МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа с изображениями

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Колованов Р.А.

Группа 9381

Тема работы: Работа с изображениями

Исходные данные:

- Язык программирования C++
- Фреймворк Qt
- Система Linux

Содержание пояснительной записки:

- Содержание
- Введение
- Формулировка задания
- Разработка программы
- Тестирование программы
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 01.03.2020

Дата сдачи реферата: 15.05.2020

Дата защиты реферата: 15.05.2020

Студент _____ Колованов Р.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

Задача курсовой работы состоит в разработке программы для работы с изображениями, в частности с изображениями формата PNG. В рамках курсовой работы было решено разработать графический интерфейс для программы – для его реализации использовался фреймворк Qt. Для считывания и сохранения изображения формата PNG была выбрана библиотека *libpng*.

Курсовая работа состоит из пояснительной записки и исходного кода разработанной программы.

В ходе работы была разработана программа для работы с изображениями формата PNG, имеющая графический интерфейс. При разработке использовались язык программирования C++, библиотека *libpng*, а также фреймворк Qt.

SUMMARY

The task of the course work is to develop a program for working with images, in particular with images in PNG format. As part of the course work, it was decided to develop a graphical interface for the program – the Qt framework was used for its implementation. To read and save a PNG image, the *libpng* library was selected.

The course work consists of an explanatory note and the source code of the developed program.

During the work, a program was developed for working with PNG images, which has a graphical interface. The development used the C++ programming language, *libpng* library, and the Qt framework.

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ

Цель работы.

Разработка программы для работы с изображениями формата PNG.

Задачи.

- Изучение языка программирования C++;
- Изучение фреймворка Qt;
- Изучение сигнатуры формата PNG;
- Изучение библиотеки libpng для работы с изображениями формата PNG;
- Написание исходного кода программы;
- Сборка программы;
- Тестирование программы.

1. ФОРМУЛИРОВКА ЗАДАНИЯ

Вариант 12.

Программа должна иметь CLI или GUI. Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Общие сведения:

- Формат картинки PNG (рекомендуем использовать библиотеку libpng)
- Файл всегда соответствует формату PNG
- Обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями
- Все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены)

Программа должна реализовывать следующий функционал по обработке PNG-файла:

1. Рисование квадрата. Квадрат определяется:
    - Координатами левого верхнего угла
    - Размером стороны
    - Толщиной линий
    - Цветом линий
    - Может быть залит или нет
    - Цветом, которым он залит, если пользователем выбран залитый
2. Поменять местами 4 куска области. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Способом обмена частей: “по кругу”, по диагонали

3. Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Функционал определяется Цветом, в который надо перекрасить самый часто встречаемый цвет.
4. Инверсия цвета в заданной области. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области

2. РАЗРАБОТКА ПРОГРАММЫ

Для удобства разработки GUI был использован фреймворк Qt, предоставляющий множество классов для разработки графического интерфейса программы. По мимо этого, для удобного считывания и записи изображений формата PNG была использована библиотека *libpng*. Разработка программы началась с реализации классов и структур для работы с изображениями формата PNG: *ImagePNG* и *ImagePart*.

2.1. Класс ImagePNG

Используется для считывания, хранения, обработки и сохранения изображений формата PNG. Следует отметить, что класс работает с изображениями формата PNG цветовой модели RGBA с глубиной цвета 8 бит. Также следует обратить внимание, что функции работы с изображением предполагают, что левый верхний пиксель изображения имеет координаты (0; 0), а не (1; 1).

Поля класса ImagePNG:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию
int width	*private*	Хранит ширину изображения в пикселях.	*0*
int height	*private*	Хранит высоту изображения в пикселях.	*0*
png_byte colorType	*private*	Хранит цветовую модель изображения.	*0*
png_byte bitDepth	*private*	Хранит глубину цвета изображения.	*0*
png_byte interlaceType	*private*	Хранит значение поля *interlace* заголовка изображения.	*0*

png_byte filteringMethod	*private*	Хранит значение поля *filtering* заголовка изображения.	*0*
png_byte compressionType	*private*	Хранит значение поля *compression* заголовка изображения.	*0*
png_bytepp pixelArray	*private*	Хранит адрес на двумерный массив байт, который используется для хранения пикселей изображения.	*nullptr*
QFileInfo fileInfo	*private*	Хранит информацию о изображении.	-
bool loaded	*private*	Содержит информацию о том, было ли считано изображение.	*false*
bool changed	*private*	Содержит информацию о том, было ли изменено изображение.	*false*
struct ImagePart	*private*	Предназначена для хранения части изображения (более подробно в пункте 2.2.).	-

Методы класса ImagePNG:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*int*	*load(const QString& imagePath, QWidget* parent = nullptr)*
public	*int*	*save(const QString& imagePath)*
private	*ImagePart**	*copyPart(const QPoint& point1, const QPoint& point2) const*
private	*void*	*pastePart(const ImagePart* part, const QPoint& point)*
public	*int*	*getWidth() const*
public	*int*	*getHeight() const*
public	*png_byte*	*getBitDepth() const*

public	*png_byte*	*getColorType() const*
public	*QString*	*getColorModelName() const*
public	*QFileInfo*	*getFileInfo() const*
public	*QPixmap*	*getPixmap() const*
public	*bool*	*isLoaded() const*
public	*bool*	*isChanged() const*
public	*bool*	*isCoordinatesCorrect(const QPoint& point) const*
public	*void*	*updateFileInfo(const QString& path = "")*
public	*int*	*getCorrectY(int y) const*
public	*int*	*getCorrectX(int x) const*
public	*QPoint&*	*correctPoint(QPoint& point) const*
public	*QPoint*	*correctPoint(const QPoint& point) const*
public	*void*	*paintSquare(const QPoint& point, int lineWidth, int squareSide, const QColor& color1, FillType fillType, const QColor& color2)*
public	*void*	*changeImageAreas(const QPoint& point1, const QPoint& point2, ExchangingMethod exchangingMethod)*
public	*void*	*replaceCommonColor(const QColor& newColor)*
public	*void*	*invertColors(const QPoint& point1, const QPoint& point2)*
public	-	*~ImagePNG()*

1. *int load(const QString& imagePath, QWidget* parent = nullptr)*

Считывает изображение из файла. Для считывания PNG изображения из файла используются функции библиотеки *libpng*. Следует отметить, что в случае попытки считывания изображения формата PNG, отличной от цветовой модели RGBA или глубины цвета 8 бит, то программа попытается преобразовать его к нужному формату при согласии пользователя. Принимает в качестве аргументов: imagePath – путь к файлу и *parent* – родительское окно для диалогового окна, которое будет отображаться в случае преобразования изображения для получения согласия от пользователя. Возвращает целое число, содержащее

информацию о результате работы функции. Может вернуть следующие значения:

- 0 – считывание выполнено успешно;
- 1 – ошибка открытия файла для считывания;
- 2 – файл не является изображением формата PNG;
- 3 – ошибка при создании *png_structp*;
- 4 – ошибка при создании *png_infop*;
- 5 – ошибка инициализации IO или считывания заголовка информации изображения;
- 6 – ошибка преобразования изображения к формату RGBA 8-bit;
- 7 – ошибка считывания таблицы пикселей изображения;
- 8 – пользователь отказался от преобразование исходного изображения к формату RGBA 8-bit.
- 9 – неизвестный формат фильтрации и сжатия изображения PNG.

Листинг 1.

```
// qDebug() – используется для отображения отладочной информации

int ImagePNG::load(const QString& imagePath, QWidget* parent) {
    // Открываем файл на чтение.
    // imagePath.toStdString().c_str() – цепочка методов преобразует строку класса QString к стандартной строке стиля Си.
    FILE* file = fopen(imagePath.toStdString().c_str(), "rb");

    // Проверяем, удалось ли получить доступ к файлу. Если нет, то выходим из функции и возвращает код ошибки 1.
    if (!file) {
        qDebug("Error in ImagePNG::load with fopen()");
        return 1;
    }

    // Считываем первые 8 байт файла и проверяем по ним, является ли файл изображением формата PNG. Если заголовок не совпал с заголовком PNG файла, то выходим из функции, отчищаем выделенную память и возвращаем код ошибки 2.
    unsigned char header[8];
    fread(header, sizeof(unsigned char), 8, file);

    if (png_sig_cmp(header, 0, 8)){
        qDebug("Error in ImagePNG::load with png_sig_cmp() (Non-PNG header)");
        fclose(file);
        return 2;
    }

    // Создаем структуру для чтения изображения. Если не удается создать – выходим из функции, отчищаем выделенную память и возвращаем код ошибки 3.
```

```
    png_structp pngStructPtr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);

    if (!pngStructPtr) {
        qDebug("Error in ImagePNG::load with png_create_read_struct()");
        fclose(file);
        return 3;
    }

    // Создаем структуру для информации изображения. Если не удается создать -
выходим из функции, отчищаем выделенную память и возвращаем код ошибки 4.
    png_infop pngInfoStructPtr = png_create_info_struct(pngStructPtr);

    if (!pngInfoStructPtr) {
        qDebug("Error in ImagePNG::load with png_create_info_struct()");
        png_destroy_read_struct(&pngStructPtr, NULL, NULL);
        fclose(file);
        return 4;
    }

    // Создаем точку обработки ошибок в последующих функциях. Если в них
возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то
выходим из функции, очищаем выделенную память и возвращаем код ошибки 5.
    if (setjmp(png_jmpbuf(pngStructPtr))) {
        qDebug("Error in ImagePNG::load with setjmp(png_jmpbuf())");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 5;
    }

    // Инициализируем ввод и вывод, сообщаем библиотеке, что мы уже считали с
начала файла 8 байт, после считываем информационные чанки изображения.
    png_init_io(pngStructPtr, file);
    png_set_sig_bytes(pngStructPtr, 8);
    png_read_info(pngStructPtr, pngInfoStructPtr);

    // Инициализируем поля объекта значениями.
    width = png_get_image_width(pngStructPtr, pngInfoStructPtr);
    height = png_get_image_height(pngStructPtr, pngInfoStructPtr);
    colorType = png_get_color_type(pngStructPtr, pngInfoStructPtr);
    bitDepth = png_get_bit_depth(pngStructPtr, pngInfoStructPtr);
    filteringMethod = png_get_filter_type(pngStructPtr, pngInfoStructPtr);
    compressionType = png_get_compression_type(pngStructPtr, pngInfoStructPtr);
    interlaceType = png_get_interlace_type(pngStructPtr, pngInfoStructPtr);

    png_set_interlace_handling(pngStructPtr);

    // Проверяем, чтобы значения полей были равны 0
    if (compressionType != 0 || filteringMethod != 0) {
        qDebug("This image format is not supported");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 9;
    }

    qDebug("Color type: %d, bit depth: %d, interlace type: %d", colorType,
bitDepth, interlaceType);

    // Если формат изображения не соответствует RGBA 8-bit, то выводим диалог
пользователю с предложением преобразовать изображение к данному формату. Если
пользователь отказывается, то выходим из функции, отчищаем выделенную память и
возвращаем код ошибки 8.
    if (bitDepth != 8 || colorType != PNG_COLOR_TYPE_RGBA) {
```

```
        QMessageBox::StandardButton button = QMessageBox::question(parent,
"Converting an image", "The selected PNG image doesn't match the RGBA 8-bit
format. Do you want to convert this image to this format?");

        if (button == QMessageBox::StandardButton::No) {
            qDebug("This image format is not supported");
            png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
            fclose(file);
            return 8;
        }
    }

    // Далее выполняем преобразования формата.
    if (bitDepth == 16) {
        qDebug("Converting bit depth: 16 to 8");
        png_set_strip_16(pngStructPtr);
        bitDepth = 8;
    }

    if (bitDepth < 8) {
        qDebug("Converting bit depth: 1, 2 or 4 to 8");

        if (colorType == PNG_COLOR_TYPE_GRAY) {
            png_set_expand_gray_1_2_4_to_8(pngStructPtr);
        } else {
            png_set_packing(pngStructPtr);
        }

        bitDepth = 8;
    }

    if (colorType == PNG_COLOR_TYPE_PALETTE) {
        qDebug("Converting color type: COLOR_TYPE_PALETTE to COLOR_TYPE_RGB");
        png_set_palette_to_rgb(pngStructPtr);
        colorType = PNG_COLOR_TYPE_RGB;
    }

    if     (colorType     ==     PNG_COLOR_TYPE_GRAY     ||     colorType     ==
PNG_COLOR_TYPE_GRAY_ALPHA) {
        qDebug("Converting color type: PNG_COLOR_TYPE_GRAY to COLOR_TYPE_RGB or
PNG_COLOR_TYPE_GRAY_ALPHA to COLOR_TYPE_RGBA");
        png_set_gray_to_rgb(pngStructPtr);

        if (colorType == PNG_COLOR_TYPE_GRAY) {
            colorType = PNG_COLOR_TYPE_RGB;
        } else {
            colorType = PNG_COLOR_TYPE_RGBA;
        }
    }

    if (colorType == PNG_COLOR_TYPE_RGB) {
        qDebug("Converting color type: COLOR_TYPE_RGB to COLOR_TYPE_RGBA, fill
alpha channel");
        png_set_add_alpha(pngStructPtr, 0xFF, PNG_FILLER_AFTER);
        colorType = PNG_COLOR_TYPE_RGBA;
    }

    if (png_get_valid(pngStructPtr, pngInfoStructPtr, PNG_INFO_tRNS)) {
        qDebug("Converting color type to alpha");
        png_set_tRNS_to_alpha(pngStructPtr);
        colorType = PNG_COLOR_TYPE_RGBA;
    }
```

```
    // Обновляем информацию о изображении.
    png_read_update_info(pngStructPtr, pngInfoStructPtr);

    colorType = png_get_color_type(pngStructPtr, pngInfoStructPtr);
    bitDepth = png_get_bit_depth(pngStructPtr, pngInfoStructPtr);

    qDebug("Converted color type: %d, bit depth: %d", colorType, bitDepth);

    // Если преобразование к формату RGBA 8-bit не удалось, то выходим из
функции, отчищаем выделенную память и возвращаем код ошибки 8.
    if (colorType != PNG_COLOR_TYPE_RGBA || bitDepth != 8) {
        qDebug("Error with converting image to RGBA 8 bit");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 6;
    }


    // Создаем точку обработки ошибок в последующих функциях. Если в них
возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то
выходим из функции, очищаем выделенную память и возвращаем код ошибки 7.
    if (setjmp(png_jmpbuf(pngStructPtr))) {
        qDebug("Error in ImagePNG::load with setjmp(png_jmpbuf()) #2");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 7;
    }

    // Выделяем память под массив пикселей (1 пиксель – 4 байта).
    pixelArray = new png_bytep[height];

    for (int y = 0; y < height; y++) {
        pixelArray[y]     =     new     png_byte[png_get_rowbytes(pngStructPtr,
pngInfoStructPtr)];
    }

    // Присваиваем значения пикселей изображения в выделенный массив.
    png_read_image(pngStructPtr, pixelArray);
    qDebug("The       image       is       successfully       loaded:       %s",
imagePath.toStdString().c_str());

    // Очищаем выделенную память.
    png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
    fclose(file);

    // Ставим флаг, что изображение загружено, а также получаем информацию о
файле изображения, присваиваем все полям объекта.
    loaded = true;
    fileInfo.setFile(imagePath);

    // Функция завершилась успешно, возвращаем код ошибки 0.
    return 0;
}
```

2. int save(const QString& imagePath)

Сохраняет изображение в файл. Следует отметить, что форматом сохраненного изображения всегда является PNG цветовой модели RGBA и глубины цвета 8-bit, даже если исходный считанный файл был другого формата.

Принимает в качестве аргумента: *imagePath* – путь сохранения файла с изображением. Возвращает целое число, содержащее информацию о результате работы функции. Может вернуть следующие значения:

- 0 – сохранение выполнено успешно;
- 1 – ошибка создания или перезаписи файла для сохранения;
- 2 – ошибка при создании *png_structp*;
- 3 – ошибка при создании *png_infop*;
- 4 – ошибка при инициализации IO;
- 5 – ошибка записи IHDR заголовка в файл;
- 6 – ошибка записи изображения в файл;
- 7 – ошибка записи конца изображения в файл;

Листинг 2.

```
int ImagePNG::save(const QString& imagePath) {
    // Открываем файл на запись. Если файл не существует, то он будет создан. Если файл уже существует, то будет перезаписан.
    FILE* file = fopen(imagePath.toStdString().c_str(), "wb");

    // Проверяем, удалось ли получить доступ к файлу. Если нет, то выходим из функции и возвращает код ошибки 1.
    if (!file) {
        qDebug("Error in ImagePNG::save with fopen()");
        return 1;
    }

    // Создаем структуру для записи изображения. Если не удается создать - выходим из функции, отчищаем выделенную память и возвращаем код ошибки 3.
    png_structp pngStructPtr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (!pngStructPtr) {
        qDebug("Error in ImagePNG::save with png_create_write_struct()");
        fclose(file);
        return 2;
    }

    // Создаем структуру для информации изображения. Если не удается создать - выходим из функции, отчищаем выделенную память и возвращаем код ошибки 3.
    png_infop pngInfoStructPtr = png_create_info_struct(pngStructPtr);
    if (!pngInfoStructPtr) {
        qDebug("Error in ImagePNG::save with png_create_info_struct()");
        png_destroy_write_struct(&pngStructPtr, NULL);
        fclose(file);
        return 3;
    }

    // Создаем точку обработки ошибок в последующих функциях. Если в них возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то выходим из функции, очищаем выделенную память и возвращаем код ошибки 4.
```

```
    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf())");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 4;
    }

    // Инициализируем ввод и вывод.
    png_init_io(pngStructPtr, file);

    // Создаем точку обработки ошибок в последующих функциях. Если в них
возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то
выходим из функции, очищаем выделенную память и возвращаем код ошибки 5.
    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #2");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 5;
    }

    // Устанавливаем IHDR чанк изображения и записываем в файл.
    png_set_IHDR(pngStructPtr, pngInfoStructPtr, width, height, bitDepth,
colorType, PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_BASE,
PNG_FILTER_TYPE_BASE);
    png_write_info(pngStructPtr, pngInfoStructPtr);

    // Создаем точку обработки ошибок в последующих функциях. Если в них
возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то
выходим из функции, очищаем выделенную память и возвращаем код ошибки 6.
    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #3");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 6;
    }

    // Записываем изображение в файл.
    png_write_image(pngStructPtr, pixelArray);

    // Создаем точку обработки ошибок в последующих функциях. Если в них
возникнет ошибка, то мы зайдем в данный блок кода. Если это произойдет, то
выходим из функции, очищаем выделенную память и возвращаем код ошибки 7.
    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #4");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 7;
    }

    // Заканчиваем запись изображения в файл.
    png_write_end(pngStructPtr, NULL);

    qDebug("The image is successfully saved: %s",
imagePath.toStdString().c_str());

    // Очищаем выделенную память.
    png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
    fclose(file);

    // Устанавливаем флаг изменения изображения как ложь, так как все
изменения были сохранены.
    changed = false;
```

```
    // Функция завершилась успешно, возвращаем код ошибки 0.
    return 0;
}
```

3. ImagePart copyPart(const QPoint& point1, const QPoint& point2) const*

Копирует часть загруженного изображения и сохраняет ее в структуру *ImagePart*. Принимает в качестве аргументов: *point1* – левый верхний угол копируемой области и *point2* – правый нижний угол копируемой области. Возвращает указатель на структуру *ImagePart*, которая содержит скопированную часть изображения.

