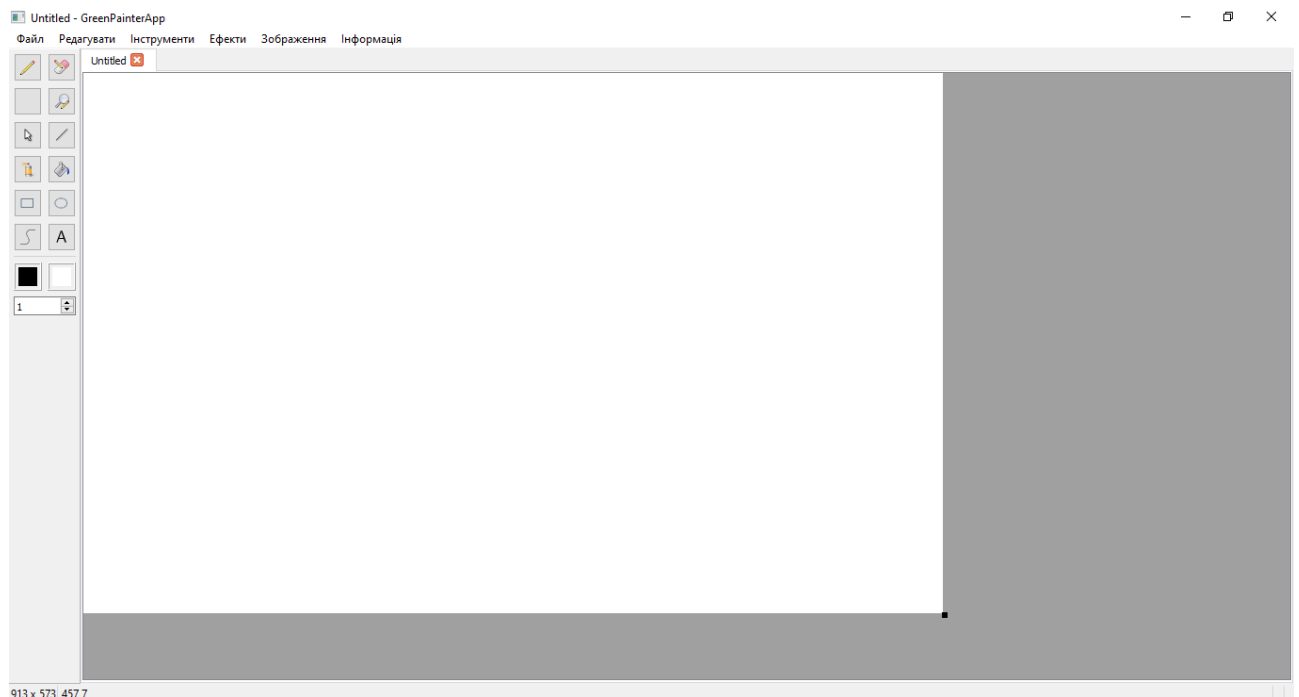


Рисунок 3.3 – Інтерфейс програми.

Основні елементи, які можуть бути присутніми на екрані, включають:

- Меню (рис.3.4): вгорі вікна редактора можуть бути розташовані різні меню, які надають доступ до різних функцій та опцій редактора, таких як 'Файл', 'Редагувати', 'Інструменти', 'Ефекти' та інші. Ці меню містять підменю з різними командами, які користувач може обрати;

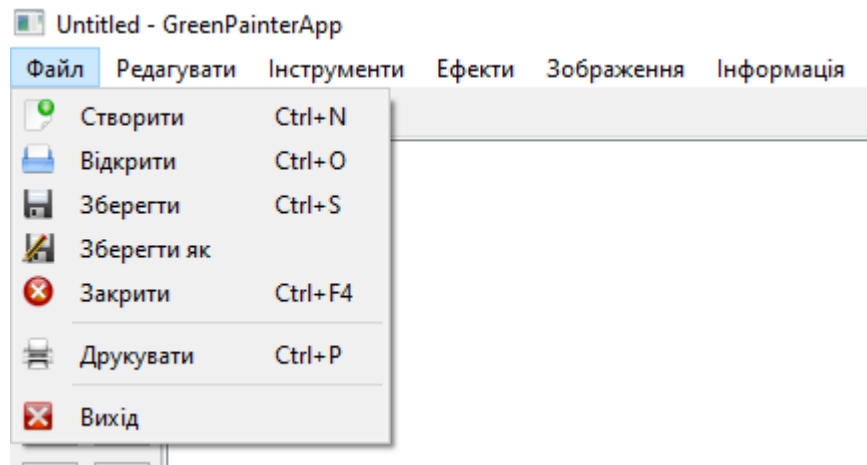


Рисунок 3.4 – Меню доступу до різних функцій та опцій редактора.

- Панель інструментів (рис.3.5): панель інструментів знаходиться збоку вікна редактора і містить набір інструментів для редагування зображення. Це можуть бути пензлі, спреї, гумка, заливка, текст, переміщення та інші інструменти;

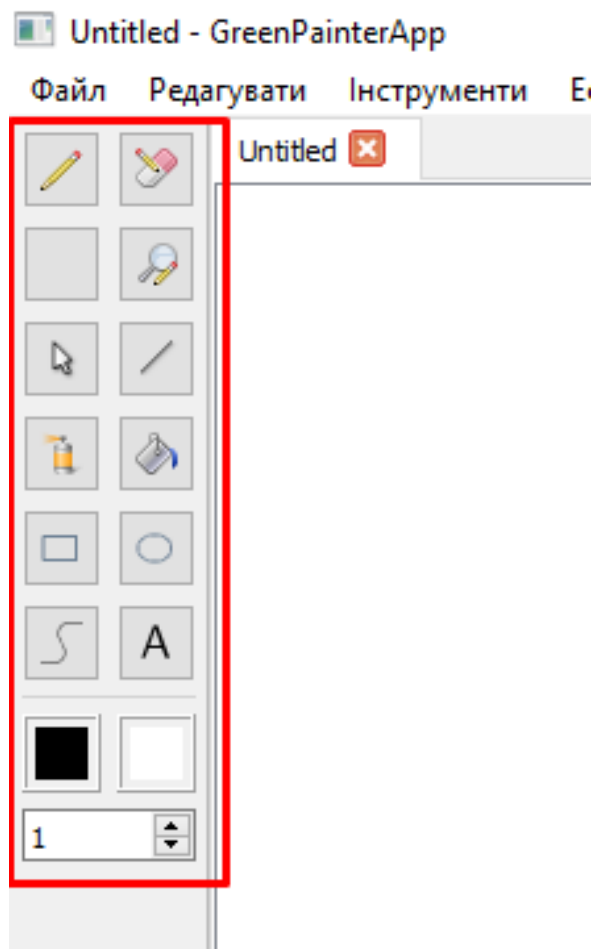


Рисунок 3.5 – Панель інструментів.

- Робоче полотно: центральна область вікна редактора відображає робоче полотно, де користувач може переглядати та редагувати зображення;
- Вікно налаштувань (рис.3.6): редактор має панель налаштувань, де користувач може змінювати різні параметри, такі як розмір полотна при завантаженні, автозбереження, глибина історії, створення гарячих клавіш;