Листинг 3.

```
ImagePNG::ImagePart* ImagePNG::copyPart(const QPoint& point1, const QPoint&
point2) const {
    // Проверяем, что изображение загружено и координаты точек корректны.
    if (isLoaded() && isCoordinatesCorrect(point1) &&
isCoordinatesCorrect(point2)) {
        // Создаем объект типа ImagePart нужного размера области.
        ImagePart* part = new ImagePart(point2.x() - point1.x() + 1,
point2.y() - point1.y() + 1);

        // Копируем значения пикселей из массива пикселей изображения в массив
пикселей части.
        for (int y = 0; y < part->height; y++) {
            for (int x = 0; x < part->width; x++) {
                setPixel(part->pixelArray[y] + 4 * x,
getPixel(pixelArray[point1.y() + y] + 4 * (point1.x() + x)));
            }
        }

        qDebug("Part of the image in area %s to %s was successfully copied",
pointToString(point1).toStdString().c_str(),
pointToString(point2).toStdString().c_str());

        // Возвращаем скопированную часть.
        return part;
    }

    // В случае ошибки возвращаем nullptr.
    return nullptr;
}
```

4. void pastePart(const ImagePart part, const QPoint& point)*

Вставляет скопированную часть изображения в определенную точку загруженного изображения. Принимает в качестве аргументов: part – указатель на скопированную часть изображения и point – место вставки скопированной

части в изображения (место, где будет расположен левый верхний угол скопированной части). Ничего не возвращает.

Листинг 4.

```
void ImagePNG::pastePart(const ImagePNG::ImagePart* part, const QPoint& point)
{
    // Проверяем, что изображение загружено, переданная часть изображения
существует и координаты точки корректны.
    if (isLoaded() && part != nullptr && isCoordinatesCorrect(point)) {
        // Рассчитываем корректные координаты краев области в случае, если
вставляемая область заходит за пределы изображения.
        int endY = getCorrectY(point.y() + part->height);
        int endX = getCorrectX(point.x() + part->width);

        // Производим копирование пикселей из массива пикселей части в массив
пикселей изображения.
        for (int y = point.y(); y < endY; y++) {
            for (int x = point.x(); x < endX; x++) {
                setPixel(pixelArray[y] + 4 * x, getPixel(part->pixelArray[y -
point.y()] + 4 * (x - point.x())));
            }
        }

        qDebug("Part of the image was successfully pasted in %s",
pointToString(point).toStdString().c_str());
    }
}
```

5. *int getWidth() const*

   Возвращает ширину изображения в пикселях.

6. *int getHeight() const*

   Возвращает высоту изображения в пикселях.

7. *png_byte getBitDepth() const*

   Возвращает глубину цвета изображения в битах (целочисленное значение).

8. *png_byte getColorType() const*

   Возвращает тип цветовой модели изображения (целочисленное значение).

9. *QString getColorModelName() const*

   Возвращает название цветовой модели изображения в виде строки. Может вернуть следующие значения в зависимости от значения поля *colorType*:

- *"RGB"*
- *"Gray"*
- *"RGBA"*
- *"Palette"*
- *"Gray alpha"*
- *"Unknown"*

Листинг 5.

```
QString ImagePNG::getColorModelName() const {
    // В зависимости от значения поля colorType возвращаем соответствующее
название цветовой модели.
    switch (colorType) {
    case PNG_COLOR_TYPE_RGB:
        return "RGB";
    case PNG_COLOR_TYPE_GRAY:
        return "Gray";
    case PNG_COLOR_TYPE_RGBA:
        return "RGBA";
    case PNG_COLOR_TYPE_PALETTE:
        return "Palette";
    case PNG_COLOR_TYPE_GRAY_ALPHA:
        return "Gray alpha";
    default:
        return "Unknown";
    }
}
```

10. QFileInfo getFileInfo() const

Возвращает информацию о файле с изображением.

11. QPixmap getPixmap() const

Преобразует данное изображение к объекту класса *QPixmap*. Возвращает изображение в виде объекта класса *QPixmap*.

Листинг 6.

```
QPixmap ImagePNG::getPixmap() const {
    // Создаем объект класса QPixmap.
    QPixmap pixmap;

    // Проверяем, что изображение загружено.
    if (isLoaded())  {
        // Создаем объект класса QImage и копирует туда наше изображение.
        QImage image(width, height, QImage::Format_RGBA64);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                png_bytep pixel = pixelArray[y] + 4 * x;
```

```
                image.setPixelColor(x, y, QColor(pixel[0], pixel[1], pixel[2],
pixel[3]));
            }
        }

        // Преобразуем изображение в QPixmap.
        pixmap = QPixmap::fromImage(image);
    }

    // Возвращаем Pixmap.
    return pixmap;
}
```

12. bool isLoaded() const

Возвращает информацию о том, было ли считано изображение.

13. bool isChanged() const

Возвращает информацию о наличии несохраненных изменений.

14. bool isCoordinatesCorrect(const QPoint& point) const

Возвращает информацию о корректности координат пикселя изображения.

15. void updateFileInfo(const QString& path = "")

Обновляет информацию о файле изображения, находящегося по пути *path*. В качестве аргумента принимает *path* – новый путь к файлу. Если *path* – пустая строка, то путь будет взят из объекта *fileInfo*. Ничего не возвращает.

16. int getCorrectY(int y) const

Получает исправленную координату Y. В качестве аргумента принимает *y* – координата Y. Если координата является отрицательной (находится выше изображения) – возвращается 0, если координата больше или равна высоте изображения (находится ниже изображения) – возвращается уменьшенное на 1 значение высоты изображения, иначе если координата не выходит за пределы изображения – возвращает переданную координату без изменений.

17. int getCorrectX(int x) const

Получает исправленную координату X. В качестве аргумента принимает *x* – координата X. Если координата является отрицательной (находится левее изображения) – возвращается 0, если координата больше или равна ширине изображения (находится правее изображения) – возвращается уменьшенное на 1 значение ширины изображения, иначе если координата не выходит за пределы изображения – возвращает переданную координату без изменений.

18. QPoint& correctPoint(QPoint& point) const

Исправляет координаты пикселя изображения. В качестве аргумента принимает point – ссылка на объект класса *QPoint,* содержащий координаты. В случае, если координата X или Y некорректна, то координаты исправляются на корректные. Возвращает переданную ссылку на объект класса *QPoint*.

19. QPoint correctPoint(const QPoint& point) const

Исправляет координаты пикселя изображения. В качестве аргумента принимает point – константная ссылка на объект класса *QPoint,* содержащий координаты. В случае, если координата X или Y некорректна, то координаты исправляются на корректные. Возвращает новый объект класса *QPoint* с исправленными координатами, при этом координаты переданного объекта не изменяются.

20. void paintSquare(const QPoint& point, int lineWidth, int squareSide, const QColor& color1, FillType fillType, const QColor& color2)

Рисует на изображении квадрат с заданными параметрами. Принимает в качестве аргументов: *point* – координаты пикселя, где нужно разместить левый верхний угол квадрата, *lineWidth* – ширина линии границ квадрата в пикселях, *squareSide* – размер стороны квадрата в пикселях, *color1* – цвет линии границ квадрата, *fillType* – тип заливки квадрата и *color2* – цвет заливки квадрата. Существуют следующие типы заливки: 1 – без заливки и 2 – с заливкой

(определение перечисления *FillType* можно найти в файле *enumerations.h*). Ничего не возвращает.

Листинг 7.

```
void ImagePNG::paintSquare(const QPoint& point, int lineWidth, int squareSide,
const QColor& color1, FillType fillType, const QColor& color2) {
    // Проверяем, загружено ли изображение. Выходим, если нет.
    if (!isLoaded()) {
        return;
    }

    qDebug("Starting drawing a square with %d line width and %d square side:",
lineWidth, squareSide);
    qDebug("Starting point: %s;", pointToString(point).toStdString().c_str());
    qDebug("Filling type: %d;", fillType);
    qDebug("Colors: %s and %s.", colorToString(color1).toStdString().c_str(),
colorToString(color2).toStdString().c_str());

    // Получаем нужные для рисования координаты различных областей.
    int xRight = getCorrectX(point.x() + squareSide - 1);
    int xInsideRight = point.x() + squareSide - lineWidth - 1;
    int xInsideLeft = point.x() + lineWidth;
    int yDown = getCorrectY(point.y() + squareSide - 1);
    int yIndiseUp = point.y() + lineWidth;
    int yIndiseDown = point.y() + squareSide - lineWidth - 1;


    // Закрашиваем нужные пиксели в области рисования квадрата.
    for (int y = point.y(); y <= yDown; y++) {
        for (int x = point.x(); x <= xRight; x++) {
            if (isCoordinatesCorrect(QPoint(x, y))) {
                if (x < xInsideLeft || x > xInsideRight || y < yIndiseUp || y >
yIndiseDown) {
                    setPixel(pixelArray[y] + 4 * x, color1);
                } else if (fillType == FillType::FILLED) {
                    setPixel(pixelArray[y] + 4 * x, color2);
                }
            }
        }
    }

    qDebug("Drawing a square finished");

    // Сообщаем, что изображение было изменено.
    changed = true;
}
```

21. void changeImageAreas(const QPoint& point1, const QPoint& point2, ExchangingMethod exchangingMethod)

Делит выбранную область изображения на четыре равные части и меняет эти части местами. Принимает в качестве аргументов: *point1* – координаты левого верхнего угла области, *point2* – координаты правого нижнего угла области и *exchangingMethod* – метод обмена частей местами. Существуют следующие

методы обмена частей: 1 – поменять части по часовой стрелке, 2 – против часовой стрелки, 3 – поменять диагональные элементы местами (определение перечисления *ExchangingMethod* можно найти в файле *enumerations.h*). Ничего не возвращает.

Листинг 8.

```
void ImagePNG::changeImageAreas(const QPoint& point1, const QPoint& point2,
ExchangingMethod exchangingMethod) {
    // Проверяем, загружено ли изображение. Выходим, если нет.
    if (!isLoaded()) {
        return;
    }

    // Исправляем точки выбранной области.
    QPoint fixedPoint1 = correctPoint(point1);
    QPoint fixedPoint2 = correctPoint(point2);

    // Делаем поправки в размерах области, чтобы ее можно было поделить на 4
равные части.
    if ((fixedPoint2.x() - fixedPoint1.x()) % 2 == 0) {
        fixedPoint2.setX(point2.x() - 1);
    }

    if ((fixedPoint2.y() - fixedPoint1.y()) % 2 == 0) {
        fixedPoint2.setY(point2.y() - 1);
    }

    // Получаем некоторые параметры для дальнейшего использования.
    int deltaX = fixedPoint2.x() - fixedPoint1.x();
    int deltaY = fixedPoint2.y() - fixedPoint1.y();
    int partWidth = deltaX / 2 + 1;
    int partHeight = deltaY / 2 + 1;
    int middleX = fixedPoint1.x() + partWidth;
    int middleY = fixedPoint1.y() + partHeight;

    // Если выбранная область слишком мала, то выходим из функции.
    if (partWidth < 2 || partHeight < 2) {
        qDebug("Area is too small!");
        return;
    }

    QPoint partPoint1 = fixedPoint1;
    QPoint partPoint2 = QPoint(middleX, fixedPoint1.y());
    QPoint partPoint3 = QPoint(fixedPoint1.x(), middleY);
    QPoint partPoint4 = QPoint(middleX, middleY);

    // Копируем четыре части.
    ImagePart* part1 = copyPart(partPoint1, QPoint(middleX - 1, middleY - 1));
    ImagePart* part2 = copyPart(partPoint2, QPoint(fixedPoint2.x(), middleY -
1));
    ImagePart* part3 = copyPart(partPoint3, QPoint(middleX - 1,
fixedPoint2.y()));
    ImagePart* part4 = copyPart(partPoint4, fixedPoint2);

    // В зависимости от методы перестановки переставляем части местами.
    if (exchangingMethod == ExchangingMethod::CLOCKWISE_METHOD) {
        pastePart(part1, partPoint2);
        pastePart(part2, partPoint4);
        pastePart(part3, partPoint1);
```

```
        pastePart(part4, partPoint3);
    } else if (exchangingMethod == ExchangingMethod::ANTICLOCKWISE_METHOD) {
        pastePart(part1, partPoint3);
        pastePart(part2, partPoint1);
        pastePart(part3, partPoint4);
        pastePart(part4, partPoint2);
    } else {
        pastePart(part1, partPoint4);
        pastePart(part2, partPoint3);
        pastePart(part3, partPoint2);
        pastePart(part4, partPoint1);
    }

    // Очищаем части.
    delete part1;
    delete part2;
    delete part3;
    delete part4;

    // Сообщаем, что изображение было изменено.
    changed = true;
}
```

22. void replaceCommonColor(const QColor& newColor)

Заменяет самый часто встречающийся цвет пикселей в изображении на другой цвет. Принимает в качестве аргумента *newColor* – цвет замены.

Листинг 9.

```
void ImagePNG::replaceCommonColor(const QColor& newColor) {
    // Проверяем, загружено ли изображение. Выходим, если нет.
    if (!isLoaded()) {
        return;
    }

    // Создаем словарь, где будем хранить цвет и количество пикселей в
изображении такого цвета.
    QMap<QColor, int> dictionary;

    // Считаем цвета.
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            QColor color = getColorFromPixel(pixelArray[y] + 4 * x);

            if (dictionary.contains(color)) {
                dictionary[color] += 1;
            } else {
                dictionary[color] = 1;
            }
        }
    }

    // Пройдемся по словарю и найдем самый часто встречающийся цвет.
    QColor commonColor = dictionary.begin().key();
    int max = dictionary.begin().value();

    for (auto pair : dictionary.toStdMap()) {
        if (pair.second > max) {
            max = pair.second;
```

```
            commonColor = pair.first;
        }
    }

    // Пробегаемся по изображению и заменяем найденный цвет на другой.
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            png_bytep pixel = pixelArray[y] + 4 * x;
            QColor color = getColorFromPixel(pixel);

            if (color == commonColor) {
                setPixel(pixel, newColor);
            }
        }
    }

    // Сообщаем, что изображение было изменено.
    changed = true;
}
```

23. void invertColors(const QPoint& point1, const QPoint& point2)

Инвертирует цвет в выбранной области изображения. Принимает в качестве аргументов: *point1* – координаты левого верхнего угла области и *point2* – координаты правого нижнего угла области.

Листинг 10.

```
void ImagePNG::invertColors(const QPoint& point1, const QPoint& point2) {
    // Проверяем, загружено ли изображение. Выходим, если нет.
    if (!isLoaded()) {
        return;
    }

    // Исправляем точки выбранной области.
    QPoint fixedPoint1 = correctPoint(point1);
    QPoint fixedPoint2 = correctPoint(point2);

    // Пробегаемся по выбранной области и инвертируем цвет.
    for (int y = fixedPoint1.y(); y < fixedPoint2.y(); y++) {
        for (int x = fixedPoint1.x(); x < fixedPoint2.x(); x++) {
            invertPixelColor(pixelArray[y] + 4 * x);
        }
    }

    // Сообщаем, что изображение было изменено.
    changed = true;
}
```

24. ~ImagePNG()

Деструктор, вызывается при уничтожении объекта класса *ImagePNG*. Очищает выделенную под хранение данных класса *ImagePNG* динамическую память.

2.2. Структура ImagePart

Используется для хранения части изображения.

Поля структуры ImagePart:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию
int width	*public*	Хранит ширину изображения в пикселях.	*0*
int height	*public*	Хранит высоту изображения в пикселях.	*0*
png_bytepp pixelArray	*public*	Хранит адрес на двумерный массив байт, который используется для хранения пикселей изображения.	*nullptr*

Методы структуры ImagePart:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*ImagePart(int width, int height)*
public	-	*~ImagePart()*

1. *ImagePart(int width, int height)*

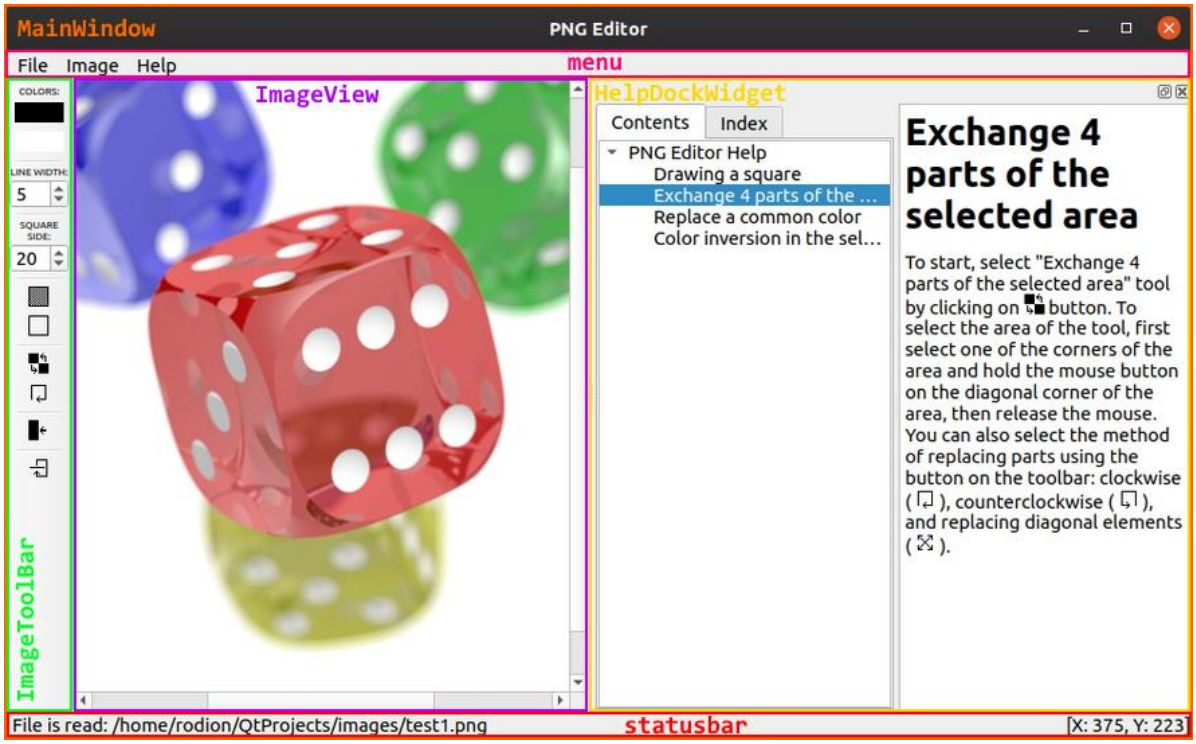
Конструктор, вызывается при создании объекта структуры *ImagePart*. Выделяет память для хранения части изображения. Принимает в качестве аргументов: *width* – ширина части изображения в пикселях и *height* – высота части изображения в пикселях.