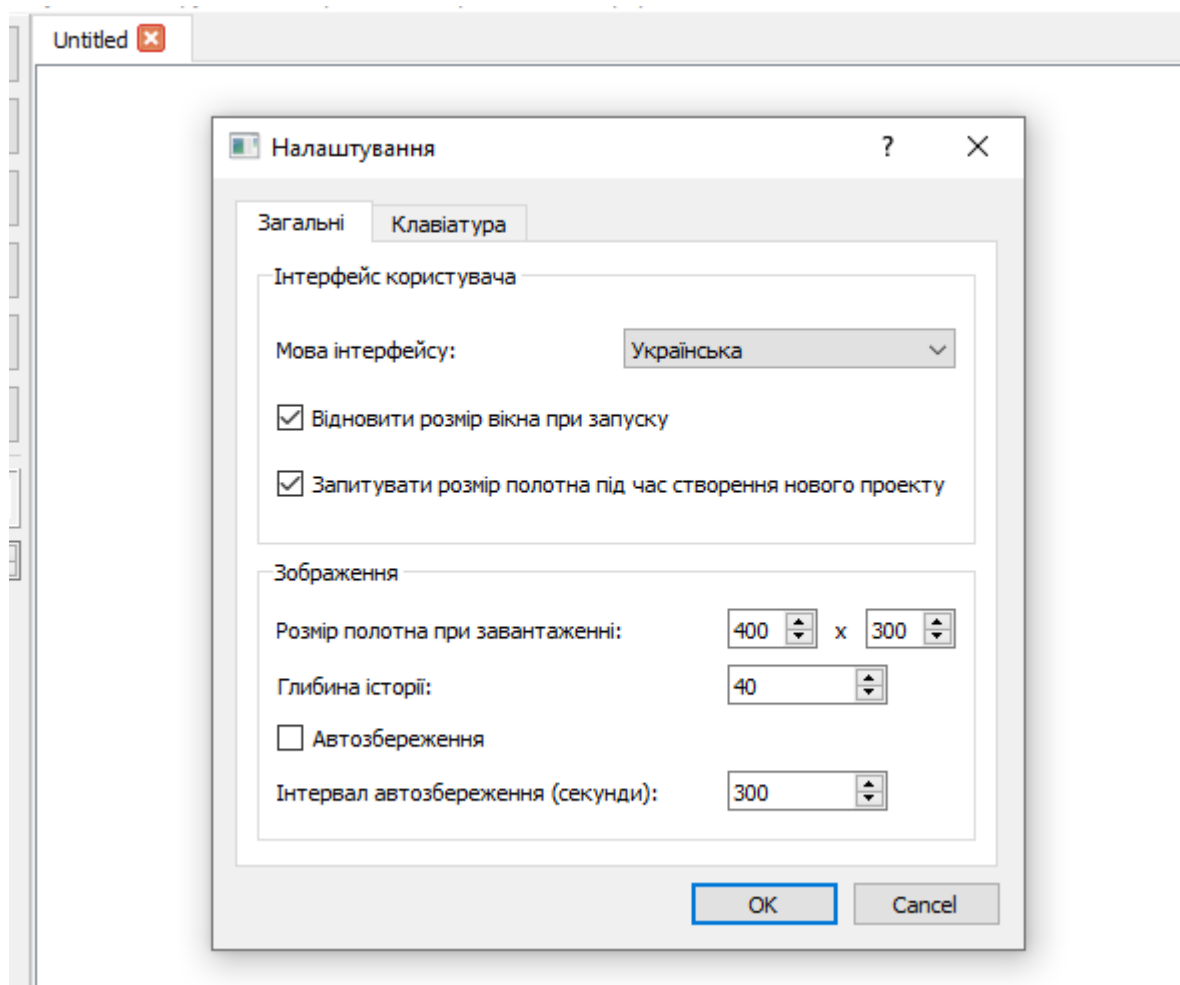


Рисунок 3.6 – Вікно налаштувань.

- Інформаційні панелі (рис.3.7): редактор має інформаційні панелі, які відображають додаткову інформацію про сам редактор;

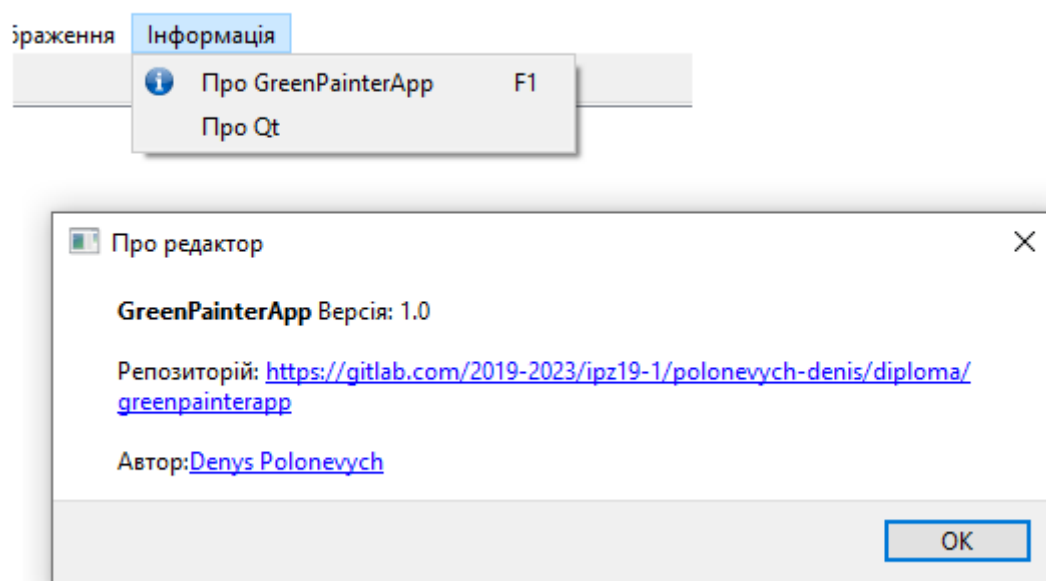


Рисунок 3.7 – Інформаційні панелі.

Загальною метою дизайну інтерфейсу графічного редактора є забезпечення зручного та ефективного способу взаємодії користувача з редактором та зручного доступу до всіх необхідних функцій та опцій.

Порядок роботи графічного редактору може варіюватися в залежності від конкретної задачі користувача. Однак, загальний порядок роботи може бути наступним:

1. Відкриття редактора: Користувач запускає графічний редактор на своєму комп'ютері;
2. Створення нового проекту або відкриття існуючого файлу (рис.3.8): Користувач має можливість створити новий проект або відкрити існуючий файл зображення, з яким він бажає працювати;

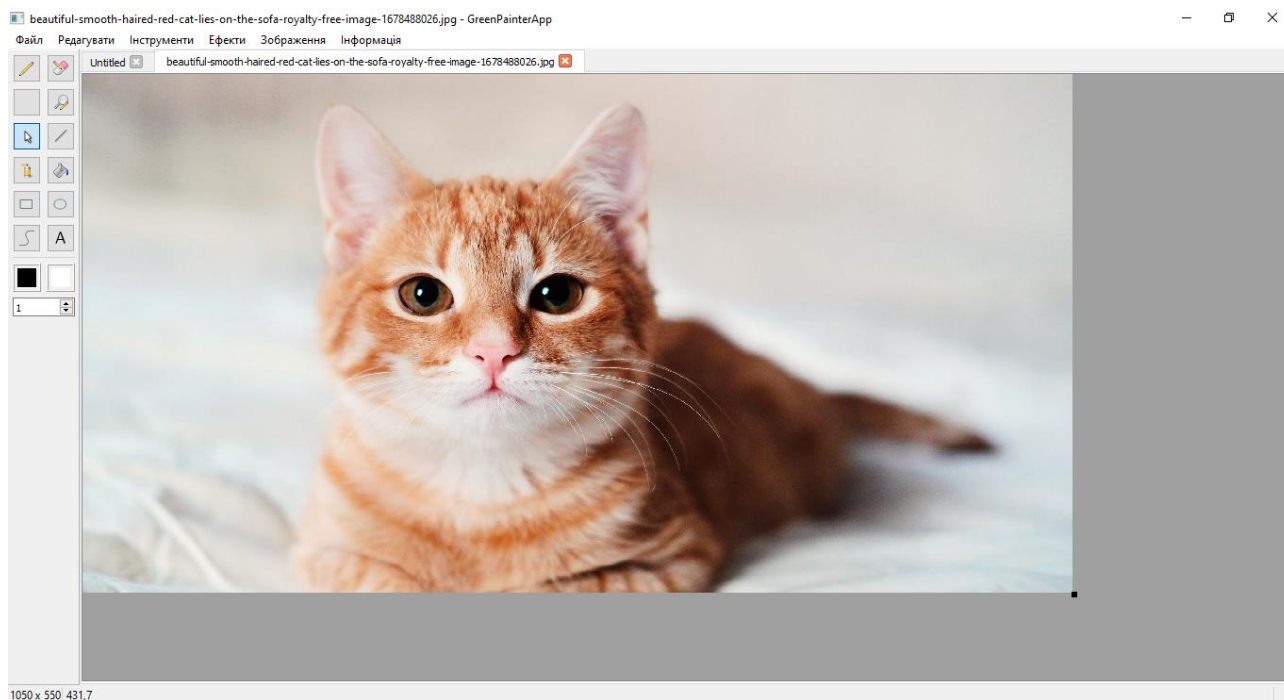


Рисунок 3.8 – Відкриття існуючого файлу у новому вікні.

3. Редагування зображення (рис.3.9): користувач може виконувати різні операції з редагування зображення, такі як малювання, розміщення об'єктів, зміна кольорів, обрізка тощо;

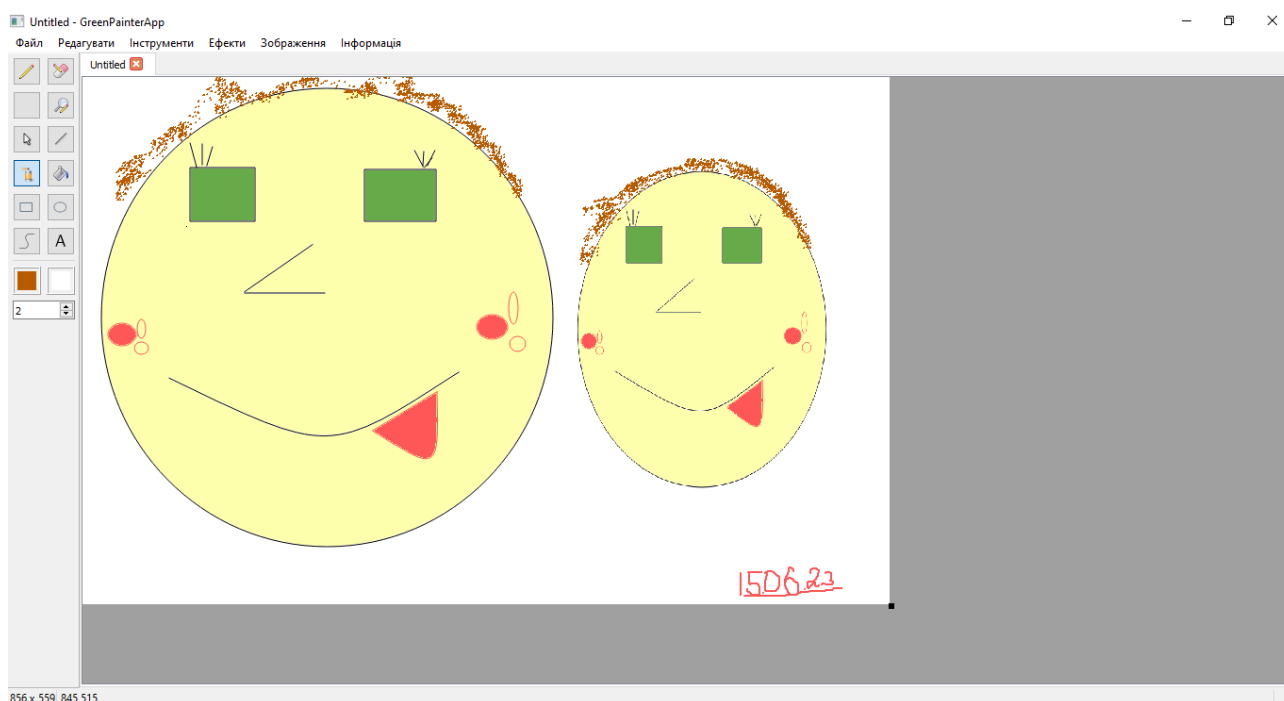


Рисунок 3.9 – Редагування зображення.