2. *~ImagePart()*

Деструктор, вызывается при уничтожении объекта структуры *ImagePart*. Очищает выделенную под хранение данных структуры *ImagePart* динамическую память.

2.3. Разработка графического интерфейса

После того, как были написаны основные классы и структуры для работы с изображениями, началась разработка классов графического интерфейса программы. Фреймворк Qt предоставляет базовые классы графического интерфейса со стандартным функционалом, поэтому все разработанные классы наследуются от классов Qt и дополняются требуемым для окон функционалом. Структура графического интерфейса программы построена следующим образом:



Из всего интерфейса часть графических элементов были написаны самостоятельно (*MainWindow*, *ImageToolBar*, *ImageView*, *ImageScene*, *HelpDockWidget*, *HelpBrowser*), а часть – сгенерированы при помощи редактора форм *QtDesigner* (*statusbar*, *menu* и так далее), все эти элементы помещены в

класс *Ui::MainWindow* как публичные поля. Стоит отметить, что в Qt существует система автоматического освобождения памяти связанных виджетов. К примеру, если удалить объект класса ImageToolBar, то все входящие в него виджеты (связанные с ним при помощи передачи параметра *parent*) также будут удалены. Поэтому в большинстве классов не требуется освобождать память, выделенную под виджеты.

2.4. Класс ImageScene

Представляет собой сцену, на которую наносится изображения для отображения. После чего данная сцена может быть отображена в окне объекта класса *ImageView*. Наследуется от класса *QGraphicsScene*.

Методы класса ImageScene:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*ImageScene(qreal x, qreal y, qreal width, qreal height, QObject* parent = nullptr)*
public	*void*	*mouseMoveEvent(QGraphicsSceneMouseEvent * event) override*
public	*void*	*mousePressEvent(QGraphicsSceneMouseEvent * event) override*
public	*void*	*mouseReleaseEvent(QGraphicsSceneMouseEvent* event) override*

Сигналы класса ImageScene:

Возвращаемое значение	Название метода и принимаемые аргументы
void	*point1Selected(QGraphicsSceneMouseEvent* event)*
void	*point2Selected(QGraphicsSceneMouseEvent* event)*
void	*mousePositionChanged(QGraphicsSceneMouseEvent* event)*

1. *ImageScene(qreal x, qreal y, qreal width, qreal height, QObject* parent = nullptr)*

Конструктор, вызывается при создании объекта класса *ImageScene*. Вызывает конструктор базового класса и передает в него параметры *x, y, width, height* и *parent*.

2. *void mouseMoveEvent(QGraphicsSceneMouseEvent* event) override*

Переопределённый метод класса QGraphicsScene, который вызывается каждый раз при перемещении указателя мыши в сцене. Активирует сигнал *mousePositionChanged(QGraphicsSceneMouseEvent* event)* и передает туда указатель *event* на объект события мыши *QGraphicsSceneMouseEvent*.

3. *void mousePressEvent(QGraphicsSceneMouseEvent* event) override*

Переопределённый метод класса QGraphicsScene, который вызывается каждый раз при нажатии пользователем на кнопку мыши в сцене. Активирует сигнал *point1Selected(QGraphicsSceneMouseEvent* event)* и передает туда указатель *event* на объект события мыши *QGraphicsSceneMouseEvent*.

4. *void mouseReleaseEvent(QGraphicsSceneMouseEvent* event) override*

Переопределённый метод класса QGraphicsScene, который вызывается каждый раз при отпускании пользователем кнопки мыши в сцене. Активирует сигнал *point2Selected(QGraphicsSceneMouseEvent* event)* и передает туда указатель *event* на объект события мыши *QGraphicsSceneMouseEvent*.

5. *void point1Selected(QGraphicsSceneMouseEvent* event)*

Сигнал, при активации которого происходит вызов слота on_imageScene_point1Selected класса *MainWindow*.

6. *void point2Selected(QGraphicsSceneMouseEvent* event)*

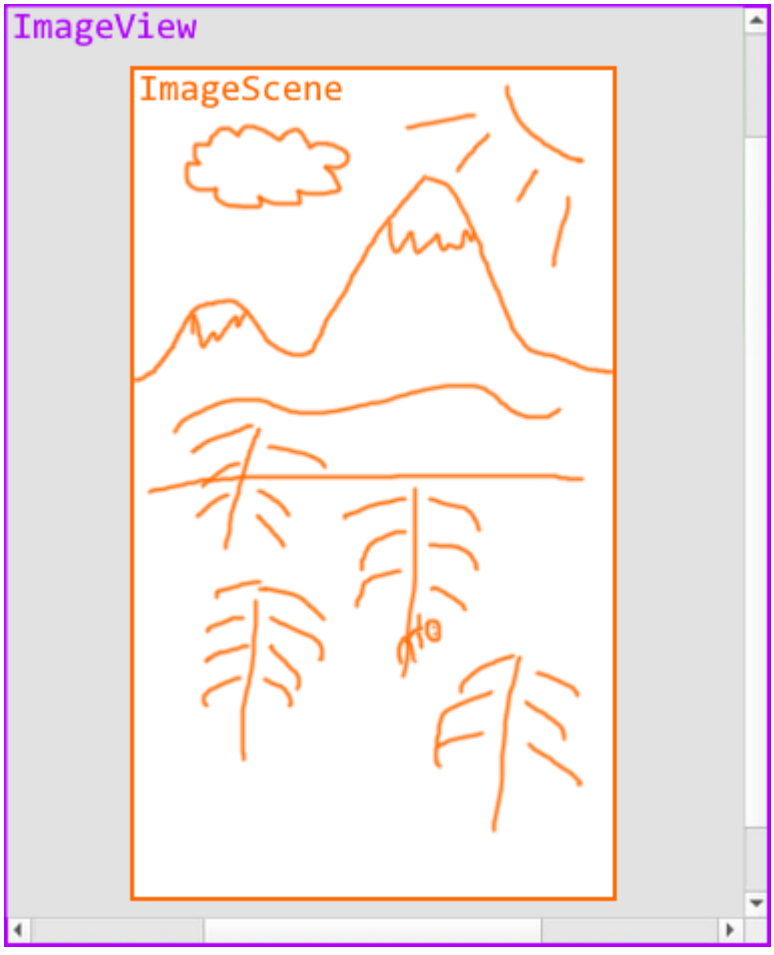
Сигнал, при активации которого происходит вызов слота on_imageScene_point2Selected класса *MainWindow*.

7. *void mousePositionChanged(QGraphicsSceneMouseEvent*)*

Сигнал, при активации которого происходит вызов слота on_imageScene_mousePositionChanged класса *MainWindow*.

2.5. Класс ImageView

Представляет собой окно, в котором будет отображаться сцена ImageScene. Класс позволяет устанавливать на сцену изображение, а также очищать её. Наследуется от класса *QGraphicsView*. Структура выглядит следующим образом:



Поля класса ImageView:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию

ImageScene scene*	*private*	Хранит адрес объекта сцены *ImageScene*.	*nullptr*

Методы класса ImageView:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*explicit ImageView(QWidget* parent = nullptr)*
public	*void*	*setImage(ImagePNG* image)*
public	*void*	*clearScene(int width = 0, int height = 0)*

1. *explicit ImageView(QWidget* parent = nullptr)*

Конструктор, вызывается при создании объекта класса *ImageView*. Вызывает конструктор базового класса и передает в него параметр *parent*. По мимо этого, происходит выделение памяти под данные объекта.

2. *void setImage(ImagePNG* image)*

Устанавливает изображение на сцену. В качестве аргумента принимает image – изображение. Ничего не возвращает.

3. *void clearScene(int width = 0, int height = 0)*

Очищает сцену. Принимает в качестве аргументов: *width* – ширина очищенной сцены и *height* – длина очищенной сцены. Ничего не возвращает.

2.6. Класс ImageToolBar

Представляет собой панель инструментов, на которой расположены различные кнопки для выбора инструментов и настройки их параметров. Класс позволяет активировать сигналы о нажатии кнопок и вызывать слоты в классе *MainWindow* для их обработки, а также хранить текущие параметры инструментов. Наследуется от класса *QToolBar*. Структура выглядит следующим образом:

Поля класса ImageToolBar:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию
QColor color1	*private*	Хранит первый цвет панели инструментов.	*QColor(Qt::black)*
QColor color2	*private*	Хранит второй цвет панели инструментов.	*QColor(Qt::white)*
ToolType selectedTool	*private*	Хранит выбранный инструмент на панели инструментов.	*ToolType::NONE_TOOL*
ExchangingMethod selectedExchangingMethod	*private*	Хранит выбранный метод обмена частей изображения для второго инструмента.	*ExchangingMethod::CLOCKWISE_METHOD*
FillType selectedFillType	*private*	Хранит выбранный тип заливки для первого инструмента.	*FillType::UNFILLED*
QToolButton changeColor1_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора первого цвета.	*nullptr*
QToolButton changeColor2_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора второго цвета.	*nullptr*
QToolButton changeTool1_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора первого инструмента.	*nullptr*
QToolButton changeTool2_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора второго инструмента.	*nullptr*
QToolButton changeTool3_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора третьего инструмента.	*nullptr*

QToolButton changeTool4_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора четвертого инструмента.	*nullptr*
QToolButton changeSquarePainting_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора типа заливки.	*nullptr*
QToolButton changeExchangeMethod_button*	*public*	Хранит адрес объекта кнопки панели инструментов для выбора метода замены частей.	*nullptr*
QSpinBox lineWidthSpinBox*	*public*	Хранит адрес объекта цифрового счетчика для настройки ширины линий.	*nullptr*
QSpinBox squareSizeSpinBox*	*public*	Хранит адрес объекта цифрового счетчика для настройки размера стороны квадрата.	*nullptr*

Методы класса ImageToolBar:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*ImageToolBar(QWidget* parent = nullptr)*
public	*QColor*	*getFirstColor() const*
public	*QColor*	*getSecondColor() const*
public	*ToolType*	*getSelectedTool() const*
public	*ExchangingMethod*	*getExchangingMethod() const*
public	*FillType*	*getFillType() const*
public	*void*	*setFirstColor(const QColor& color)*
public	*void*	*setSecondColor(const QColor& color)*
public	*void*	*setToolType(ToolType newToolType)*
public	*void*	*setExchangingMethod(ExchangingMethod newExchangingMethod)*
public	*void*	*setFillType(FillType newFillType)*
public	*void*	*resetTools()*

1. *ImageToolBar(QWidget* parent = nullptr)*

Конструктор, вызывается при создании объекта класса *ImageToolBar*. Вызывает конструктор базового класса и передает в него параметр *parent*. По мимо этого, происходит выделение памяти под данные объекта.

Листинг 11.

```
ImageToolBar::ImageToolBar(QWidget *parent): QToolBar(parent) {
    // Устанавливаем стили для виджета.
    setStyleSheet("QLabel {qproperty-alignment: AlignCenter;font-size: 8px;}
QToolButton {padding: 0px; margin: 0px 1px 0px 1px; width: 40px;} QSpinBox
{height: 17px; width: 20px;}");
    setContextMenuPolicy(Qt::PreventContextMenu);
    setObjectName("toolbar");
    setMovable(false);
    setIconSize(QSize(40, 17));

    // Создаем элементы панели инструментов (кнопки и т.д.).
    changeColor1_button = createToolButtonColor(this, "changeColor1", "Color
1", QColor(Qt::black), false);
    changeColor2_button = createToolButtonColor(this, "changeColor2", "Color
2", QColor(Qt::white), false);
    changeTool1_button = createToolButton(this, "changeTool1", "Drawing a
square", ":/icons/square_tool.png", true);
    changeTool2_button = createToolButton(this, "changeTool2", "Exchange 4
parts of the selected area", ":/icons/exchanging_tool.png", true);
    changeTool3_button = createToolButton(this, "changeTool3", "Replace a
common color", ":/icons/common_color_tool.png", false);
    changeTool4_button = createToolButton(this, "changeTool4", "Color
inversion in the selected area", ":/icons/invert_color.png", true);
    changeSquarePainting_button = createToolButton(this,
"changeSquarePainting", "Square filling", ":/icons/square_unpainted.png",
false);
    changeExchangeMethod_button = createToolButton(this,
"changeExchangeMethod", "Exchanging parts method", ":/icons/exchanging_1.png",
false);

    lineWidthSpinBox = createSpinBox(this, 5, 0, 1000000, "Line width");
    squareSizeSpinBox = createSpinBox(this, 20, 0, 1000000, "Square side");

    QLabel* label1 = new QLabel("COLORS:", this);
    QLabel* label2 = new QLabel("LINE WIDTH:", this);
    QLabel* label3 = new QLabel("SQUARE\nSIDE:", this);

    // Добавляем созданные виджеты на панель инструментов.
    addWidget(label1);
    addWidget(changeColor1_button);
    addWidget(changeColor2_button);
    addSeparator();
    addWidget(label2);
    addWidget(lineWidthSpinBox);
```

```
    addSeparator();
    addWidget(label3);
    addWidget(squareSizeSpinBox);
    addSeparator();
    addWidget(changeTool1_button);
    addWidget(changeSquarePainting_button);
    addSeparator();
    addWidget(changeTool2_button);
    addWidget(changeExchangeMethod_button);
    addSeparator();
    addWidget(changeTool3_button);
    addSeparator();
    addWidget(changeTool4_button);
}
```

2. *QColor getFirstColor() const*

Возвращает выбранный первый цвет на панели инструментов.

3. *QColor getSecondColor() const*

Возвращает выбранный второй цвет на панели инструментов.

4. *ToolType getSelectedTool() const*

Возвращает выбранный инструмент на панели инструментов.

5. *ExchangingMethod getExchangingMethod() const*

Возвращает выбранный метод обмена частей на панели инструментов.

6. *FillType getFillType() const*

Возвращает выбранный тип заливки на панели инструментов.

7. *void setFirstColor(const QColor& color)*

Устанавливает выбранный первый цвет на панели инструментов. В качестве аргумента принимает *color* – цвет. Ничего не возвращает.

8. *void setSecondColor(const QColor& color)*

Устанавливает выбранный второй цвет на панели инструментов. В качестве аргумента принимает *color* – цвет. Ничего не возвращает.

9. void setToolType(ToolType newToolType)

Устанавливает выбранный инструмент на панели инструментов. В качестве аргумента принимает *newToolType* – инструмент. Ничего не возвращает.

10. void setExchangingMethod(ExchangingMethod newExchangingMethod)

Устанавливает выбранный метод обмена частей на панели инструментов. В качестве аргумента принимает *newExchangingMethod* – метод обмена частей. Ничего не возвращает.

11. void setFillType(FillType newFillType)

Устанавливает выбранный тип заливки на панели инструментов. В качестве аргумента принимает *newFillType* – тип заливки. Ничего не возвращает.

12. void resetTools()

Сбрасывает выбранный инструмент. Ничего не возвращает.

2.7 Класс HelpBrowser

Представляет собой текстовое поле, в котором будет отображаться выбранная страница справки в формате HTML. Наследуется от класса *QTextBrowser*. Страницы справки формата HTML объединены в сжатую коллекцию Qt для справок, которая после будет загружаться в объект *helpEngine*.

Поля класса HelpBrowser:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию
QHelpEngine helpEngine*	*private*	Хранит адрес объекта движка справки Qt.	*nullptr*

Методы класса ImageToolBar:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*HelpBrowser(const QString& collectionPath, QWidget* parent = nullptr)*
public	*QVariant*	*loadResource(int type, const QUrl &name)*
public	*QHelpContentWidget**	*getContentWidget()*
public	*QHelpIndexWidget**	*getIndexWidget()*
public	-	*~HelpBrowser()*

1. *HelpBrowser(const QString& collectionPath, QWidget* parent = nullptr)*

Конструктор, вызывается при создании объекта класса *HelpBrowser*. Вызывает конструктор базового класса и передает в него параметр *parent*. По мимо этого, происходит выделение памяти под объект *helpEngine*, в конструктор которого передается путь *collectionPath* к коллекции со страницами справки.

2. *QVariant loadResource(int type, const QUrl &name)*

Перегруженный метод. Перегрузка требуется для того, чтобы отлавливать специальные ссылки справки Qt (*qthelp://*) и для них загружать содержимое не из сети Интернет, а из объекта *helpEngine*, который содержит HTML-файлы справки.

3. *QHelpContentWidget* getContentWidget()*

Возвращает виджет со структурированным деревом страниц справки.

4. *QHelpIndexWidget* getIndexWidget()*

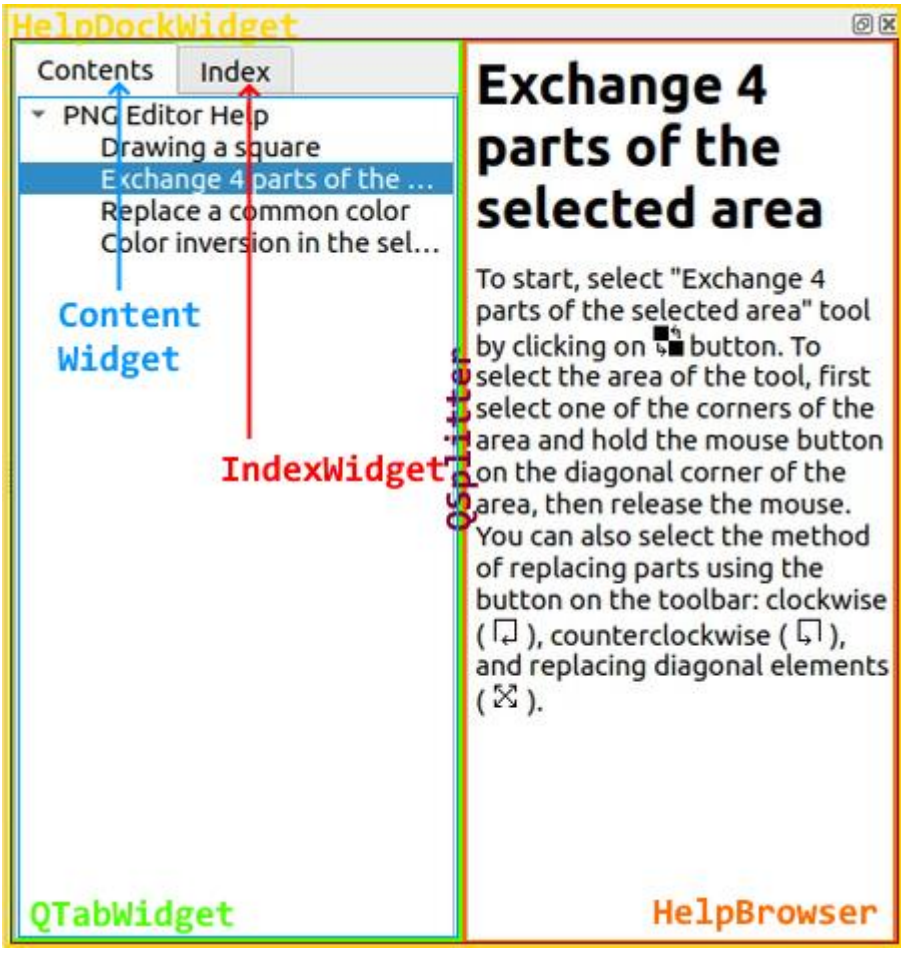
Возвращает виджет со списком всех страниц справки.

5. *~HelpBrowser()*

Деструктор, вызывается при уничтожении объекта класса *HelpBrowser*. Очищает выделенную под хранение данных класса *HelpBrowser* динамическую память.

2.8. Класс HelpDockWidget

Представляет собой окно справки, в котом содержится HelpBrowser, а также меню выборы страницы справки. Наследуется от класса *QDockWidget*. Структура выглядит следующим образом:



Поля класса HelpBrowser:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию

HelpBrowser helpBrowser*	*private*	Хранит адрес объекта класса *HelpBrowser*.	*nullptr*

Методы класса ImageToolBar:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*HelpDockWidget(QWidget* parent = nullptr)*

1. *HelpDockWidget(QWidget* parent = nullptr)*

Конструктор, вызывается при создании объекта класса *HelpDockWidget*. Вызывает конструктор базового класса и передает в него параметр *parent*.

Листинг 12.

```
HelpDockWidget::HelpDockWidget(QWidget* parent): QDockWidget(parent) {
    // Создаем внутренние элементы для виджета.
    QSplitter* horizontalSplitter = new QSplitter(Qt::Horizontal, this);
    QTabWidget* tabWidget = new QTabWidget(horizontalSplitter);
    HelpBrowser* helpBrowser = new HelpBrowser("docs/help.qhc",
horizontalSplitter);

    // Вкладываем виджеты.
    tabWidget->setMaximumWidth(320);
    tabWidget->addTab(helpBrowser->getContentWidget(), "Contents");
    tabWidget->addTab(helpBrowser->getIndexWidget(), "Index");
    horizontalSplitter->insertWidget(0, tabWidget);
    horizontalSplitter->insertWidget(1, helpBrowser);
    horizontalSplitter->hide();

    // Устанавливаем начальную страницу в браузере.
    helpBrowser->setSource(QUrl("qthelp://coursework.help/docs/index.html"));

    setWidget(horizontalSplitter);

    // Связываем слоты и сигналы.
    connect(helpBrowser->getContentWidget(), SIGNAL(linkActivated(const
QUrl&)), helpBrowser, SLOT(setSource(const QUrl&)));
    connect(helpBrowser->getIndexWidget(), SIGNAL(linkActivated(const QUrl&,
const QString&)), helpBrowser, SLOT(setSource(const QUrl&)));

    // Прячем док-виджет.
    hide();
}
```

2.9. Класс Ui::MainWindow

Данный класс был сгенерирован автоматически фреймворком Qt. Представляет собой класс с публичными полями, являющимися указателями на

виджеты, которые будут сгенерированы и настроены при вызове метода *setupUi()*. Вся информация о виджетах получается из файла *mainwindow.ui*, который формируется после создания и настройки формы в редакторе *QDesigner*.

2.10. Класс MainWindow

Представляет собой основное окно приложения. В нем содержатся другие графические элементы интерфейса программы. Данный класс является связующим звеном между классами работы с изображением и классами графического интерфейса.

Поля класса MainWindow:

Тип и название поля	Модификатор доступа	Предназначение	Значение по умолчанию
Ui::MainWindow ui*	*private*	Хранит адрес объект класса *Ui::MainWindow*, который содержит часть графических элементов интерфейса, сгенерированных при помощи Qt Designer автоматически.	*nullptr*
ImageToolBar toolbar*	*private*	Хранит адрес объекта класса *ImageToolBar*, представляющего собой панель инструментов приложения.	*nullptr*
ImageView imageViewWidget*	*private*	Хранит адрес объекта класса *ImageView*, представляющего собой окно отображения	*nullptr*

		редактируемого изображения.	
*HelpDockWidget**	*private*	Хранит адрес объекта класса *HelpDockWidget*, представляющего собой окно справки приложения.	*nullptr*
ImagePNG image*	*private*	Хранит адрес объекта класса *ImagePNG*, хранящего редактируемое изображение.	*nullptr*
QLabel coordinateLabel*	*private*	Хранит адрес объекта класса *QLabel*, представляющего собой текстовую строку в графическом интерфейсе. Данная строка отображает пользователю координаты его мыши в окне отображения изображения.	*nullptr*
QPoint point1	*private*	Вспомогательная переменная, которая хранит координаты первого нажатия пользователя на окне отображения изображения.	-

Методы класса MainWindow:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*MainWindow(QWidget *parent = nullptr)*
public	*void*	*updateCoordinates(const QPoint& point)*
public	*void*	*showError(const QString& message)*
public	*void*	*showStatusMessage(const QString& message, int timeout = 0)*
public	*void*	*deleteImage()*
public	-	*~MainWindow()*

Слоты класса MainWindow:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
private	*void*	*on_openFile_triggered()*
private	*void*	*on_saveFile_triggered()*
private	*void*	*on_saveFileAs_triggered()*
private	*void*	*on_closeFile_triggered()*
private	*void*	*on_changeColor1_clicked()*
private	*void*	*on_changeColor2_clicked()*
private	*void*	*on_changeTool1_clicked()*
private	*void*	*on_changeTool2_clicked()*
private	*void*	*on_changeTool3_clicked()*
private	*void*	*on_changeTool4_clicked()*
private	*void*	*on_changeExchangeMethod_clicked()*
private	*void*	*on_changeSquarePainting_clicked()*
private	*void*	*on_imageScene_point1Selected(QGraphicsSceneMouseEvent*)*
private	*void*	*on_imageScene_point2Selected(QGraphicsSceneMouseEvent*)*
private	*void*	*on_imageScene_mousePositionChanged(QGraphicsSceneMouseEvent*)*
private	*void*	*on_about_triggered()*
private	*void*	*on_help_triggered()*
private	*void*	*on_imageInfo_triggered()*

1. *MainWindow(QWidget *parent = nullptr)*

Конструктор, вызывается при создании объекта класса *MainWindow*. Выделяет память для хранения данных класса *MainWindow* и устанавливает побочные виджеты на главное окно приложения.

Листинг 13.

```
MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new
Ui::MainWindow) {
    // Выделяем память под объекты интерфейса и изображения.
    imageViewWidget = new ImageView(this);
    toolbar = new ImageToolBar(this);
    coordinateLabel = new QLabel(pointToString(QPoint(0, 0)), this);
    helpDockWidget = new HelpDockWidget(this);

    // Настраиваем интерфейс программы.
    ui->setupUi(this);
```

```
    ui->statusbar->addPermanentWidget(coordinateLabel);

    addToolBar(Qt::ToolBarArea::LeftToolBarArea, toolbar);
    addDockWidget(Qt::RightDockWidgetArea, helpDockWidget);
    setCentralWidget(imageViewWidget);

    // Связываем сигнал и слот выхода из программы.
    connect(ui->exit, SIGNAL(triggered()), QCoreApplication::instance(),
SLOT(quit()));
}
```

2. *void updateCoordinates(const QPoint& point)*

Обновляет значения координат курсора в текстовой строке графического интерфейса, находящейся в строке состояния. Принимает в качестве аргумента *point* – координаты курсора в окне изображения. Ничего не возвращает.

3. *void showError(const QString& message)*

Отображает окно диалога с сообщением об ошибке. Принимает в качестве аргумента *message* – сообщение об ошибке. Ничего не возвращает.

4. *void showStatusMessage(const QString& message, int timeout = 0)*

Отображает сообщение в строке состояния приложения. Принимает в качестве аргументов: *message* – сообщение и *timeout* – время отображения в миллисекундах. Ничего не возвращает.

5. *void deleteImage()*

Очищает память, занимаемую изображением и устанавливает указателю *image* значение *nullptr*. Ничего не возвращает.

6. *~MainWindow()*

Деструктор, вызывается при уничтожении объекта класса *MainWindow*. Очищает выделенную под хранение данных класса *MainWindow* динамическую память.

7. void on_openFile_triggered()

Слот вызывается при нажатии пользователем на кнопку открытия файла изображения. Отображает пользователю диалог выбора файла, после чего создает объект класса *ImagePNG* и загружает в него выбранный файл. Если до этого пользователь работал с другим файлом, то он закрывается и в случае необходимости сохраняется.

Листинг 14.

```
void MainWindow::on_openFile_triggered() {
    // Получаем путь до изображения. Для этого показываем пользователю диалог
выбора пути до изображения.
    QString filePath = QFileDialog::getOpenFileName(this, tr("Открыть файл"),
"~/",  tr("Image (*.png)"));

    // Если путь не был получен, то выходим из функции.
    if (filePath == nullptr) {
        return;
    }

    // Если пользователь до этого работал с файлом и не сохранил изменения, то
даем ему эту возможность.
    if (image != nullptr && image->isChanged()) {
        QMessageBox::StandardButton button = QMessageBox::question(this,
"Saving changes", "Save changes to " + image->getFileInfo().filePath() + "
before closing?", QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel, QMessageBox::Save);

        if (button == QMessageBox::Save) {
            on_saveFile_triggered();
            deleteImage();
            imageViewWidget->clearScene();
        } else if (button == QMessageBox::Cancel) {
            return;
        } else {
            deleteImage();
            imageViewWidget->clearScene();
        }
    }

    // Загружаем изображение.
    image = new ImagePNG;
    int error = image->load(filePath, this);

    // Обрабатываем ошибки.
    if (error == 0) {
        showStatusMessage("File is read: " + filePath);
        imageViewWidget->setImage(image);
    } else {
        switch (error) {
        case 1:
            showError("The file cannot be accessed. Make sure that the file
path is correct.");
            break;
        case 2:
            showError("The file doesn't match the PNG format.");
            break;
```

```
        case 6:
            showError("This PNG image format is not supported.");
            break;
        default:
            showError("Error opening the image.");
        }
    }
}
```

8. *void on_saveFile_triggered()*

Слот вызывается при нажатии пользователем на кнопку сохранения изображения в файл как. Сохраняет изображение *image* по пути местоположения его файла.

Листинг 15.

```
void MainWindow::on_saveFile_triggered() {
    // Проверяем, что файл открыт.
    if (image != nullptr && image->isLoaded()) {
        // Сохраняем изображение.
        int error = image->save(image-
>getFileInfo().filePath().toStdString().c_str());

        // Обрабатываем ошибки.
        if (error == 0) {
            image->updateFileInfo();
            showStatusMessage("File is saved: " + image-
>getFileInfo().filePath());
        } else {
            switch (error) {
            case 1:
                showError("The file cannot be accessed. Make sure that the
file path is correct.");
                break;
            default:
                showError("Error saving the image.");
            }
        }
    } else {
        showError("The image wasn't loaded.");
    }
}
```

9. *void on_saveFileAs_triggered()*

Слот вызывается при нажатии пользователем на кнопку сохранения изображения в файл. Отображает пользователю диалог выбора файла, после чего сохраняет изображение *image* по выбранному пути.

Листинг 16.

```
void MainWindow::on_saveFileAs_triggered() {
    // Проверяем, что изображение загруено.
```

```
    if (image == nullptr || !image->isLoaded()) {
        showError("The image wasn't loaded.");
        return;
    }

    // Получаем путь до места сохранения. Для этого показываем пользователю
диалог выбора пути.
    QString filePath = QFileDialog::getSaveFileName(this, tr("Save Image"),
"~/", tr("Image (*.png)"));

    if (filePath == nullptr) {
        return;
    }

    // Сохраняем изображение.
    int error = image->save(filePath.toStdString().c_str());

    // Обрабатываем ошибки.
    if (error == 0) {
        image->updateFileInfo(filePath);
        showStatusMessage("File is saved: " + filePath);
    } else {
        switch (error) {
        case 1:
            showError("The file cannot be accessed. Make sure that the file path
is correct.");
            break;
        default:
            showError("Error saving the image.");
        }
    }
}
```

10. void on_closeFile_triggered()

Слот вызывается при нажатии пользователем на кнопку закрытия изображения. Завершает работу с файлом, в случае необходимости сохраняет изменения.

Листинг 17.

```
void MainWindow::on_closeFile_triggered() {
    // Если пользователь не сохранил изменения, то даем ему эту возможность.
    if (image != nullptr && image->isChanged()) {
        QMessageBox::StandardButton button = QMessageBox::question(this,
"Saving changes", "Save changes to " + image->getFileInfo().filePath() + "
before closing?", QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel, QMessageBox::Save);

        if (button == QMessageBox::Save) {
            on_saveFile_triggered();
        } else if (button == QMessageBox::Cancel) {
            return;
        }
    }

    // Удаляем изображение и очищаем сцену.
    if (image != nullptr) {
```

```
        deleteImage();
        imageViewWidget->clearScene();
    }
}
```

11. void on_changeColor1_clicked()

Слот вызывается при нажатии пользователем на кнопку изменения первого цвета. Отображает пользователю диалог выбора цвета, после чего заменяет первый цвет на выбранный. По мимо этого функция изменяет иконку кнопки на выбранный цвет.

12. void on_changeColor2_clicked()

Слот вызывается при нажатии пользователем на кнопку изменения второго цвета. Отображает пользователю диалог выбора цвета, после чего заменяет второй цвет на выбранный. По мимо этого функция изменяет иконку кнопки на выбранный цвет.

13. void on_changeTool1_clicked()

Слот вызывается при нажатии пользователем на кнопку выбора первого инструмента. Устанавливает активный инструмент – первый.

14. void on_changeTool2_clicked()

Слот вызывается при нажатии пользователем на кнопку выбора первого инструмента. Устанавливает активный инструмент – второй.

15. void on_changeTool3_clicked()

Слот вызывается при нажатии пользователем на кнопку выбора первого инструмента. Вызывает функцию *replaceCommonColor()*.

16. void on_changeTool4_clicked()

Слот вызывается при нажатии пользователем на кнопку выбора первого инструмента. Устанавливает активный инструмент – четвертый.

17. void on_changeExchangeMethod_clicked()

Слот вызывается при нажатии пользователем на кнопку изменения метода обмена частей. Переключает выбранный метод обмена частей на следующий.

18. void on_changeSquarePainting_clicked()

Слот вызывается при нажатии пользователем на кнопку изменения заливки квадрата. Переключает режим заливки на противоположный выбранному.

19. void on_imageScene_point1Selected(QGraphicsSceneMouseEvent event)*

Слот вызывается при нажатии пользователем на кнопку мыши в какой-либо точке сцены. Происходит сохранение выбранной первой точки сцены в поле *point1*. Далее происходит проверка, что пользователь выбрал первый инструмент, и в этом случае происходит вызов функции *paintSquare()*.

Листинг 18.

```
void MainWindow::on_imageScene_point1Selected(QGraphicsSceneMouseEvent* event)
{
    // Проверяем, что пользователь нажал левой кнопкой мыши.
    if (event->buttons().testFlag(Qt::MouseButton::LeftButton)) {
        QPoint point = event->scenePos().toPoint();
        int selectedTool = toolbar->getSelectedTool();
        // Сохраняем выбранную точку.
        point1 = point;

        // В случае, если выбран первый инструмент, вызываем соответствующую
ему функцию.
        if (selectedTool == 1) {
            image->paintSquare(point, toolbar->lineWidthSpinBox->value(),
toolbar->squareSizeSpinBox->value(),
                               toolbar->getFirstColor(), toolbar-
>getFillType(), toolbar->getSecondColor());
            // Обновляем сцену (для показа изменений).
            imageViewWidget->setImage(image);
            showStatusMessage("Operation complete: Paint square at " +
pointToString(point));
        }
    }
```

```
}
```

20. void on_imageScene_point2Selected(QGraphicsSceneMouseEvent event)*

Слот вызывается при отпускании пользователем кнопки мыши в какой-либо точке сцены. Происходит проверка, что пользователь выбрал второй или четвертый инструмент, и в этом случае происходит вызов соответствующих этим инструментам функций.