4. Редагування зображення: застосування ефектів;

Редактор має можливість застосовувати ефекти наступного типу: чорно-білий (рис.3.10), негатив (рис.3.11), бінаризація (рис.3.12), гаусів фільтр (рис.3.13), гамма (рис.3.14), різкість (рис.3.15), власне розмиття (рис.3.16).

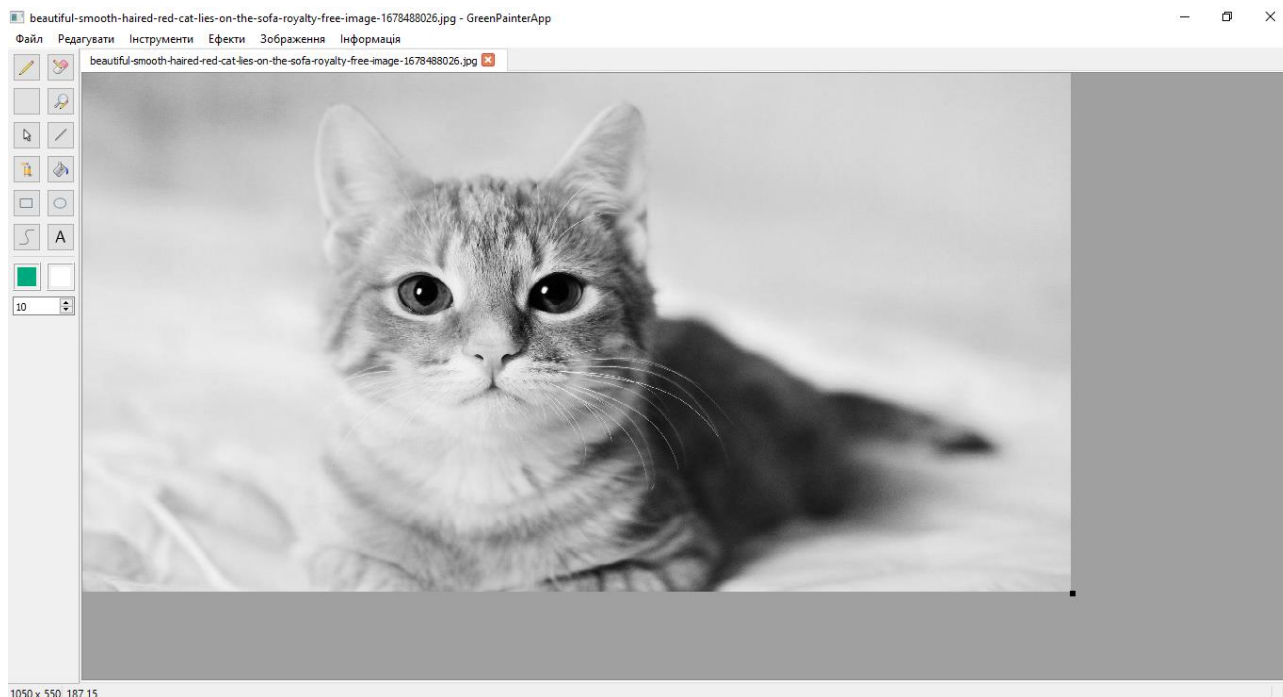


Рисунок 3.10 – Застосування ефекту ‘чорно-білий’.

Чорно-білий фільтр є одним з найпоширеніших фільтрів, який застосовується у графічних редакторах для перетворення зображень з кольорового простору в чорно-білий простір. Цей фільтр зводить кольорову інформацію до відтінків сірого, використовуючи методи конвертації.

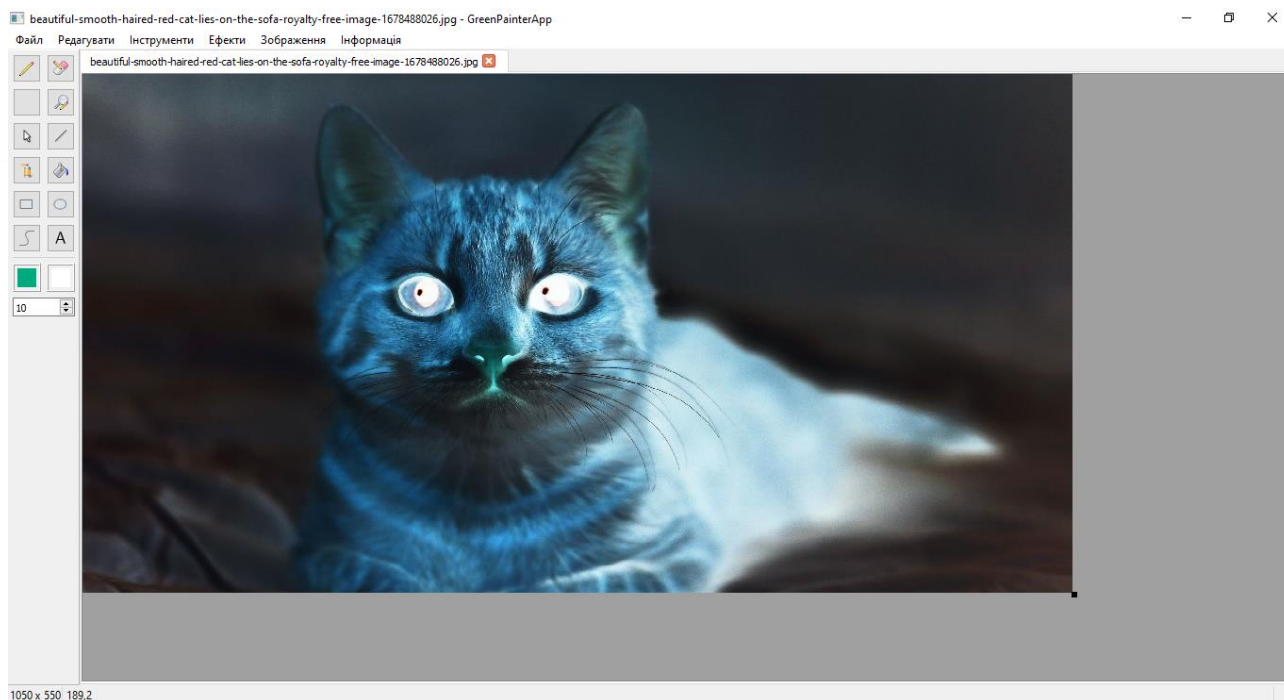


Рисунок 3.11 – Застосування ефекту ‘негатив’.

Негативний фільтр є одним з основних ефектів, що використовуються у графічних редакторах для створення ефекту негативу зображення. Він змінює кольори пікселів, інвертуючи їх значення, тобто темні кольорові відтінки стають світлими, а світлі - темними.

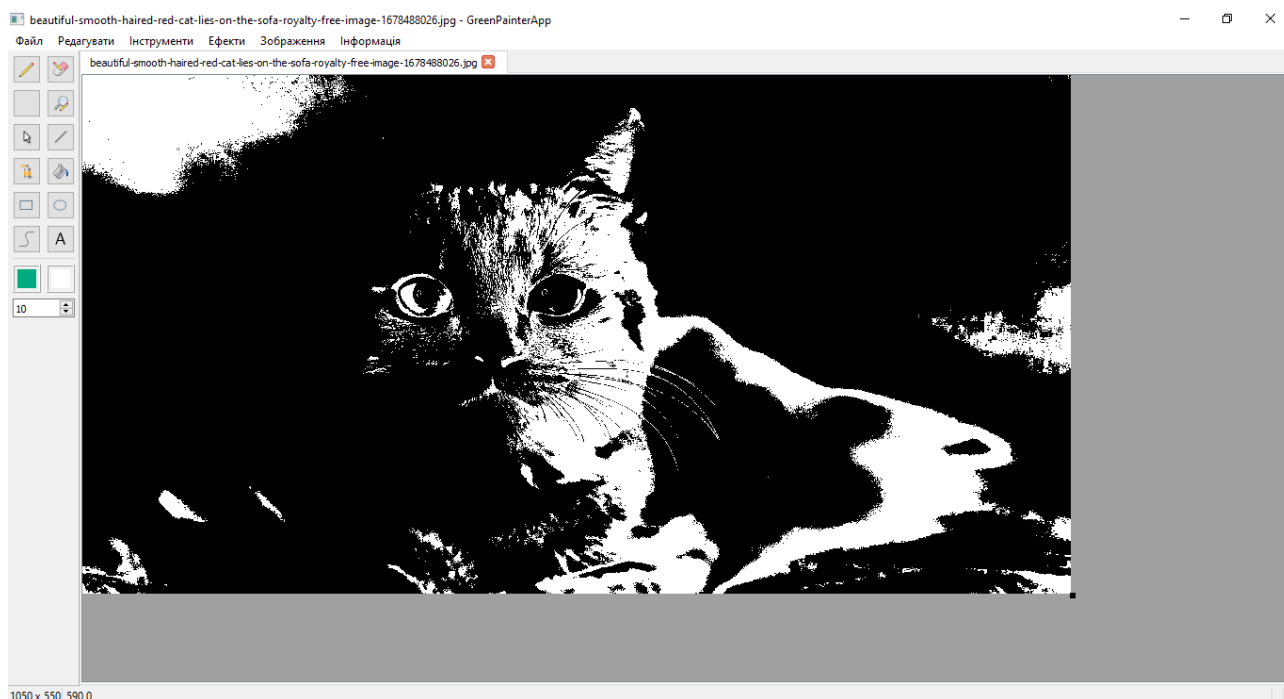


Рисунок 3.12 – Застосування ефекту ‘бінаризація’.