Листинг 18.

```
void MainWindow::on_imageScene_point2Selected(QGraphicsSceneMouseEvent* event)
{
    QPoint point = event->scenePos().toPoint();
    int selectedTool = toolbar->getSelectedTool();

    // Производим корректировку точек (так, чтобы первая точка области была
верхней левой, а вторая – правой нижней).
    if (point.x() < point1.x()) {
        int x = point.x();
        point.setX(point1.x());
        point1.setX(x);
    }

    if (point.y() < point1.y()) {
        int y = point.y();
        point.setY(point1.y());
        point1.setY(y);
    }

    // Получение первой точки области происходит при помощи слота
on_imageScene_point1Selected().

    // Вызываем соответствующие выбранным инструментам функции.
    if (selectedTool == 4) {
        image->invertColors(point1, point);
        imageViewWidget->setImage(image);
        showStatusMessage("Operation complete: Invert color in area " +
pointToString(point1) + " to " + pointToString(point));
    } else if (selectedTool == 2) {
        image->changeImageAreas(point1, point, toolbar-
>getExchangingMethod());
        imageViewWidget->setImage(image);
        showStatusMessage("Operation complete: Exchange parts in area " +
pointToString(point1) + " to " + pointToString(point));
    }
}
```

21. void on_imageScene_mousePositionChanged(QGraphicsSceneMouseEvent event)*

Слот вызывается при перемещении указателя мыши на сцене. Обновляет значения координат указателя мыши в строке состояния.

22. void on_about_triggered()

Слот вызывается при нажатии пользователем на кнопку “About” в меню приложения. Показывает диалог с информацией о приложении.

23. void on_help_triggered()

Слот вызывается при нажатии пользователем на кнопку “Help” в меню приложения. Показывает виджет со справкой приложения.

24. void on_imageInfo_triggered()

Слот вызывается при нажатии пользователем на кнопку “Image Info” в меню приложения. Показывает виджет с информацией о изображении.

2.11. Перечисления

1. *enum ToolType*

Хранит тип инструмента.

Элемент	Значение	Пояснение
NONE_TOOL	0	Инструмент не выбран.
SQUARE_PAINT_TOOL	1	Рисование квадрата.
EXCHANGING_PARTS_TOOL	2	Обмен частей местами.
CHANGE_COMMON_COLOR_TOOL	3	Изменение самого часто встречающегося цвета.
INVERT_COLOR_TOOL	4	Инвертирование цвета.

2. *enum ExchangingMethod*

Хранит метод обмена частей.

Элемент	Значение	Пояснение
CLOCKWISE_METHOD	1	Обмен по часовой стрелке.
ANTICLOCKWISE_METHOD	2	Обмен против часовой стрелки.
DIAGONAL_METHOD	3	Обмен по диагонали.

3. *enum FillType*

Хранит тип заливки.

Элемент	Значение	Пояснение
UNFILLED	1	Без заливки.
FILLED	2	С заливкой.

2.12. Вспомогательные функции

1. *QString pointToString(const QPoint& point)*

Функция преобразует объект класса QPoint к объекту класса QString. В качестве аргумента принимает *point* – объект точки. Возвращает объект класса QString, содержащий информацию о координатах точки *point*.

2. *QString colorToString(const QColor& color)*

Функция преобразует объект класса QColor к объекту класса QString. В качестве аргумента принимает *color* – объект цвета. Возвращает объект класса QString, содержащий информацию о компонентах цвета *color*.

3. *QColor getPixel(png_bytep pixel)*

Функция возвращает цвет пикселя *pixel*, который принимается в качестве аргумента.

4. *void setPixel(png_bytep pixel, const QColor& color)*

Функция устанавливает новый цвет *color* пикселю *pixel*, которые принимаются в качестве аргумента.

5. *QColor getColorFromPixel(png_bytep pixel)*

Функция возвращает цвет пикселя *pixel*, который принимается в качестве аргумента.

6. *void invertPixelColor(png_bytep pixel)*

Функция инвертирует цвет пикселя *pixel*, который принимается в качестве аргумента.

7. *QToolButton* createToolButtonColor(QWidget* parent, QString objectName, QString toolTip, const QColor& color, bool checkable)*

Создает в динамической памяти объект класса *QToolButton* с соответствующими параметрами, которые передаются в качестве аргументов. Возвращает указатель на созданный в памяти объект.

8. *QToolButton* createToolButton(QWidget* parent, QString objectName, QString toolTip, QString path, bool checkable)*

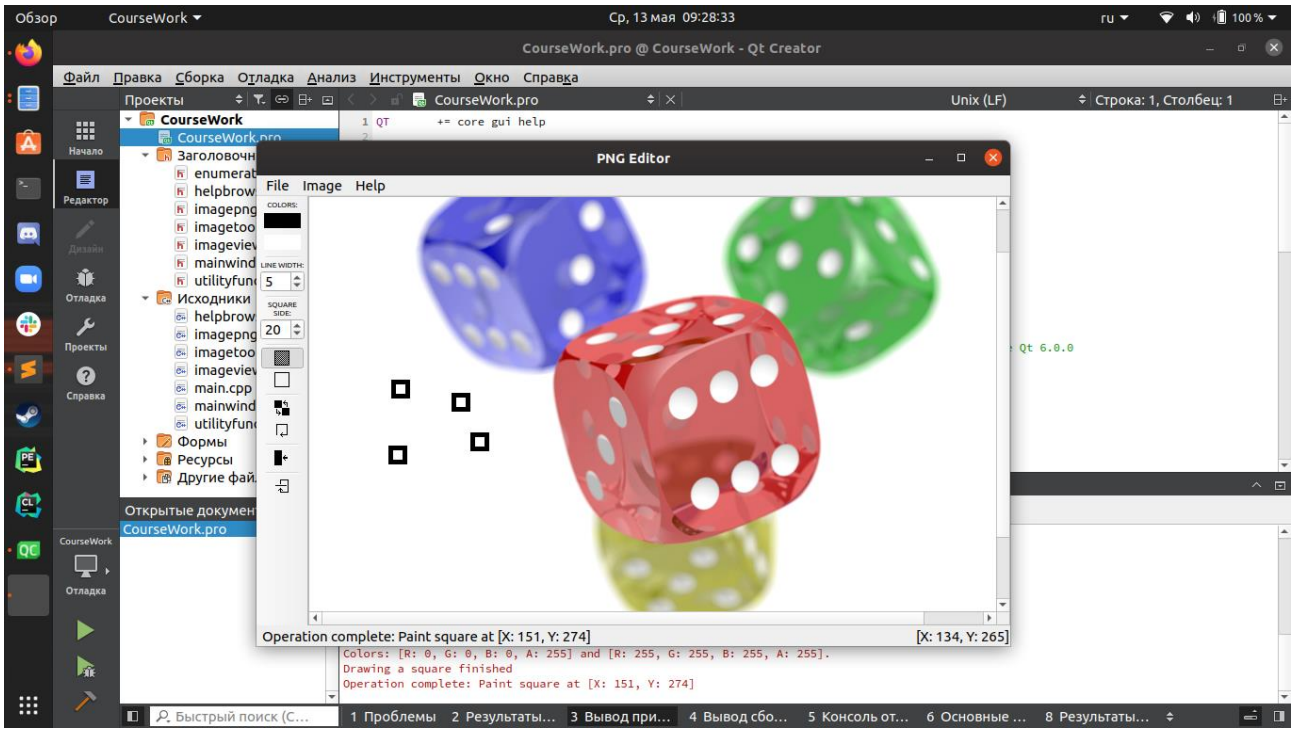
Создает в динамической памяти объект класса *QToolButton* с соответствующими параметрами, которые передаются в качестве аргументов. Возвращает указатель на созданный в памяти объект.

9. *QSpinBox* createSpinBox(QWidget* parent, int initialValue, int minValue, int maxValue, QString toolTip)*

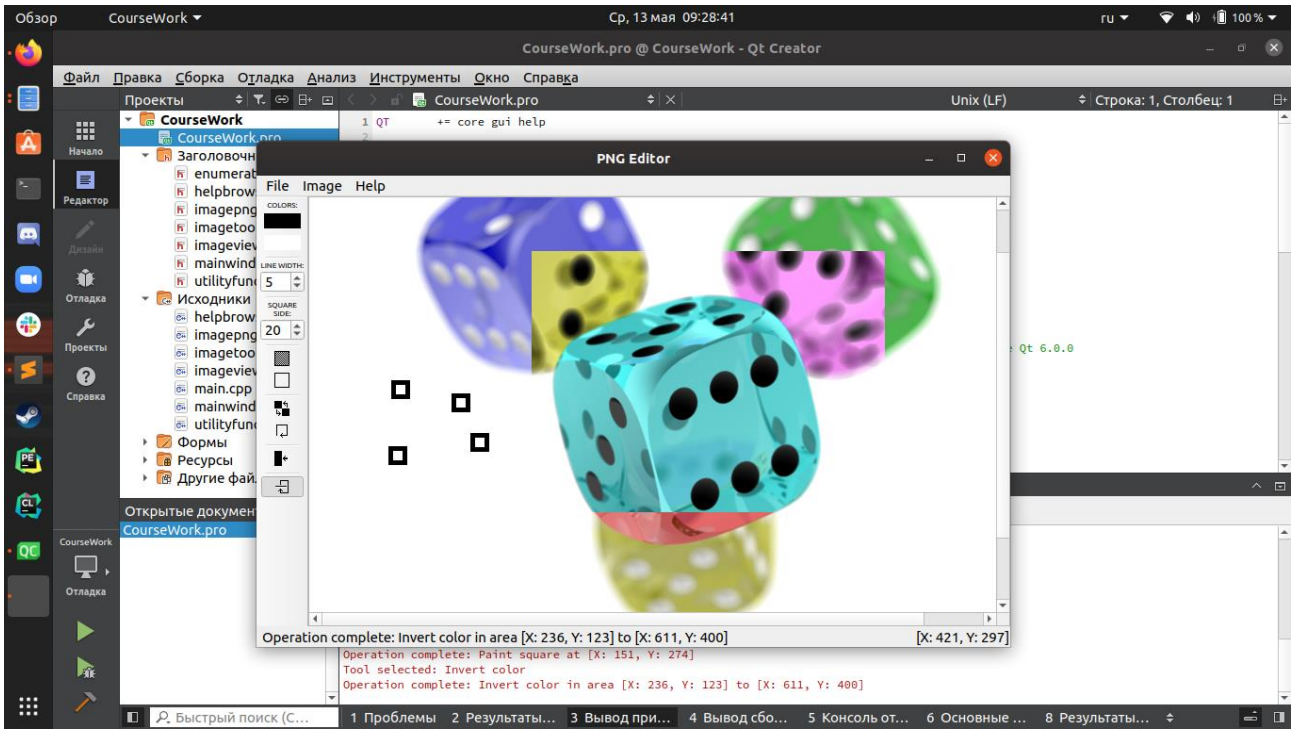
Создает в динамической памяти объект класса *QSpinBox* с соответствующими параметрами, которые передаются в качестве аргументов. Возвращает указатель на созданный в памяти объект.

3. ТЕСТИРОВАНИЕ РАБОТЫ ПРОГРАММЫ

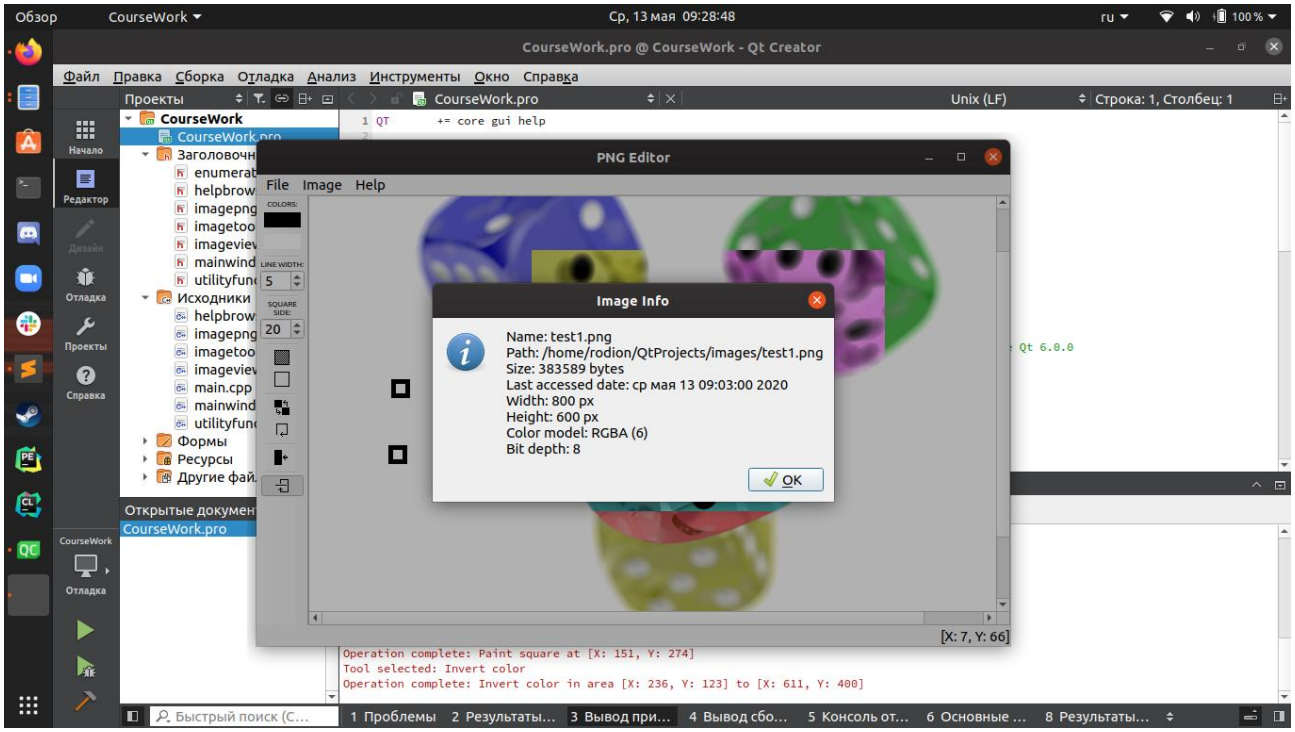
Тест №1: Рисование квадрата



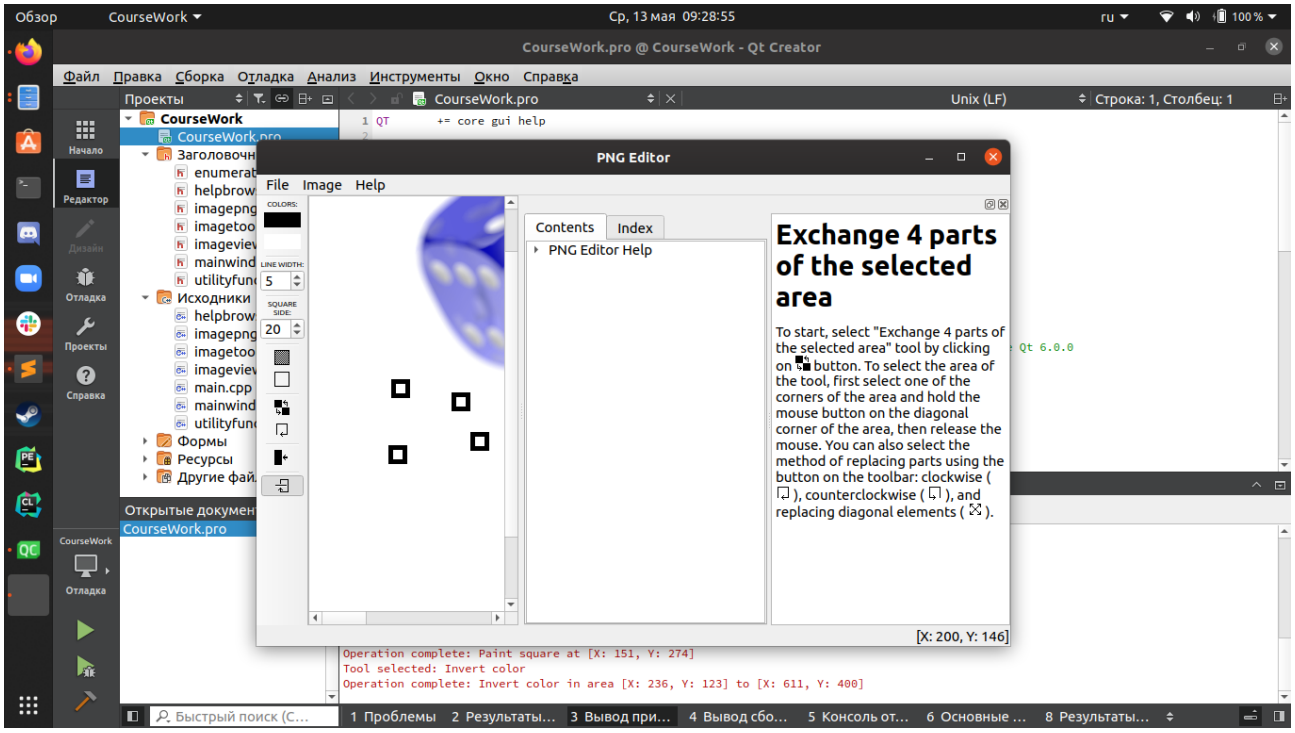
Тест №2: Инвертирование цвета



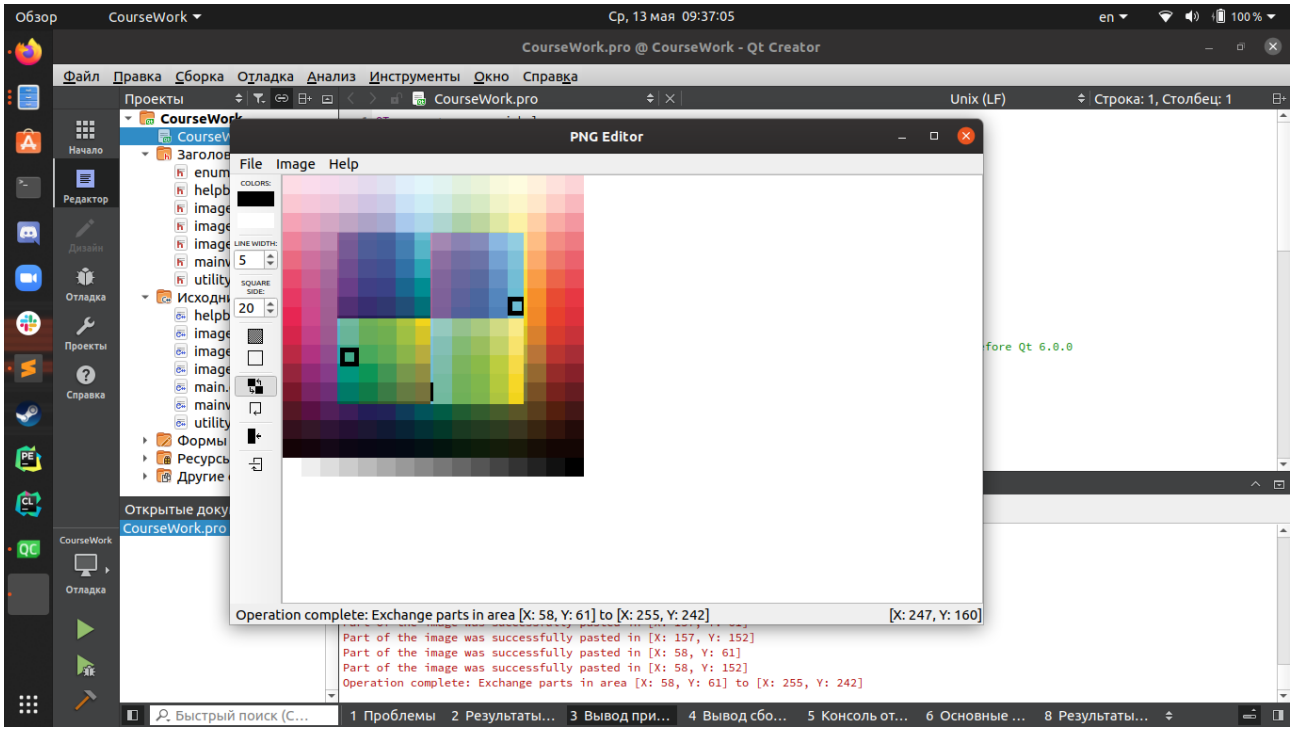
Тест №3: Информация о изображении



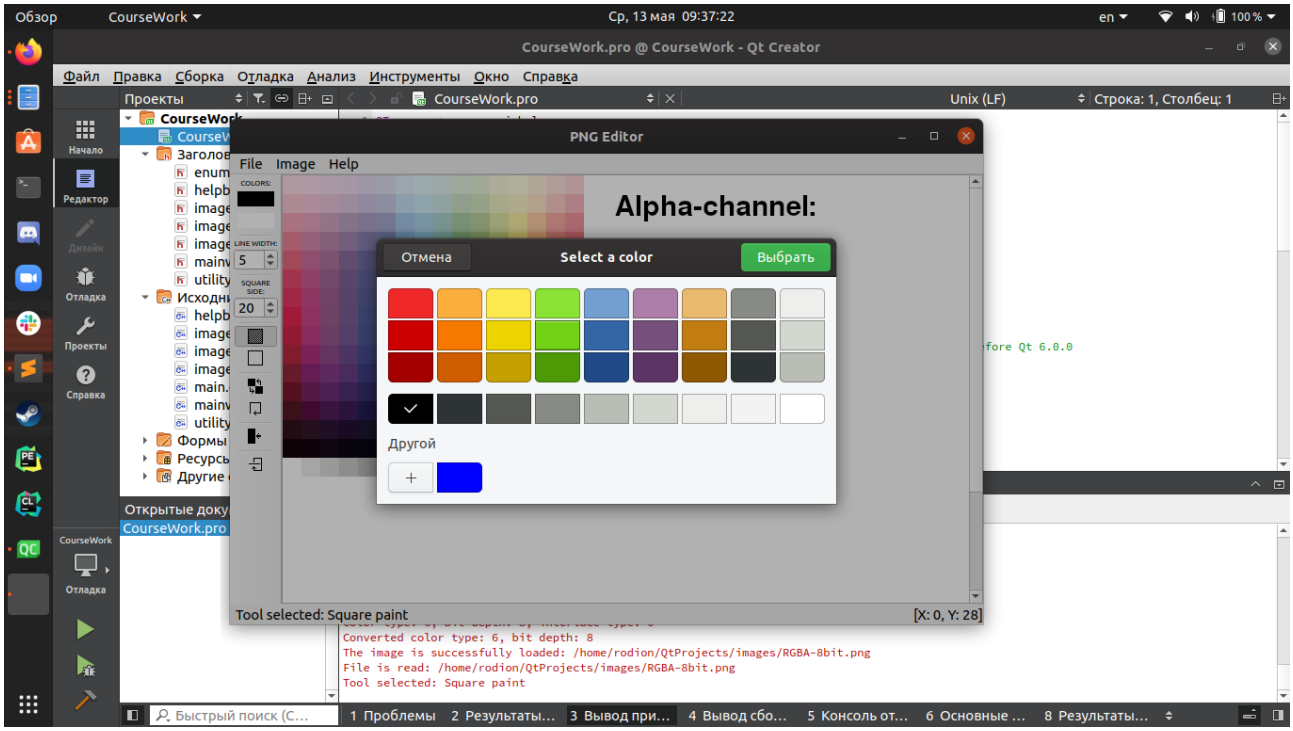
Тест №4: Справка



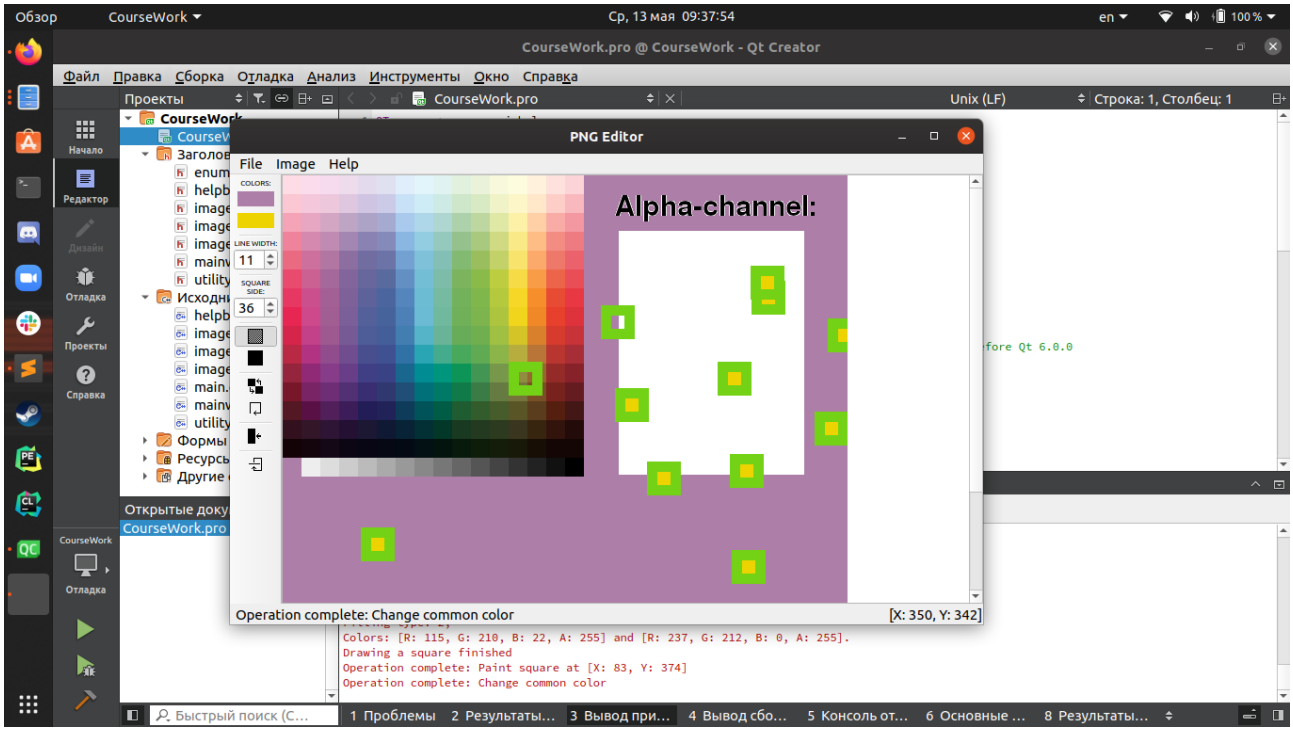
Тест №5: Поменять местами части



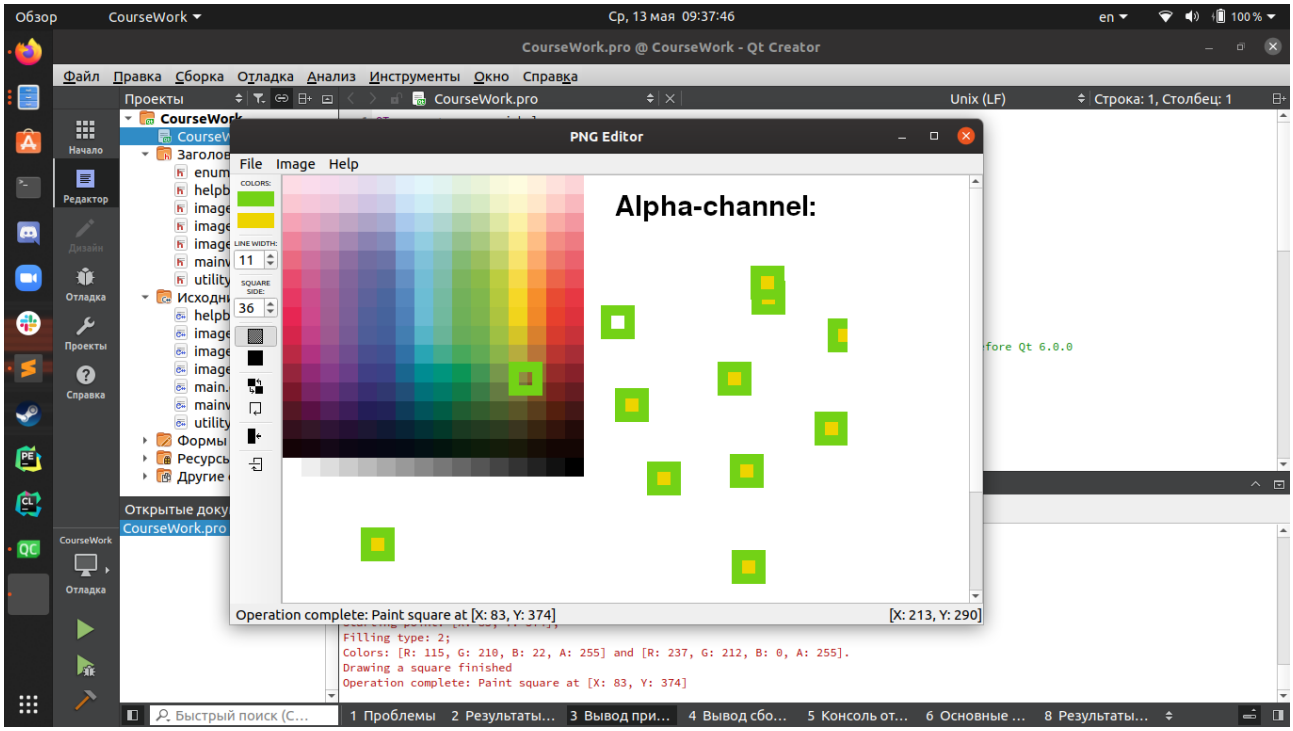
Тест №6: Выбор цвета



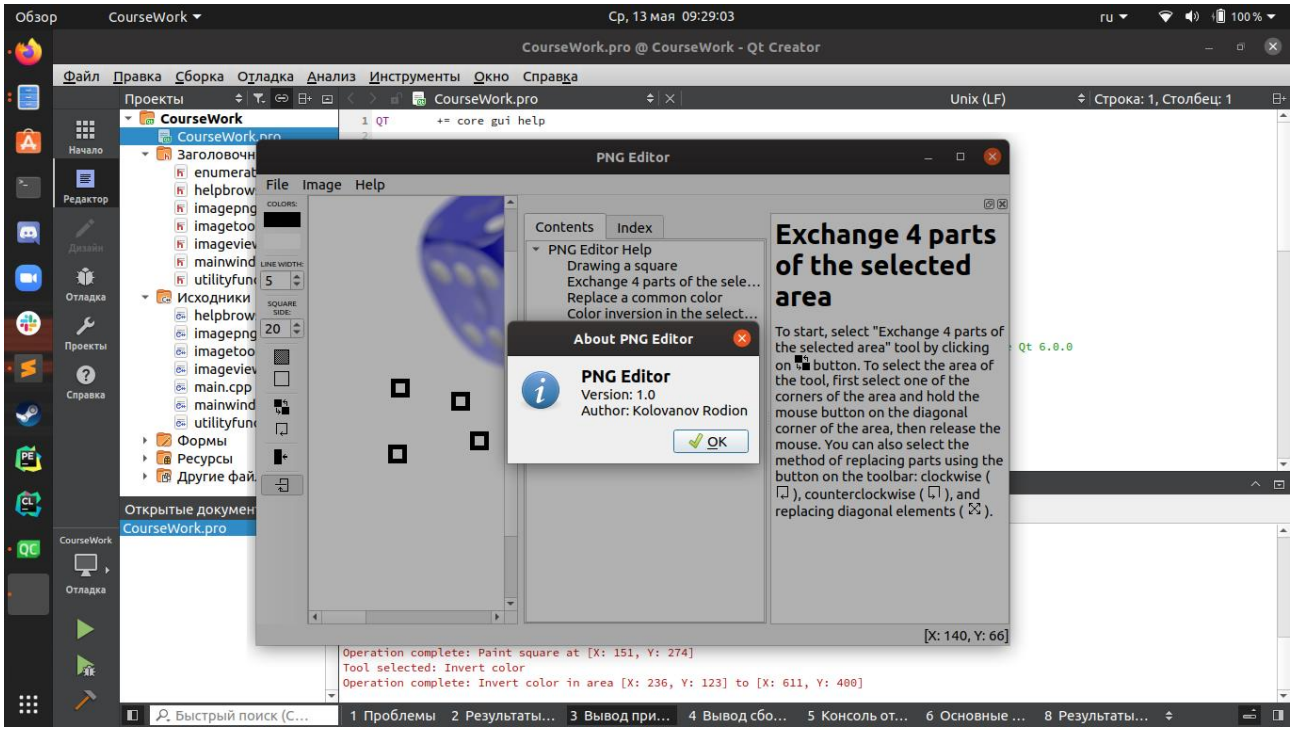
Тест №7: Замена самого часто встречающегося цвета