Бінаризація є ефектом, що використовується у графічних редакторах для перетворення зображення на двохкольорову (чорно-білу) градацію, де пікселі залежно від їхнього яскравості або інтенсивності класифікуються як чорні або білі.

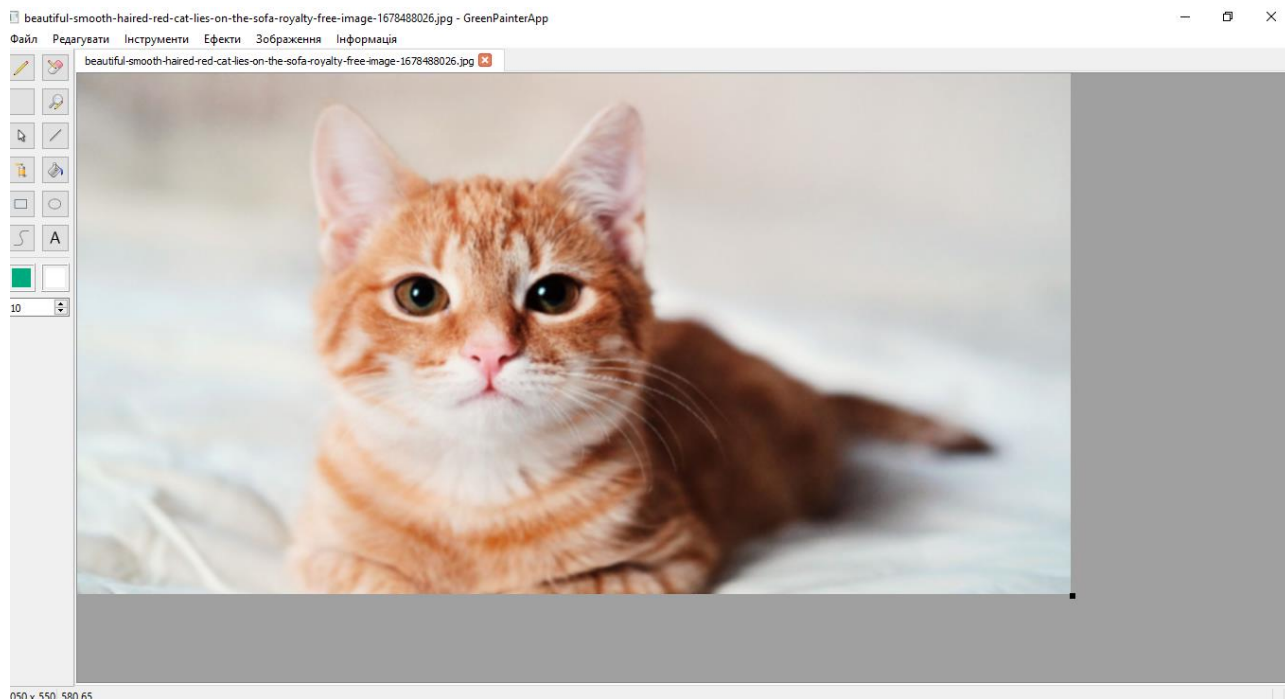


Рисунок 3.13 – Застосування ефекту ‘гаусовий фільтр’.

Гаусовий фільтр є одним з найпоширеніших фільтрів, що використовуються у графічних редакторах для згладжування зображень та зменшення шуму. Цей фільтр базується на математичній функції Гауса, яка створює розмиття зображення шляхом використання вагованих середніх значень пікселів.

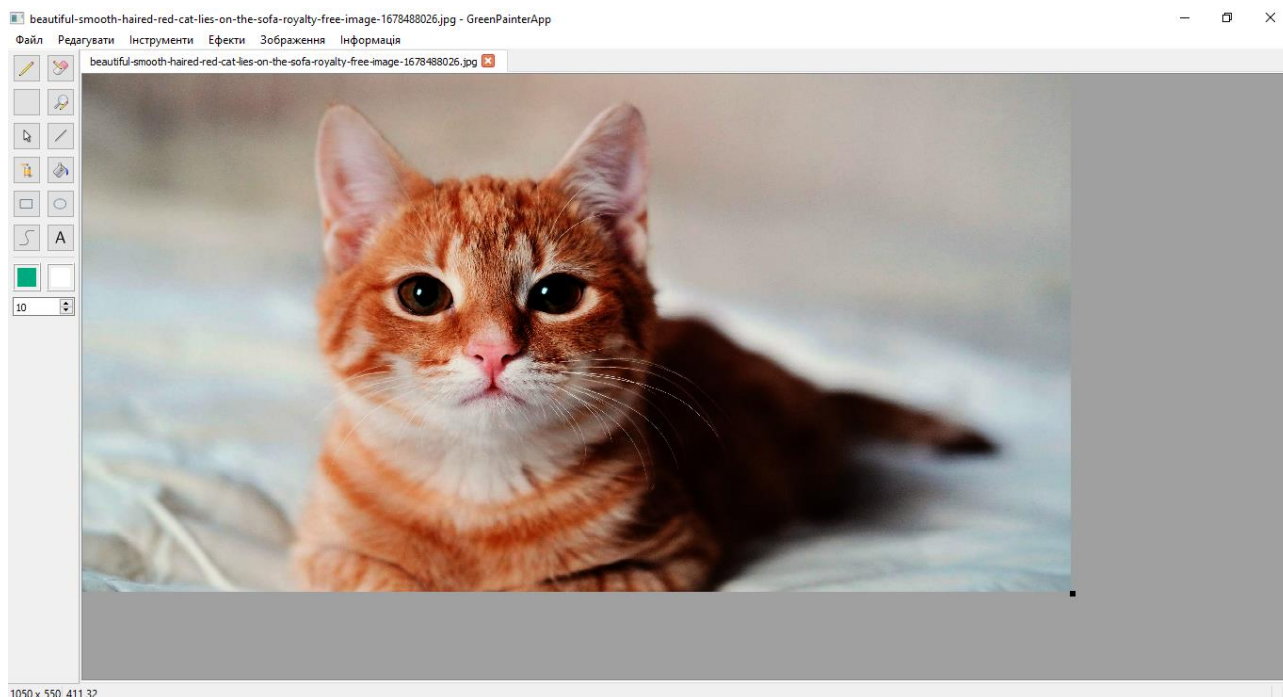


Рисунок 3.14 – Застосування ефекту 'гамма'.

Гамма-фільтр є одним з ефектів, що використовуються у графічних редакторах для корекції яскравості та контрастності зображення за допомогою гамма-корекції. Він змінює гамма-коефіцієнт, який впливає на спосіб, яким зображення відображається на екрані, змінюючи яскравість та контрастність різних тонів.

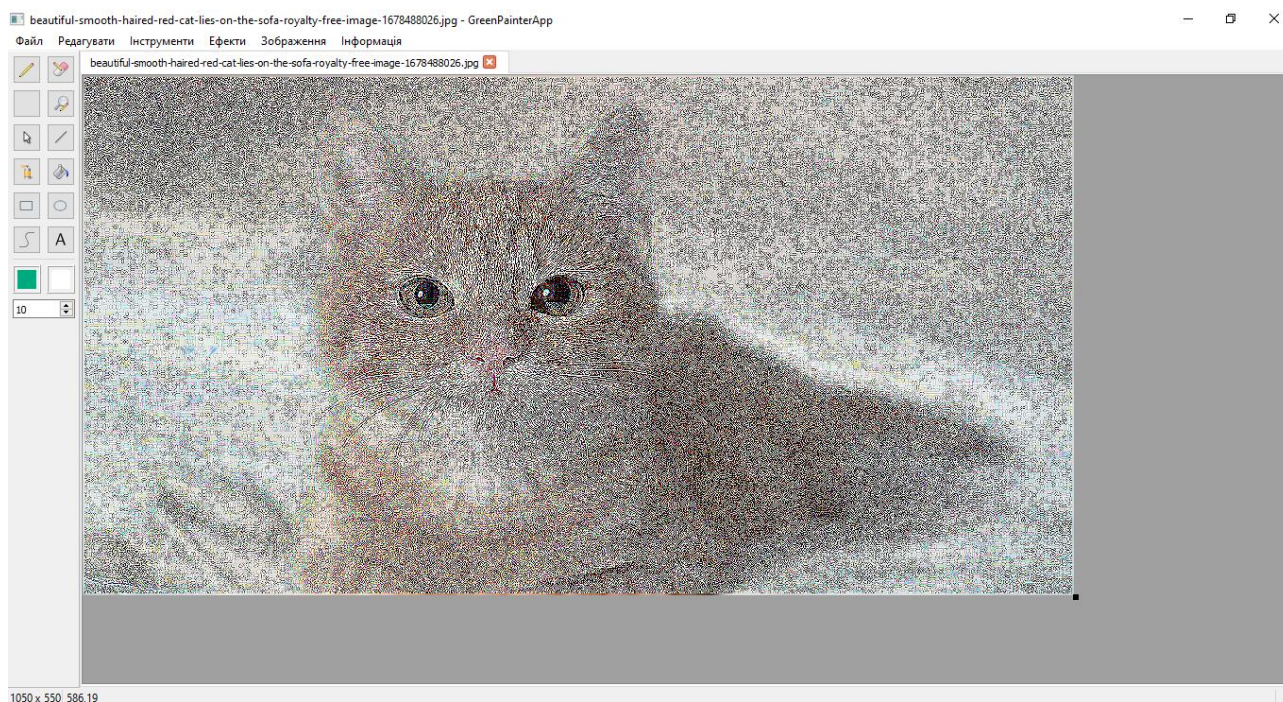


Рисунок 3.15 – Застосування ефекту 'різкість'.

Ефект різкості в графічних редакторах використовується для підсилення деталей та контурів зображення, що призводить до вигляду більш чіткого та виразного зображення. Цей ефект робить краї об'єктів більш різкими шляхом підсилення контрасту між сусідніми пікселями.

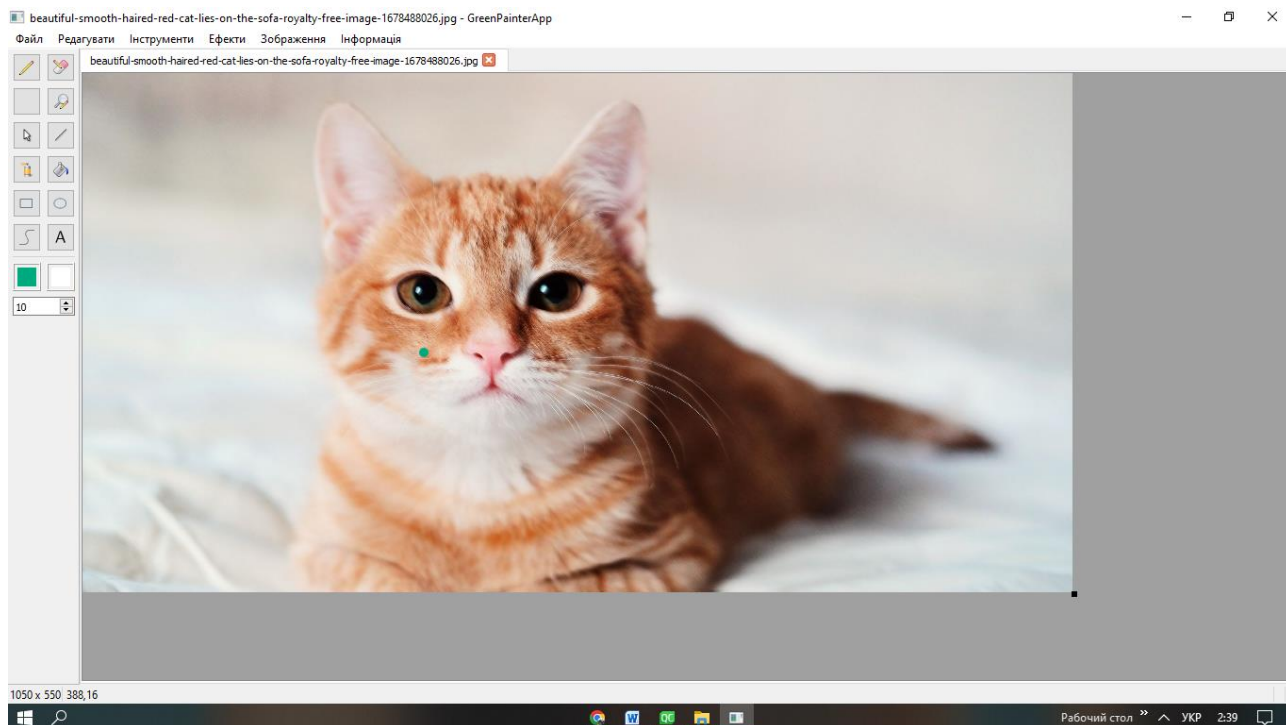


Рисунок 3.16 – Застосування ефекту ‘власний’.

5. Застосування тексту (рис.3.17): користувач може додавати текстові блоки, заголовки, підписи;

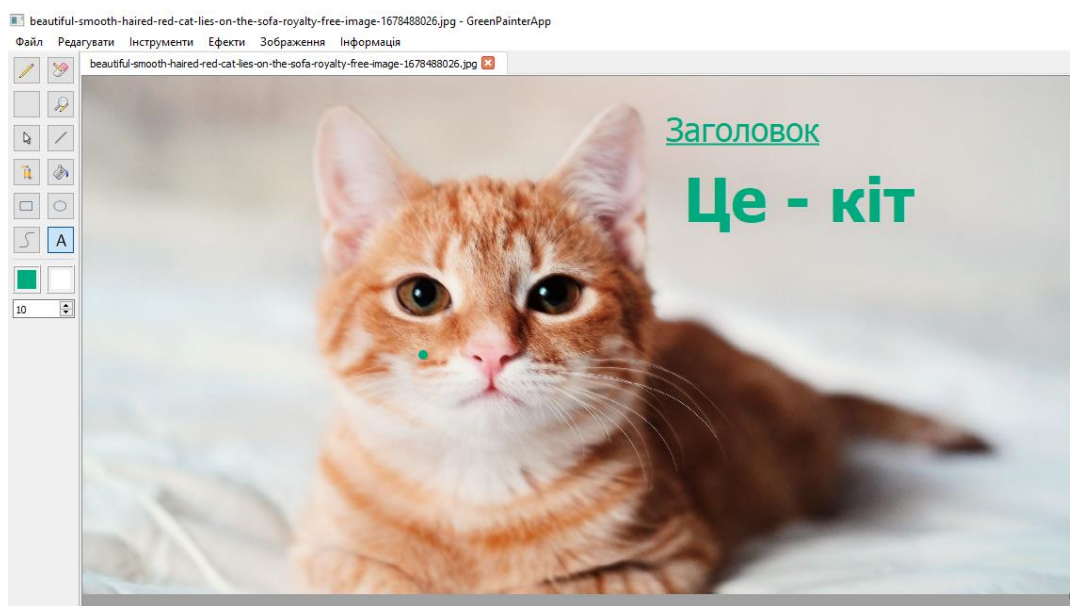


Рисунок 3.17 – Застосування тексту.

6. Збереження зображення (рис.3.18): після завершення редагування зображення користувач має можливість зберегти його у вибраному форматі, такому як JPEG, PNG, TIFF тощо;

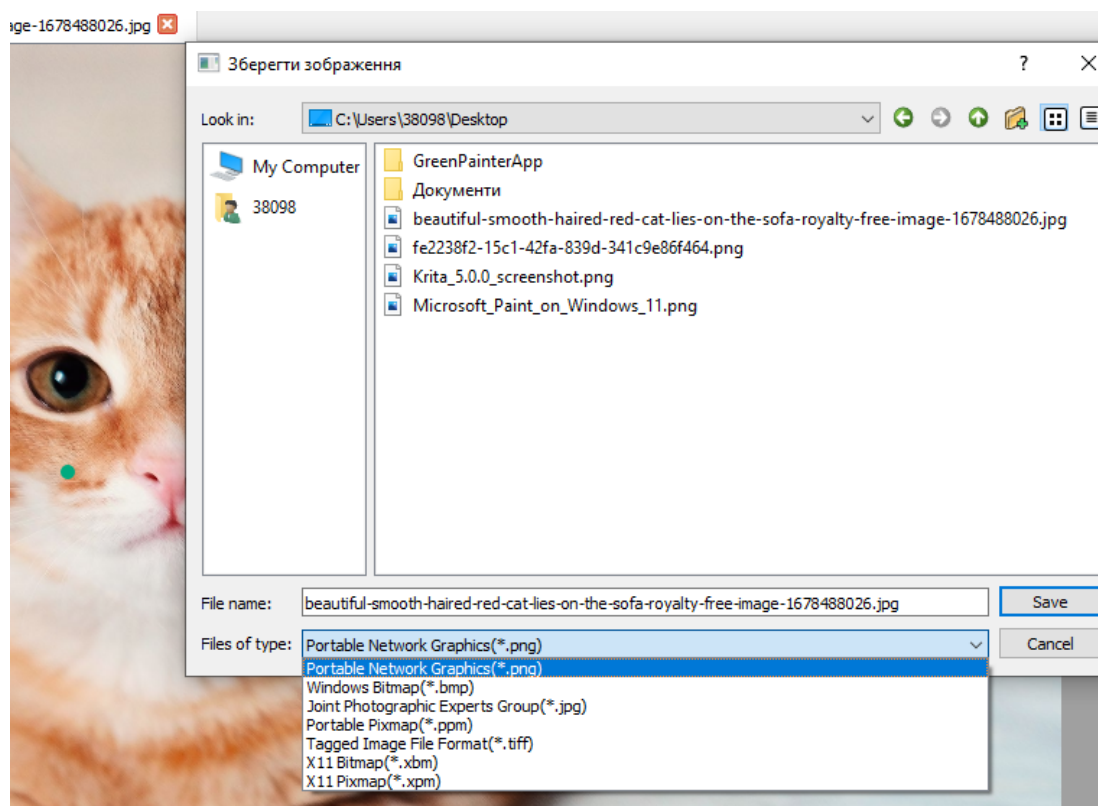


Рисунок 3.18 – Збереження зображення.