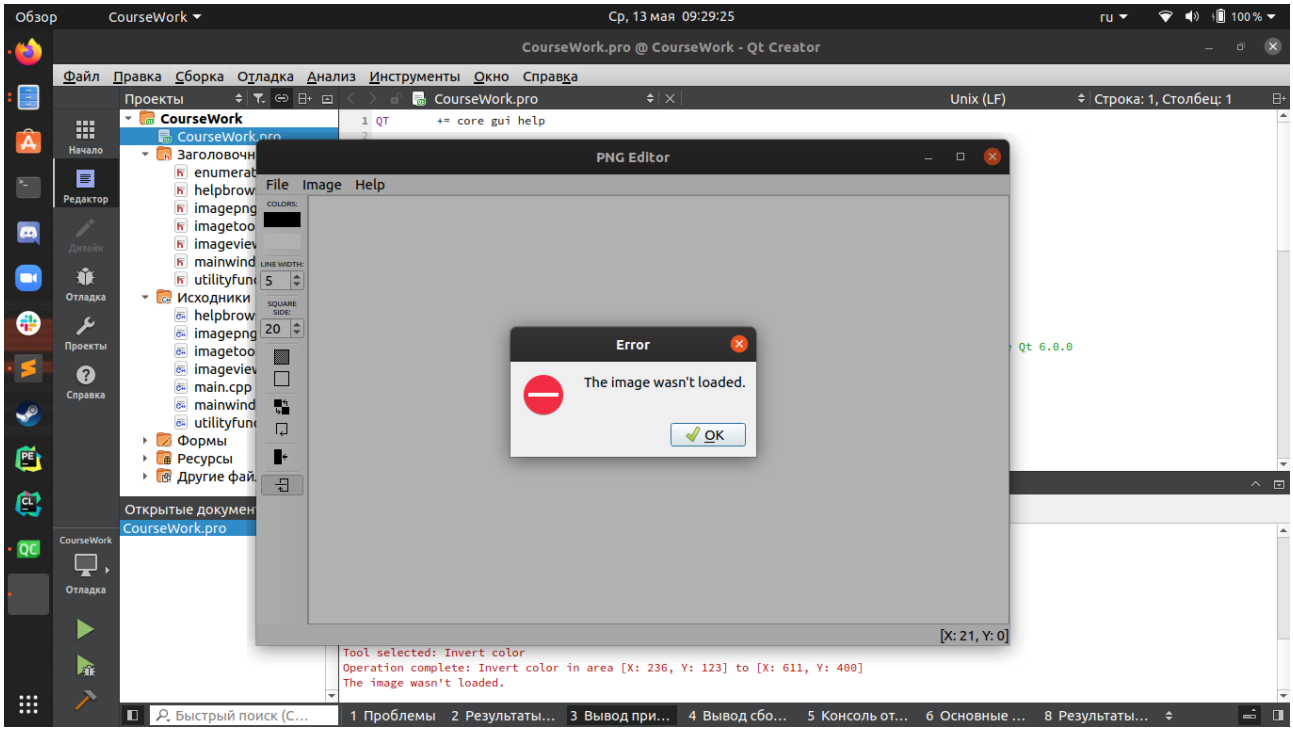
Тест №8: Рисование квадрата с заливкой

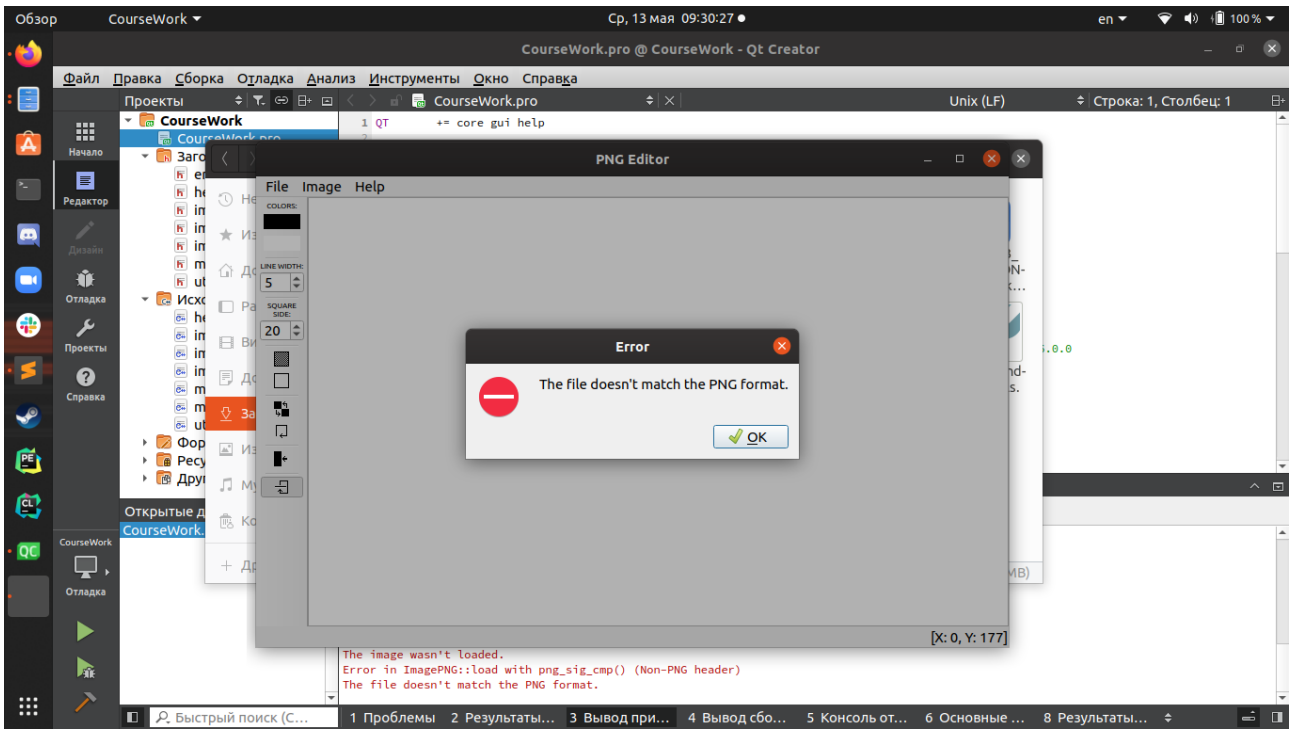


Тест №9: Окно информации о программе



Тест №10: Обработка ошибок




Обзор
CourseWork
Ср, 13 мая 09:30:27
en
100 %
CourseWork.pro @ CourseWork - Qt Creator
Файл Правка Сборка Отладка Анализ Инструменты Окно Справка
Проекты
CourseWork.pro
Unix (LF)
Строка: 1, Столбец: 1
1 QT += core gui help
Начало
Редактор
Дизайн
Отладка
Проекты
Справка
CourseWork
Отладка
PNG Editor
File Image Help
COLORS:
LINE WIDTH:
5
SQUARE SIDE:
20
[X: 0, Y: 177]
Error
The file doesn't match the PNG format.
OK
The image wasn't loaded.
Error in ImagePNG::load with png_sig_cmp() (Non-PNG header)
The file doesn't match the PNG format.
Открытые д
CourseWork.
Быстрый поиск (C...
1 Проблемы
2 Результаты...
3 Вывод при...
4 Вывод сбо...
5 Консоль от...
6 Основные ...
8 Результаты...

ЗАКЛЮЧЕНИЕ

В ходе работы над поставленным заданием удалось разработать программу с графическим интерфейсом, способную считывать, обрабатывать и сохранять изображения формата PNG. Программа была успешно протестирована на работоспособность.

Программа обладает следующим функционалом:

- Рисование квадрата;
- Деление области на 4 части и их обмен местами;
- Замена самого часто встречающегося цвета на другой цвет;
- Инвертирование цвета в заданной области.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Qt Documentation // Qt Docs. URL: https://doc.qt.io/ (дата обращения: 10.05.2020).

2. libpng Home Page // libpng.org. URL: http://www.libpng.org/pub/png/libpng.html (дата обращения: 10.05.2020).

3. Standart C++ Library reference // cplusplus.com. URL: http://www.cplusplus.com/reference/ (дата обращения: 10.05.2020).

4. CS106B Style Guide // Stanford University. URL: http://stanford.edu/class/archive/cs/cs106b/cs106b.1158/styleguide.shtml (дата обращения: 10.05.2020).

5. PNG — not GIF! // Хабр. URL: https://habr.com/ru/post/130472/ (дата обращения: 10.05.2020).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл *src/utilityfunctions.h*:

```
#ifndef UTILITY_FUNCTIONS_H
#define UTILITY_FUNCTIONS_H

#include "png.h"
#include <QString>
#include <QPoint>
#include <QColor>
#include <QToolButton>
#include <QSpinBox>


QString pointToString(const QPoint& point);
QString colorToString(const QColor& color);
QColor getPixel(png_bytep pixel);
void setPixel(png_bytep pixel, const QColor& color);
QColor getColorFromPixel(png_bytep pixel);
void invertPixelColor(png_bytep pixel);
QToolButton* createToolButtonColor(QWidget* parent, QString objectName,
QString toolTip, const QColor& color, bool checkable);
QToolButton* createToolButton(QWidget* parent, QString objectName,
QString toolTip, QString path, bool checkable);
QSpinBox* createSpinBox(QWidget* parent, int initialValue, int minValue,
int maxValue, QString toolTip);


#endif // UTILITY_FUNCTIONS_H
```

Файл *src/utilityfunctions.cpp*:

```
#include <utilityfunctions.h>


QString pointToString(const QPoint& point) {
    return "[X: " + QString::number(point.x()) + ", Y: " +
QString::number(point.y()) + "]";
}

QString colorToString(const QColor& color) {
    return "[R: " + QString::number(color.red()) + ", G: " +
QString::number(color.green()) + ", B: " + QString::number(color.blue())+
", A: " + QString::number(color.alpha()) + "]";
}

QColor getPixel(png_bytep pixel) {
    if (pixel != nullptr) {
        return QColor(pixel[0], pixel[1], pixel[2], pixel[3]);
    }

    return QColor();
```

```
}

void setPixel(png_bytep pixel, const QColor& color) {
    if (pixel != nullptr) {
        pixel[0] = color.red();
        pixel[1] = color.green();
        pixel[2] = color.blue();
        pixel[3] = color.alpha();
    }
}

QColor getColorFromPixel(png_bytep pixel) {
    return QColor(pixel[0], pixel[1], pixel[2], pixel[3]);
}

void invertPixelColor(png_bytep pixel) {
    if (pixel != nullptr) {
        pixel[0] = 255 - pixel[0];
        pixel[1] = 255 - pixel[1];
        pixel[2] = 255 - pixel[2];
    }
}

QToolButton* createToolButtonColor(QWidget* parent, QString objectName,
QString toolTip, const QColor& color, bool checkable) {
    QPixmap pixmap(39, 16);
    QToolButton* button = new QToolButton(parent);

    pixmap.fill(color);
    button->setObjectName(objectName);
    button->setToolTip(toolTip);
    button->setIcon(QIcon(pixmap));

    if (checkable) {
        button->setCheckable(checkable);
    }

    return button;
}

QToolButton* createToolButton(QWidget* parent, QString objectName,
QString toolTip, QString path, bool checkable) {
    QToolButton* button = new QToolButton(parent);

    button->setObjectName(objectName);
    button->setToolTip(toolTip);
    button->setIcon(QIcon(path));

    if (checkable) {
        button->setCheckable(checkable);
    }

    return button;
}

QSpinBox* createSpinBox(QWidget* parent, int initialValue, int minValue,
int maxValue, QString toolTip) {
```

```
    QSpinBox* spinBox = new QSpinBox(parent);

    spinBox->setValue(initialValue);
    spinBox->setRange(minValue, maxValue);
    spinBox->setToolTip(toolTip);

    return spinBox;
}
```

Файл *src/mainwindow.h*:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include "imageview.h"
#include "imagetoolbar.h"
#include "imagepng.h"
#include "helpbrowser.h"
#include "enumerations.h"

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE


class MainWindow: public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    void updateCoordinates(const QPoint& point);
    void showError(const QString& message);
    void showStatusMessage(const QString& message, int timeout = 0);
    void deleteImage();
    ~MainWindow();

private slots:
    void on_openFile_triggered();
    void on_saveFile_triggered();
    void on_saveFileAs_triggered();
    void on_closeFile_triggered();
    void on_changeColor1_clicked();
    void on_changeColor2_clicked();
    void on_changeTool1_clicked();
    void on_changeTool2_clicked();
    void on_changeTool3_clicked();
    void on_changeTool4_clicked();
    void on_changeExchangeMethod_clicked();
    void on_changeSquarePainting_clicked();
    void on_imageScene_point1Selected(QGraphicsSceneMouseEvent* event);
    void on_imageScene_point2Selected(QGraphicsSceneMouseEvent* event);
```

```
    void on_imageScene_mousePositionChanged(QGraphicsSceneMouseEvent*
event);
    void on_about_triggered();
    void on_help_triggered();
    void on_imageInfo_triggered();

private:
    Ui::MainWindow* ui = nullptr;
    ImageToolBar* toolbar = nullptr;
    ImageView* imageViewWidget = nullptr;
    HelpDockWidget* helpDockWidget = nullptr;
    ImagePNG* image = nullptr;
    QLabel* coordinateLabel = nullptr;
    QPoint point1;
};


#endif // MAINWINDOW_H
```

Файл *src/mainwindow.cpp*:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "utilityfunctions.h"
#include <QMessageBox>
#include <QFileDialog>
#include <QColorDialog>
#include <QGraphicsSceneMouseEvent>
#include <QGridLayout>
#include <QDateTime>
#include <QDebug>


MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new
Ui::MainWindow) {
    imageViewWidget = new ImageView(this);
    toolbar = new ImageToolBar(this);
    coordinateLabel = new QLabel(pointToString(QPoint(0, 0)), this);
    helpDockWidget = new HelpDockWidget(this);

    ui->setupUi(this);
    ui->statusbar->addPermanentWidget(coordinateLabel);

    addToolBar(Qt::ToolBarArea::LeftToolBarArea, toolbar);
    addDockWidget(Qt::RightDockWidgetArea, helpDockWidget);
    setCentralWidget(imageViewWidget);

    connect(ui->exit, SIGNAL(triggered()), QCoreApplication::instance(),
SLOT(quit()));
}

void MainWindow::updateCoordinates(const QPoint& point) {
    coordinateLabel->setText(pointToString(point));
}

void MainWindow::showError(const QString& message) {
```

```
    qDebug("%s", message.toStdString().c_str());
    QMessageBox::critical(this, "Error", message);
}

void MainWindow::showStatusMessage(const QString& message, int timeout) {
    qDebug("%s", message.toStdString().c_str());
    ui->statusbar->showMessage(message, timeout);
}

void MainWindow::deleteImage() {
    delete image;
    image = nullptr;
}

MainWindow::~MainWindow() {
    deleteImage();
    delete ui;
}

void MainWindow::on_openFile_triggered() {
    QString filePath = QFileDialog::getOpenFileName(this, tr("Открыть
файл"), "~/",  tr("Image (*.png)"));

    if (filePath == nullptr) {
        return;
    }

    if (image != nullptr && image->isChanged()) {
        QMessageBox::StandardButton button = QMessageBox::question(this,
"Saving changes", "Save changes to " + image->getFileInfo().filePath() +
" before closing?", QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel, QMessageBox::Save);

        if (button == QMessageBox::Save) {
            on_saveFile_triggered();
            deleteImage();
            imageViewWidget->clearScene();
        } else if (button == QMessageBox::Cancel) {
            return;
        } else {
            deleteImage();
            imageViewWidget->clearScene();
        }
    }

    image = new ImagePNG;
    int error = image->load(filePath, this);

    if (error == 0) {
        showStatusMessage("File is read: " + filePath);
        imageViewWidget->setImage(image);
    } else {
        switch (error) {
        case 1:
            showError("The file cannot be accessed. Make sure that the
file path is correct.");
            break;
```

```
        case 2:
            showError("The file doesn't match the PNG format.");
            break;
        case 6:
            showError("This PNG image format is not supported.");
            break;
        default:
            showError("Error opening the image.");
        }
    }
}

void MainWindow::on_saveFile_triggered() {
    if (image != nullptr && image->isLoaded()) {
        int error = image->save(image-
>getFileInfo().filePath().toStdString().c_str());

        if (error == 0) {
            image->updateFileInfo();
            showStatusMessage("File is saved: " + image-
>getFileInfo().filePath());
        } else {
            switch (error) {
            case 1:
                showError("The file cannot be accessed. Make sure that
the file path is correct.");
                break;
            default:
                showError("Error saving the image.");
            }
        }
    } else {
        showError("The image wasn't loaded.");
    }
}

void MainWindow::on_saveFileAs_triggered() {
    if (image == nullptr || !image->isLoaded()) {
        showError("The image wasn't loaded.");
        return;
    }

    QString filePath = QFileDialog::getSaveFileName(this, tr("Save
Image"), "~/",  tr("Image (*.png)"));

    if (filePath == nullptr) {
        return;
    }

    int error = image->save(filePath.toStdString().c_str());

    if (error == 0) {
        image->updateFileInfo(filePath);
        showStatusMessage("File is saved: " + filePath);
    } else {
        switch (error) {
        case 1:
```

```
            showError("The file cannot be accessed. Make sure that the
file path is correct.");
            break;
        default:
            showError("Error saving the image.");
        }
    }
}

void MainWindow::on_closeFile_triggered() {
    if (image != nullptr && image->isChanged()) {
        QMessageBox::StandardButton button = QMessageBox::question(this,
"Saving changes", "Save changes to " + image->getFileInfo().filePath() +
" before closing?", QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel, QMessageBox::Save);

        if (button == QMessageBox::Save) {
            on_saveFile_triggered();
        } else if (button == QMessageBox::Cancel) {
            return;
        }
    }

    if (image != nullptr) {
        deleteImage();
        imageViewWidget->clearScene();
    }
}

void MainWindow::on_changeColor1_clicked() {
    QColor color = QColorDialog::getColor(toolbar->getFirstColor(), this,
"Select a color");

    if (color.isValid()) {
        QPixmap pixmap(39, 16);

        pixmap.fill(color);

        toolbar->setFirstColor(color);
        toolbar->changeColor1_button->setIcon(QIcon(pixmap));
    }
}

void MainWindow::on_changeColor2_clicked() {
    QColor color = QColorDialog::getColor(toolbar->getFirstColor(), this,
"Select a color");

    if (color.isValid()) {
        QPixmap pixmap(39, 16);

        pixmap.fill(color);

        toolbar->setSecondColor(color);
        toolbar->changeColor2_button->setIcon(QIcon(pixmap));
    }
}
```

```
void MainWindow::on_changeTool1_clicked() {
    toolbar->resetTools();

    if (image != nullptr && image->isLoaded()) {
        toolbar->changeTool1_button->setChecked(true);
        toolbar->setToolType(ToolType::SQUARE_PAINT_TOOL);
        showStatusMessage("Tool selected: Square paint");
    } else {
        showError("The image wasn't loaded.");
    }
}

void MainWindow::on_changeTool2_clicked() {
    toolbar->resetTools();

    if (image != nullptr && image->isLoaded()) {
        toolbar->changeTool2_button->setChecked(true);
        toolbar->setToolType(ToolType::EXCHANGING_PARTS_TOOL);
        showStatusMessage("Tool selected: Exchanging parts");
    } else {
        showError("The image wasn't loaded.");
    }
}

void MainWindow::on_changeTool3_clicked() {
    if (image != nullptr && image->isLoaded()) {
        image->replaceCommonColor(toolbar->getFirstColor());
        imageViewWidget->setImage(image);
        showStatusMessage("Operation complete: Change common color");
    } else {
        showError("The image wasn't loaded.");
    }
}

void MainWindow::on_changeTool4_clicked() {
    toolbar->resetTools();

    if (image != nullptr && image->isLoaded()) {
        toolbar->changeTool4_button->setChecked(true);
        toolbar->setToolType(ToolType::INVERT_COLOR_TOOL);
        showStatusMessage("Tool selected: Invert color");
    } else {
        showError("The image wasn't loaded.");
    }
}

void MainWindow::on_changeExchangeMethod_clicked() {
    if (toolbar->getExchangingMethod() == 1) {
        toolbar->changeExchangeMethod_button-
>setIcon(QIcon(":/icons/exchanging_2.png"));
        toolbar-
>setExchangingMethod(ExchangingMethod::ANTICLOCKWISE_METHOD);
        showStatusMessage("Exchange method selected: Anticlockwise");
    } else if (toolbar->getExchangingMethod() == 2) {
        toolbar->changeExchangeMethod_button-
>setIcon(QIcon(":/icons/exchanging_3.png"));
        toolbar->setExchangingMethod(ExchangingMethod::DIAGONAL_METHOD);
```

```
        showStatusMessage("Exchange method selected: Diagonal");
    } else {
        toolbar->changeExchangeMethod_button-
>setIcon(QIcon(":/icons/exchanging_1.png"));
        toolbar->setExchangingMethod(ExchangingMethod::CLOCKWISE_METHOD);
        showStatusMessage("Exchange method selected: Clockwise");
    }
}

void MainWindow::on_changeSquarePainting_clicked() {
    if (toolbar->getFillType() == FillType::UNFILLED) {
        toolbar->changeSquarePainting_button-
>setIcon(QIcon(":/icons/square_painted.png"));
        toolbar->setFillType(FillType::FILLED);
        showStatusMessage("Square painting selected: Filled");
    } else {
        toolbar->changeSquarePainting_button-
>setIcon(QIcon(":/icons/square_unpainted.png"));
        toolbar->setFillType(FillType::UNFILLED);
        showStatusMessage("Square painting selected: Unfilled");
    }
}

void MainWindow::on_imageScene_point1Selected(QGraphicsSceneMouseEvent*
event) {
    if (event->buttons().testFlag(Qt::MouseButton::LeftButton)) {
        QPoint point = event->scenePos().toPoint();
        int selectedTool = toolbar->getSelectedTool();
        point1 = point;

        if (selectedTool == 1) {
            image->paintSquare(point, toolbar->lineWidthSpinBox->value(),
toolbar->squareSizeSpinBox->value(),
                               toolbar->getFirstColor(), toolbar-
>getFillType(), toolbar->getSecondColor());
            imageViewWidget->setImage(image);
            showStatusMessage("Operation complete: Paint square at " +
pointToString(point));
        }
    }
}

void MainWindow::on_imageScene_point2Selected(QGraphicsSceneMouseEvent*
event) {
    QPoint point = event->scenePos().toPoint();
    int selectedTool = toolbar->getSelectedTool();

    if (point.x() < point1.x()) {
        int x = point.x();
        point.setX(point1.x());
        point1.setX(x);
    }

    if (point.y() < point1.y()) {
        int y = point.y();
        point.setY(point1.y());
        point1.setY(y);
```

```
    }

    if (selectedTool == 4) {
        image->invertColors(point1, point);
        imageViewWidget->setImage(image);
        showStatusMessage("Operation complete: Invert color in area " +
pointToString(point1) + " to " + pointToString(point));
    } else if (selectedTool == 2) {
        image->changeImageAreas(point1, point, toolbar-
>getExchangingMethod());
        imageViewWidget->setImage(image);
        showStatusMessage("Operation complete: Exchange parts in area " +
pointToString(point1) + " to " + pointToString(point));
    }
}

void
MainWindow::on_imageScene_mousePositionChanged(QGraphicsSceneMouseEvent*
event) {
    QPoint point(event->scenePos().toPoint());
    static QPoint previousPosition = QPoint(-100000, -100000);

    if (point != previousPosition) {
        coordinateLabel->setText(pointToString(point));
        previousPosition = point;
    }
}

void MainWindow::on_about_triggered() {
    QMessageBox* aboutBox = new QMessageBox(this);
    aboutBox->setWindowTitle("About PNG Editor");
    aboutBox->setText("<b style='font-size: 18px;'>PNG
Editor</b><br>Version: 1.0<br>Author:  Kolovanov Rodion");
    aboutBox->setIcon(QMessageBox::Icon::Information);
    aboutBox->show();
}

void MainWindow::on_help_triggered() {
    helpDockWidget->show();
}

void MainWindow::on_imageInfo_triggered() {
    if (image != nullptr && image->isLoaded()) {
        QString info = "Name: " + image->getFileInfo().fileName() +
                       "\nPath: " + image->getFileInfo().filePath()  +
                       "\nSize: " + QString::number(image-
>getFileInfo().size()) + " bytes" +
                       "\nLast accessed date: " + image-
>getFileInfo().lastRead().toString() +
                       "\nWidth: " + QString::number(image->getWidth()) +
" px" +
                       "\nHeight: " + QString::number(image->getHeight())
+ " px" +
                       "\nColor model: " + image->getColorModelName() + "
(" + QString::number(image->getColorType()) + ")" +
                       "\nBit depth: " + QString::number(image-
>getBitDepth());
```

```
        QMessageBox* infoWidget = new QMessageBox(this);
        infoWidget->setWindowTitle("Image Info");
        infoWidget->setText(info);
        infoWidget->setIcon(QMessageBox::Icon::Information);
        infoWidget->show();
    } else {
        showError("The image wasn't loaded.");
    }
}
```