7. Друк (рис.3.19): Користувач може друкувати зображення безпосередньо з редактора;

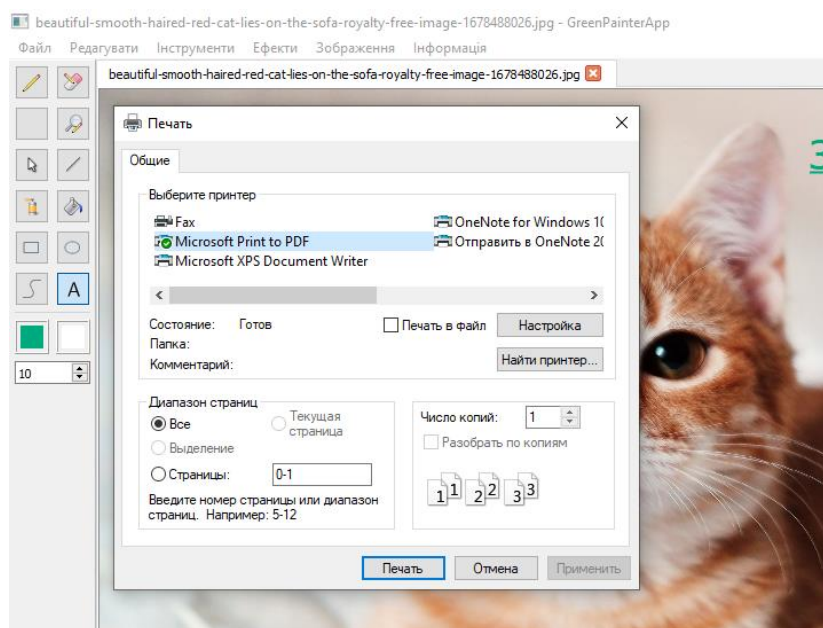


Рисунок 3.19 – Друк зображення.

8. Трансформації зображення (рис.3.20): включають різні операції для зміни розміру, повороту;

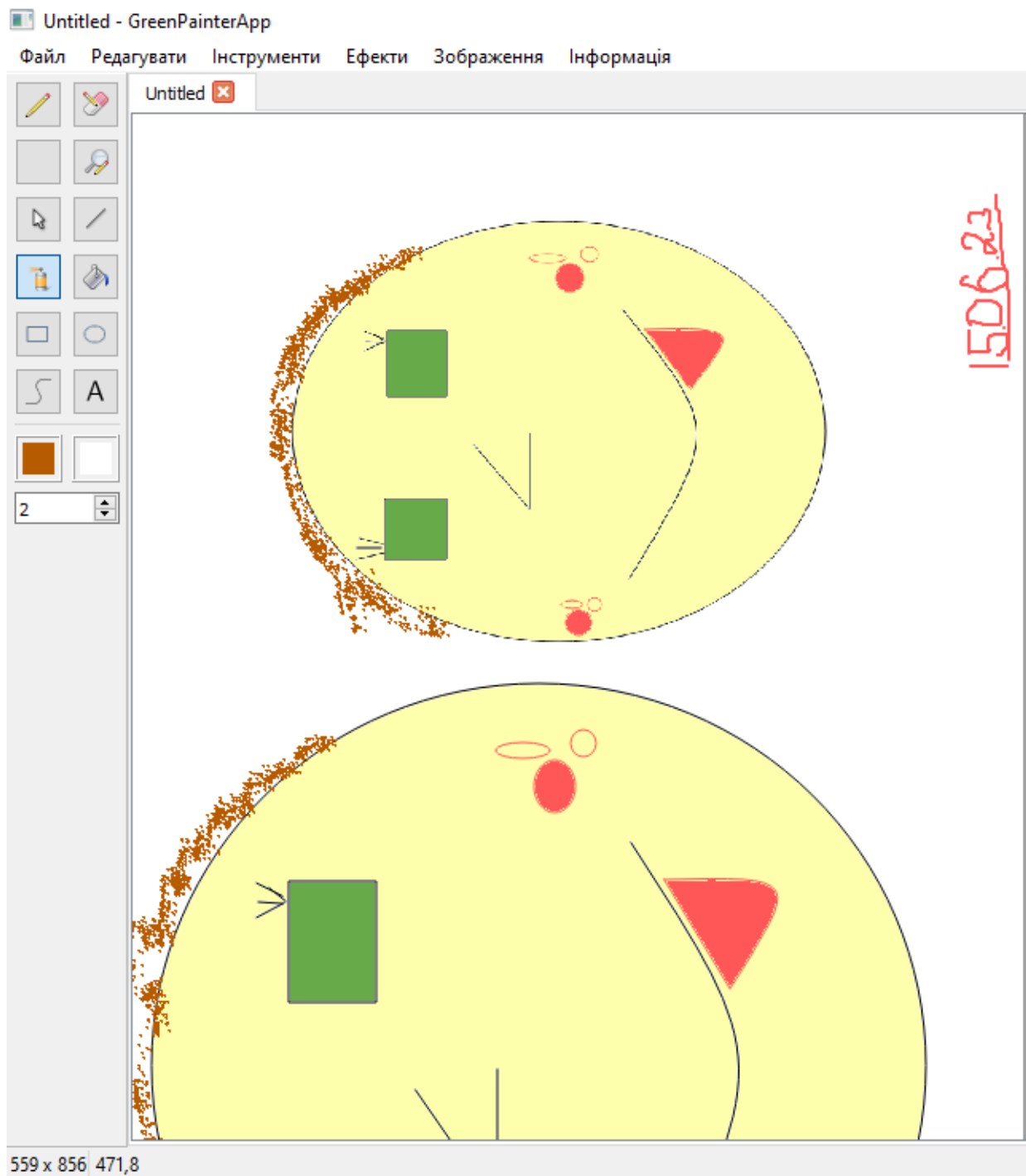


Рисунок 3.20 – Трансформації зображення.

9. Створення зображень з прозорим фоном (рис. 3.21);

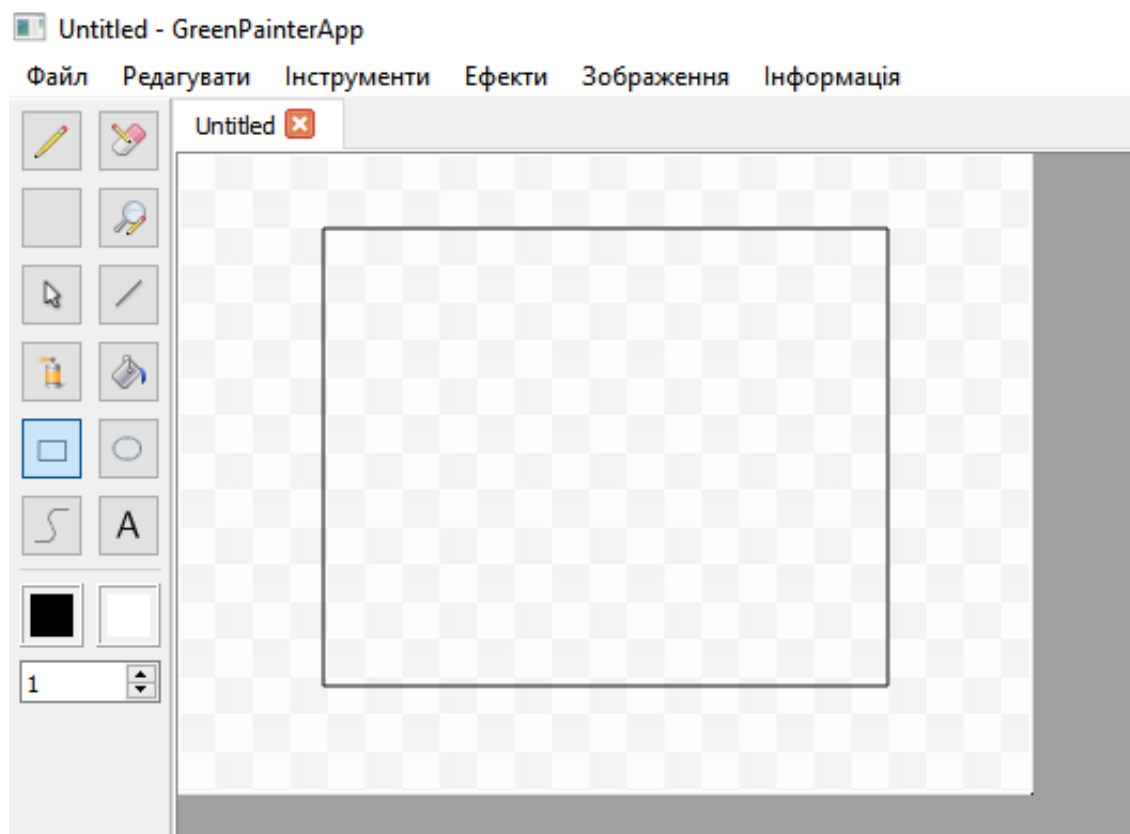


Рисунок 3.21 – Створення зображень з прозорим фоном.

10. Створення гарячих клавіш (рис.3.22): короткі комбінації клавіш, які використовуються для виконання певних дій або запуску певних функцій в програмі без необхідності використовувати мишу або навігацію по меню;

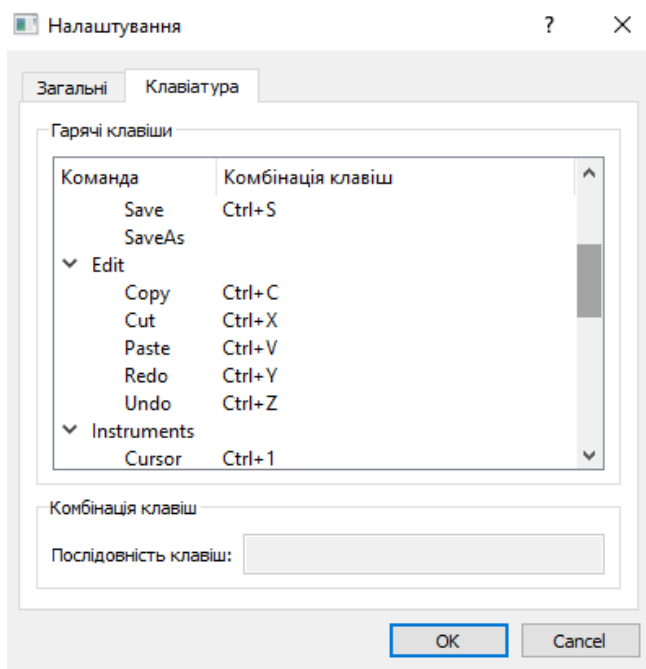


Рисунок 3.22 – Редагування гарячих клавіш.

11. Закриття редактора (рис.3.23): після завершення роботи з редактором користувач може закрити програму або залишити її активною для подальшої роботи;

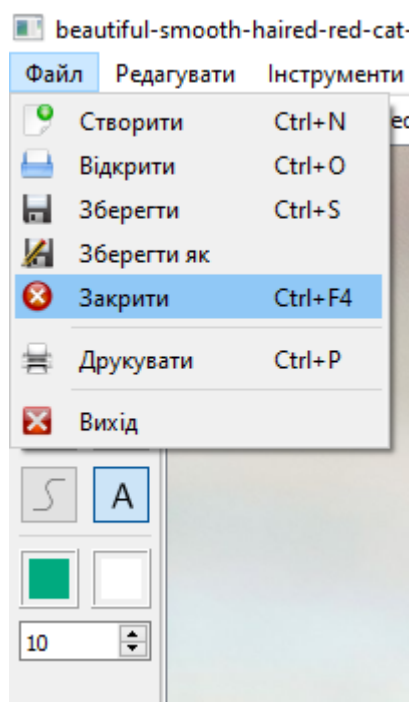


Рисунок 3.23 – Закриття редактора.

У цьому підрозділі ми розглянули інтерфейс і робочий процес растрового графічного редактора. Ці візуальні матеріали дають уявлення про зручний дизайн і функціональні можливості, які пропонує наша система, дозволяючи користувачам ефективно користуватись графічним редактором.

3.3 Тестування роботи растрового графічного редактора

Тестування роботи растрового графічного редактора може включати декілька етапів, щоб перевірити різні аспекти його функціональності та продуктивності. Важливі аспекти, які потрібно врахувати під час тестування:

- Функціональність інструментів: тестування кожного інструмента окремо; можливість змінити параметри; взаємодія з іншими інструментами; тестування реакції на обробку системи та швидкість інструментів;
- Підтримка різних форматів зображень: відкриття зображень у різних форматах, таких як JPEG, PNG;
- Редагування та маніпуляція зображеннями: виконання усього функціоналу редактора;
- Продуктивність та стабільність: розподілення ресурсів комп'ютера, оптимізація різних алгоритмів, наприклад ефектів;
- Інтерфейс користувача;
- Контроль якості зображення: збереження в оригінальному та різних форматах, перевірка деталей і кольорів;

Проводячи тестування роботи програмного забезпечення, ми можемо переконатися, що розроблюваний продукт, відповідає необхідним функціональним можливостям і оптимально працює в різних умовах. Ці тести допомагають виявити та усунути будь-які потенційні проблеми або помилки, забезпечуючи стабільну роботу редактора.

Висновки до розділу 3

Загалом третій розділ був присвячений інтерфейсу та порядку роботи з растровим графічним редактором. Він охоплює три основні напрямки: встановлення та налаштування параметрів додатку, структуру інтерфейсу та тестування роботи програмного комплексу.

У першій частині було розглянуто процедуру встановлення та налаштування параметрів системи. Дотримуючись наданих інструкцій, користувачі можуть ефективно встановити та налаштувати параметри системи, що дозволить їм адаптувати її до конкретних цілей.

Друга частина розділу була присвячена структурі інтерфейсу растрового графічного редактора. У ній було представлено огляд різних компонентів і модулів, що складають інтерфейс системи, пропонуючи користувачам комплексне розуміння того, як вони взаємодіють з системою. Структура інтерфейсу забезпечує зручний та інтуїтивно зрозумілий досвід роботи, що сприяє ефективній роботі з редактором.

Нарешті, було розглянуто такий важливий аспект, як тестування роботи програмного комплексу. Було підкреслено важливість проведення тестів для забезпечення функціональності, надійності та продуктивності системи. Ретельне тестування роботи системи дозволяє виявити та вирішити потенційні проблеми, гарантуючи якісний та надійний програмний продукт для редагування графіки.

ВИСНОВКИ

Даний проект охопив комплексний аналіз, розробку та впровадження растрового графічного редактора. Проект був розділений на три основні розділи, кожен з яких стосувався конкретних аспектів процесу розробки і завершувався цінними висновками.

Перший розділ був присвячений первинному аналізу інформаційних технологій. Розділ розпочався з чіткої постановки завдання, окреслення цілей та завдань проекту. Було проведено глибокий аналіз аналогічних програмних продуктів з урахуванням їх особливостей, функціональних можливостей та придатності до конкретних вимог підприємства.

Крім того, в розділі було розглянуто критично важливе рішення щодо вибору архітектури для розробки редактора. Було оцінено різні архітектурні варіанти, враховуючи такі фактори, як масштабованість, продуктивність і зручність обслуговування.

Другий розділ був присвячений розробці та впровадженню редактора. Розділ розпочався з визначення варіантів використання, виявлення різних сценаріїв і функціональних можливостей, які повинна підтримувати система. Було визначено об'єктно-орієнтовану структуру системи, висвітлено взаємозв'язки між різними компонентами та їх ролі у виконанні сценаріїв використання.

Одним з ключових аспектів розділу була розробка та реалізація алгоритмів системи. Ретельна увага була приділена алгоритмам, необхідним для виконання таких завдань, як застосування графічних перетворень зображення. Алгоритми були розроблені з метою оптимізації ефективності, точності та гнучкості відповідно до конкретних потреб користувача.

Третій розділ був присвячений інтерфейсу та процедурі роботи растровим графічним редактором. Він охоплював рекомендації щодо встановлення та налаштування параметрів системи, що забезпечило плавний і безпроблемний процес налаштування. Структура інтерфейсу системи була ретельно

розроблена, щоб запропонувати користувачам інтуїтивно зрозумілу та зручну роботу.

На додаток до дизайну інтерфейсу, тестування роботи програмного пакету було підкреслено як вирішальний крок у забезпеченні функціональності, надійності та продуктивності системи.

Таким чином, цей проект продемонстрував комплексний підхід до розробки растрового графічного редактора.

Реалізований програмний комплекс має весь необхідний функціонал для швидкого редагування фотографій. Завдяки тому, редактор здатний вирішувати найрізноманітніші завдання, тому програмний комплекс має великі можливості для розширення в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Computer Graphics: Principles and Practice / John Hughes, 2013 — O'Reilly
2. What Is a Photo Editor and How to Become One? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ziprecruiter.com/career/Photo-Editor/What-Is-How-to-Become>
3. Qt Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://doc.qt.io/>
4. Use the Graphic editor - Kameleoon - User Manual. [Електронний ресурс] – Режим доступу до ресурсу: <https://help.kameleoon.com/use-graphic-editor/>
5. How to Deploy Your Qt Cross-Platform Applications? [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/swlh/how-to-deploy-your-qt-cross-platform-applications-to-windows-operating-system-by-using-windeployqt>
6. C++ Gui Programming With Qt4 (with Cd) / Марк Саммерфілд, 2008 — O'Reilly
7. Microsoft Paint. [Електронний ресурс] – Режим доступу до ресурсу: <https://mspaint.humanhead.com/#local:bb62c35f2d414>
8. Adobe Photoshop. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.adobe.com/ua/products/photoshop.html>
9. Krita. [Електронний ресурс] – Режим доступу до ресурсу: <https://krita.org/>
10. Комплексний підхід до створення систем. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.bmsserv.net/ua/products/solutions/72-complexapproach.html>
11. The C++ Programming Language / Б'ярн Страуструп, 1986. - 1368 с — Packt Publishing
12. Paint System: Qt GUI 6.5.1 [Електронний ресурс] – Режим доступу до ресурсу: <https://doc.qt.io/qt-6/paintsystem.html>
13. Критичні ситуації при виділенні пам'яті. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.bestprog.net/uk/2018/09/25/dynamic-and-static-allocation-of-memory-advantages-and-disadvantages-allocating->

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						59

memory-for-single-variables-by-the-operators-new-and-delete-possible-critical-situations-when-allocating-memory-init_ua/

14. Розробка кроссплатформенних додатків. [Електронний ресурс] – Режим доступу до ресурсу: <https://webcase.com.ua/uk/krossplatformennaya-razrabotka/>
15. Object-oriented programming in C++ / Річард Джонсонбо, 1995. 476с – О'Рейлі
16. Hands-On Design Patterns with C++ / Fedor G. Pikus , 2019. 534с – О'Рейлі
17. Qt for Open Source Development. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.qt.io/download-open-source>
18. Аналіз вимог до програмного забезпечення / Юрій Грицюк, 2018 — Yakaboo
19. Що таке атрибути якості програмного забезпечення? [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.myservername.com/what-are-quality-attributes>
20. Performance Optimization in Software Development [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/the-andela-way/performance-optimization-in-software-development-ae7952ab885e>
21. Тестування продуктивності. [Електронний ресурс] – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/testuvannya-produktivnosti/>
22. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>
23. UML Class Diagram. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>
24. Фабричний метод. [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/factory-method/cpp/example>
25. QImage Class. [Електронний ресурс] – Режим доступу до ресурсу: <https://doc.qt.io/qt-6/qimage.html>

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						60

26. What is Activity Diagram? [Электронный ресурс] – Режим доступа до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>
27. What is Sequence Diagram? [Электронный ресурс] – Режим доступа до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>
28. Understanding Otsu's Method for Image Segmentation. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.baeldung.com/cs/otsu-segmentation>
29. Image Processing – Bilinear Interpolation. [Электронный ресурс] – Режим доступа до ресурсу: <https://theailearner.com/2018/12/29/image-processing-bilinear-interpolation/>
30. Gaussian Blur. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.w3.org/Talks/2012/0125-HTML-Tehran/Gaussian.xhtml>
31. Specifying Build Settings. [Электронный ресурс] – Режим доступа до ресурсу: <https://doc.qt.io/qtcreator/creator-build-settings.html>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						62

Реалізація класу, що відповідає за інструмент ‘Текст’.

Лістинг textinstrument.cpp:

```
#include "textinstrument.h"
#include "../imagearea.h"
#include "../datasingleton.h"
#include "../undocommand.h"
#include "../dialogs/textdialog.h"

#include <QPainter>

TextInstrument::TextInstrument(QObject* parent) :
    AbstractSelection(parent)
{
    mText = QString();
    mIsEdited = false;
}

void TextInstrument::updateText(ImageArea* imageArea, QString textString)
{
    mText = textString;
    imageArea->setImage(mImageCopy);
    if (!mIsEdited)
    {
        makeUndoCommand(*imageArea);
        mIsEdited = true;
    }
    paint(*imageArea);
    drawBorder(*imageArea);
}

void TextInstrument::resize(ImageArea& imageArea)
{
    paint(imageArea);
}

void TextInstrument::move(ImageArea& imageArea)
{
    paint(imageArea);
}

void TextInstrument::completeSelection(ImageArea& imageArea)
{
    TextDialog* td = new TextDialog(mText, &imageArea);
    connect(td, SIGNAL(textChanged(ImageArea*, QString)), this,
    SLOT(updateText(ImageArea*, QString)));
    connect(this, &TextInstrument::sendCloseTextDialog, td, &QDialog::accept);
    connect(td, &TextDialog::canceled, this, &TextInstrument::cancel);
    td->setAttribute(Qt::WA_DeleteOnClose);
    td->show();
}

void TextInstrument::clear()
{
    mText = QString();
    mIsEdited = false;
    emit sendCloseTextDialog();
}

void TextInstrument::cancel(ImageArea* imageArea)
{
    mText = QString();
```

```

        clearSelection(*imageArea);
    }

void TextInstrument::paint(ImageArea& imageArea, bool, bool)
{
    if (mTopLeftPoint != mBottomRightPoint)
    {
        QPainter painter(imageArea.getImage());
        painter.setRenderHint(QPainter::Antialiasing);
        painter.setRenderHint(QPainter::TextAntialiasing);
        painter.setPen(QPen(DataSingleton::Instance()->getPrimaryColor()));
        painter.setFont(DataSingleton::Instance()->getTextFont());
        painter.drawText(QRect(mTopLeftPoint, mBottomRightPoint), mText);
        painter.end();
        imageArea.setEdited(true);
        imageArea.update();
    }
}

void TextInstrument::showMenu(ImageArea& imageArea)
{
    emit sendCloseTextDialog();
    completeSelection(imageArea);
}

```

ДОДАТОК Б

Файл заголовка, що містить визначення класу та функцій реалізації інструменту ‘Текст’.

Лістинг textinstrument.h:

```

#ifndef TEXTINSTRUMENT_H
#define TEXTINSTRUMENT_H

#include "abstractselection.h"
#include <QtCore/QObject>

/**
 * @brief Text instrument class.
 *
 */
class TextInstrument : public AbstractSelection
{
    Q_OBJECT
public:
    explicit TextInstrument(QObject* parent = 0);

private:
    void startAdjusting(ImageArea&);
    void startSelection(ImageArea&);
    void startResizing(ImageArea&);
    void startMoving(ImageArea&);
    void select(ImageArea&);
    void resize(ImageArea& imageArea);
    void move(ImageArea& imageArea);
    void completeSelection(ImageArea& imageArea);
    void completeResizing(ImageArea&);
    void completeMoving(ImageArea&);
    void clear();
    void paint(ImageArea& imageArea, bool = false, bool = false);
    void showMenu(ImageArea& imageArea);

    QString mText;
}

```

```

    bool mIsEdited;

signals:
    void sendCloseTextDialog();

private slots:
    void updateText(ImageArea*, QString);
    /**
     * @brief Clears text and remove selection.
     *
     * @param imageArea ImageArea for applying changes.
     */
    void cancel(ImageArea*);
};

#endif // TEXTINSTRUMENT_H

```

ДОДАТОК В

Реалізація класу, що відповідає за інструмент ‘Піпетка’.

Лістинг colorpickerpaletteinstrument.cpp:

```

#include "colorpickerpaletteinstrument.h"
#include "../imagearea.h"
#include "../datasingleton.h"
#include <QColorDialog>

ColorpickerPaletteInstrument::ColorpickerPaletteInstrument(QObject* parent) :
    AbstractInstrument(parent) {

}

ColorpickerPaletteInstrument::ColorpickerPaletteInstrument(AbstractInstrument* parent)
:
    AbstractInstrument(parent) {

}

void ColorpickerPaletteInstrument::mousePressEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::mouseMoveEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::mouseReleaseEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::showColorPalletteInstrument(ImageArea& imageArea) {
    imageArea.setIsPaint(true);
    DataSingleton* dataSingleton = DataSingleton::Instance();
    QList<QColor> colorStack = dataSingleton->getColorStack();
    for (int i = 0; i < colorStack.size(); i++) {
        QColorDialog::setCustomColor(colorStack.size() - i - 1, colorStack[i]);
    }
}

```

```

        QColor color = QColorDialog::getColor(
            dataSingleton->getPrimaryColor(),
            &imageArea, "Обрати колір",
            {
                QColorDialog::ShowAlphaChannel,
                QColorDialog::DontUseNativeDialog
            });
        dataSingleton->pushColorStack(color);
        dataSingleton->setPrimaryColor(color);
        imageArea.emitPrimaryColorView();
        imageArea.setIsPaint(false);
        imageArea.emitRestorePreviousInstrument();
    }

    void ColorpickerPaletteInstrument::paint(ImageArea& imageArea, bool isSecondaryColor,
        bool) {
        (void)imageArea;
        (void)isSecondaryColor;
    }

```

ДОДАТОК Г

Файл заголовка, що містить визначення класу та функцій реалізації інструменту ‘Піпетка’.

Лістинг colorpickerpaletteinstrument.h:

```

#include "colorpickerpaletteinstrument.h"
#include "../imagearea.h"
#include "../datasingleton.h"
#include <QColorDialog>

ColorpickerPaletteInstrument::ColorpickerPaletteInstrument(QObject* parent) :
    AbstractInstrument(parent) {

}

ColorpickerPaletteInstrument::ColorpickerPaletteInstrument(AbstractInstrument* parent)
:
    AbstractInstrument(parent) {

}

void ColorpickerPaletteInstrument::mousePressEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::mouseMoveEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::mouseReleaseEvent(QMouseEvent* event, ImageArea&
imageArea) {
    (void)imageArea;
    (void)event;
}

void ColorpickerPaletteInstrument::showColorPalletteInstrument(ImageArea& imageArea) {
    imageArea.setIsPaint(true);
}

```

```

DataSingleton* dataSingleton = DataSingleton::Instance();
QList<QColor> colorStack = dataSingleton->getColorStack();
for (int i = 0; i < colorStack.size(); i++) {
    QColorDialog::setCustomColor(colorStack.size() - i - 1, colorStack[i]);
}
QColor color = QColorDialog::getColor(
    dataSingleton->getPrimaryColor(),
    &imageArea, "Обрати колір",
    {
        QColorDialog::ShowAlphaChannel,
        QColorDialog::DontUseNativeDialog
    });
dataSingleton->pushColorStack(color);
dataSingleton->setPrimaryColor(color);
imageArea.emitPrimaryColorView();
imageArea.setIsPaint(false);
imageArea.emitRestorePreviousInstrument();
}

void ColorpickerPaletteInstrument::paint(ImageArea& imageArea, bool isSecondaryColor,
bool) {
    (void)imageArea;
    (void)isSecondaryColor;
}

```