Файл *src/main.cpp*:

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    MainWindow mainWindow;
    mainWindow.show();

    return app.exec();
}
```

Файл *src/imageview.h*:

```
#ifndef IMAGEVIEW_H
#define IMAGEVIEW_H

#include <QGraphicsView>
#include <imagepng.h>


class ImageScene: public QGraphicsScene {
    Q_OBJECT

public:
    ImageScene(qreal x, qreal y, qreal width, qreal height, QObject*
parent = nullptr);
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event) override;
    void mousePressEvent(QGraphicsSceneMouseEvent* event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent* event) override;

signals:
    void point1Selected(QGraphicsSceneMouseEvent* event);
    void point2Selected(QGraphicsSceneMouseEvent* event);
    void mousePositionChanged(QGraphicsSceneMouseEvent* event);
};


class ImageView: public QGraphicsView {
    Q_OBJECT
```

```
public:
    explicit ImageView(QWidget* parent = nullptr);
    void setImage(ImagePNG* image);
    void clearScene(int width = 0, int height = 0);

private:
    ImageScene* scene = nullptr;
};


#endif // IMAGEVIEW_H
```

Файл *src/imageview.cpp*:

```
#include "imageview.h"
#include <QDebug>
#include <QMouseEvent>


ImageView::ImageView(QWidget *parent) : QGraphicsView(parent) {
    setMouseTracking(true);
    setObjectName("imageView");
    setAlignment(Qt::AlignLeft | Qt::AlignTop);

    scene = new ImageScene(0, 0, 0, 0, this);
    scene->setObjectName("imageScene");

    setScene(scene);
}

void ImageView::setImage(ImagePNG* image) {
    QPixmap imagePixmap = image->getPixmap();

    clearScene(image->getWidth(), image->getHeight());
    scene->addPixmap(imagePixmap);

    setScene(scene);
}

void ImageView::clearScene(int width, int height) {
    scene->clear();
    scene->setSceneRect(0, 0, width, height);
    setScene(scene);
}

ImageScene::ImageScene(qreal x, qreal y, qreal width, qreal height,
QObject *parent):
    QGraphicsScene(x, y, width, height, parent) {}

void ImageScene::mouseMoveEvent(QGraphicsSceneMouseEvent* event) {
    emit mousePositionChanged(event);
}

void ImageScene::mousePressEvent(QGraphicsSceneMouseEvent* event) {
    emit point1Selected(event);
}
```

```
void ImageScene::mouseReleaseEvent(QGraphicsSceneMouseEvent* event) {
    emit point2Selected(event);
}
```

Файл *src/imagetoolbar.h*:

```
#ifndef IMAGETOOLBAR_H
#define IMAGETOOLBAR_H

#include <QToolBar>
#include <QSpinBox>
#include <QToolButton>
#include "enumerations.h"


class ImageToolBar: public QToolBar {
    Q_OBJECT

private:
    QColor color1 = QColor(Qt::black);
    QColor color2 = QColor(Qt::white);
    ToolType selectedTool = ToolType::NONE_TOOL;
    ExchangingMethod selectedExchangingMethod =
ExchangingMethod::CLOCKWISE_METHOD;
    FillType selectedFillType = FillType::UNFILLED;

public:
    ImageToolBar(QWidget* parent = nullptr);

    QColor getFirstColor() const;
    QColor getSecondColor() const;
    ToolType getSelectedTool() const;
    ExchangingMethod getExchangingMethod() const;
    FillType getFillType() const;
    void setFirstColor(const QColor& color);
    void setSecondColor(const QColor& color);
    void setToolType(ToolType newToolType);
    void setExchangingMethod(ExchangingMethod newExchangingMethod);
    void setFillType(FillType newFillType);
    void resetTools();

    QToolButton* changeColor1_button = nullptr;
    QToolButton* changeColor2_button = nullptr;
    QToolButton* changeTool1_button = nullptr;
    QToolButton* changeTool2_button = nullptr;
    QToolButton* changeTool3_button = nullptr;
    QToolButton* changeTool4_button = nullptr;
    QToolButton* changeSquarePainting_button = nullptr;
    QToolButton* changeExchangeMethod_button = nullptr;
    QSpinBox* lineWidthSpinBox = nullptr;
    QSpinBox* squareSizeSpinBox = nullptr;
};

#endif // IMAGETOOLBAR_H
```

Файл *src/imagetoolbar.cpp*:

```
#include "imagetoolbar.h"
#include "utilityfunctions.h"
#include <QToolButton>
#include <QLabel>


ImageToolBar::ImageToolBar(QWidget *parent): QToolBar(parent) {
    setStyleSheet("QLabel {qproperty-alignment: AlignCenter;font-size:
8px;} QToolButton {padding: 0px; margin: 0px 1px 0px 1px; width: 40px;}
QSpinBox {height: 17px; width: 20px;}");
    setContextMenuPolicy(Qt::PreventContextMenu);
    setObjectName("toolbar");
    setMovable(false);
    setIconSize(QSize(40, 17));

    changeColor1_button = createToolButtonColor(this, "changeColor1",
"Color 1", QColor(Qt::black), false);
    changeColor2_button = createToolButtonColor(this, "changeColor2",
"Color 2", QColor(Qt::white), false);
    changeTool1_button = createToolButton(this, "changeTool1", "Drawing a
square", ":/icons/square_tool.png", true);
    changeTool2_button = createToolButton(this, "changeTool2", "Exchange
4 parts of the selected area", ":/icons/exchanging_tool.png", true);
    changeTool3_button = createToolButton(this, "changeTool3", "Replace a
common color", ":/icons/common_color_tool.png", false);
    changeTool4_button = createToolButton(this, "changeTool4", "Color
inversion in the selected area", ":/icons/invert_color.png", true);
    changeSquarePainting_button = createToolButton(this,
"changeSquarePainting", "Square filling", ":/icons/square_unpainted.png",
false);
    changeExchangeMethod_button = createToolButton(this,
"changeExchangeMethod", "Exchanging parts method",
":/icons/exchanging_1.png", false);

    lineWidthSpinBox = createSpinBox(this, 5, 0, 1000000, "Line width");
    squareSizeSpinBox = createSpinBox(this, 20, 0, 1000000, "Square
side");

    QLabel* label1 = new QLabel("COLORS:", this);
    QLabel* label2 = new QLabel("LINE WIDTH:", this);
    QLabel* label3 = new QLabel("SQUARE\nSIDE:", this);

    addWidget(label1);
    addWidget(changeColor1_button);
    addWidget(changeColor2_button);
    addSeparator();
    addWidget(label2);
    addWidget(lineWidthSpinBox);
    addSeparator();
    addWidget(label3);
    addWidget(squareSizeSpinBox);
    addSeparator();
    addWidget(changeTool1_button);
    addWidget(changeSquarePainting_button);
```

```
    addSeparator();
    addWidget(changeTool2_button);
    addWidget(changeExchangeMethod_button);
    addSeparator();
    addWidget(changeTool3_button);
    addSeparator();
    addWidget(changeTool4_button);
}

QColor ImageToolBar::getFirstColor() const {
    return color1;
}

QColor ImageToolBar::getSecondColor() const {
    return color2;
}

void ImageToolBar::setFirstColor(const QColor& color) {
    color1 = color;
}

void ImageToolBar::setSecondColor(const QColor& color) {
    color2 = color;
}

void ImageToolBar::resetTools() {
    selectedTool = ToolType::NONE_TOOL;

    changeTool1_button->setChecked(false);
    changeTool2_button->setChecked(false);
    changeTool4_button->setChecked(false);
}

ToolType ImageToolBar::getSelectedTool() const {
    return selectedTool;
}

ExchangingMethod ImageToolBar::getExchangingMethod() const {
    return selectedExchangingMethod;
}

FillType ImageToolBar::getFillType() const {
    return selectedFillType;
}

void ImageToolBar::setToolType(ToolType newToolType) {
    selectedTool = newToolType;
}

void ImageToolBar::setExchangingMethod(ExchangingMethod
newExchangingMethod) {
    selectedExchangingMethod = newExchangingMethod;
}

void ImageToolBar::setFillType(FillType newFillType) {
    selectedFillType = newFillType;
}
```

Файл *src/imagepng.h*:

```
#ifndef IMAGEPNG_H
#define IMAGEPNG_H

#include <png.h>
#include <QPixmap>
#include <QFileInfo>
#include "enumerations.h"


class ImagePNG {
private:
    struct ImagePart;

    int width = 0;
    int height = 0;
    png_byte colorType = 0;
    png_byte bitDepth = 0;
    png_byte interlaceType = 0;
    png_byte filteringMethod = 0;
    png_byte compressionType = 0;
    png_bytepp pixelArray = nullptr;

    QFileInfo fileInfo;
    bool loaded = false;
    bool changed = false;

    ImagePart* copyPart(const QPoint& point1, const QPoint& point2)
const;
    void pastePart(const ImagePart* part, const QPoint& point);

public:
    int load(const QString& imagePath, QWidget* parent = nullptr);
    int save(const QString& imagePath);
    int getWidth() const;
    int getHeight() const;
    png_byte getBitDepth() const;
    png_byte getColorType() const;
    QString getColorModelName() const;
    QFileInfo getFileInfo() const;
    QPixmap getPixmap() const;
    bool isLoaded() const;
    bool isChanged() const;
    bool isCoordinatesCorrect(const QPoint& point) const;
    void updateFileInfo(const QString& path = "");
    int getCorrectY(int y) const;
    int getCorrectX(int x) const;
    QPoint& correctPoint(QPoint& point) const;
    QPoint correctPoint(const QPoint& point) const;

    void paintSquare(const QPoint& point, int lineWidth, int squareSide,
const QColor& color1, FillType fillType, const QColor& color2);
    void changeImageAreas(const QPoint& point1, const QPoint& point2,
ExchangingMethod exchangingMethod);
```

```
    void replaceCommonColor(const QColor& newColor);
    void invertColors(const QPoint& point1, const QPoint& point2);

    ~ImagePNG();
};


struct ImagePNG::ImagePart {
    int width = 0;
    int height = 0;
    png_bytep* pixelArray = nullptr;

    ImagePart(int width, int height);
    ~ImagePart();
};


#endif // IMAGEPNG_H
```

Файл *src/imagepng.cpp*:

```
#include "imagepng.h"
#include "utilityfunctions.h"
#include <cmath>
#include <QMessageBox>
#include <QDebug>


int ImagePNG::load(const QString& imagePath, QWidget* parent) {
    FILE* file = fopen(imagePath.toStdString().c_str(), "rb");

    if (!file) {
        qDebug("Error in ImagePNG::load with fopen()");
        return 1;
    }

    unsigned char header[8];
    fread(header, sizeof(unsigned char), 8, file);

    if (png_sig_cmp(header, 0, 8)){
        qDebug("Error in ImagePNG::load with png_sig_cmp() (Non-PNG
header)");
        fclose(file);
        return 2;
    }

    png_structp pngStructPtr =
png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);

    if (!pngStructPtr) {
        qDebug("Error in ImagePNG::load with png_create_read_struct()");
        fclose(file);
        return 3;
    }

    png_infop pngInfoStructPtr = png_create_info_struct(pngStructPtr);
```

```
    if (!pngInfoStructPtr) {
        qDebug("Error in ImagePNG::load with png_create_info_struct()");
        png_destroy_read_struct(&pngStructPtr, NULL, NULL);
        fclose(file);
        return 4;
    }

    if (setjmp(png_jmpbuf(pngStructPtr))) {
        qDebug("Error in ImagePNG::load with setjmp(png_jmpbuf())");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 5;
    }

    png_init_io(pngStructPtr, file);
    png_set_sig_bytes(pngStructPtr, 8);
    png_read_info(pngStructPtr, pngInfoStructPtr);

    width = png_get_image_width(pngStructPtr, pngInfoStructPtr);
    height = png_get_image_height(pngStructPtr, pngInfoStructPtr);
    colorType = png_get_color_type(pngStructPtr, pngInfoStructPtr);
    bitDepth = png_get_bit_depth(pngStructPtr, pngInfoStructPtr);
    filteringMethod = png_get_filter_type(pngStructPtr,
pngInfoStructPtr);
    compressionType = png_get_compression_type(pngStructPtr,
pngInfoStructPtr);
    interlaceType = png_get_interlace_type(pngStructPtr,
pngInfoStructPtr);

    png_set_interlace_handling(pngStructPtr);

    if (compressionType != 0 || filteringMethod != 0) {
        qDebug("This image format is not supported");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 9;
    }

    qDebug("Color type: %d, bit depth: %d, interlace type: %d",
colorType, bitDepth, interlaceType);

    if (bitDepth != 8 || colorType != PNG_COLOR_TYPE_RGBA) {
        QMessageBox::StandardButton button =
QMessageBox::question(parent, "Converting an image", "The selected PNG
image doesn't match the RGBA 8-bit format. Do you want to convert this
image to this format?");

        if (button == QMessageBox::StandardButton::No) {
            qDebug("This image format is not supported");
            png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr,
NULL);
            fclose(file);
            return 8;
        }
    }
```

```
    if (bitDepth == 16) {
        qDebug("Converting bit depth: 16 to 8");
        png_set_strip_16(pngStructPtr);
        bitDepth = 8;
    }

    if (bitDepth < 8) {
        qDebug("Converting bit depth: 1, 2 or 4 to 8");

        if (colorType == PNG_COLOR_TYPE_GRAY) {
            png_set_expand_gray_1_2_4_to_8(pngStructPtr);
        } else {
            png_set_packing(pngStructPtr);
        }

        bitDepth = 8;
    }

    if (colorType == PNG_COLOR_TYPE_PALETTE) {
        qDebug("Converting color type: COLOR_TYPE_PALETTE to
COLOR_TYPE_RGB");
        png_set_palette_to_rgb(pngStructPtr);
        colorType = PNG_COLOR_TYPE_RGB;
    }

    if (colorType == PNG_COLOR_TYPE_GRAY || colorType ==
PNG_COLOR_TYPE_GRAY_ALPHA) {
        qDebug("Converting color type: PNG_COLOR_TYPE_GRAY to
COLOR_TYPE_RGB or PNG_COLOR_TYPE_GRAY_ALPHA to COLOR_TYPE_RGBA");
        png_set_gray_to_rgb(pngStructPtr);

        if (colorType == PNG_COLOR_TYPE_GRAY) {
            colorType = PNG_COLOR_TYPE_RGB;
        } else {
            colorType = PNG_COLOR_TYPE_RGBA;
        }
    }

    if (colorType == PNG_COLOR_TYPE_RGB) {
        qDebug("Converting color type: COLOR_TYPE_RGB to COLOR_TYPE_RGBA,
fill alpha channel");
        png_set_add_alpha(pngStructPtr, 0xFF, PNG_FILLER_AFTER);
        colorType = PNG_COLOR_TYPE_RGBA;
    }

    if (png_get_valid(pngStructPtr, pngInfoStructPtr, PNG_INFO_tRNS)) {
        qDebug("Converting color type to alpha");
        png_set_tRNS_to_alpha(pngStructPtr);
        colorType = PNG_COLOR_TYPE_RGBA;
    }

    png_read_update_info(pngStructPtr, pngInfoStructPtr);

    colorType = png_get_color_type(pngStructPtr, pngInfoStructPtr);
    bitDepth = png_get_bit_depth(pngStructPtr, pngInfoStructPtr);
```

```
    qDebug("Converted color type: %d, bit depth: %d", colorType,
bitDepth);

    if (colorType != PNG_COLOR_TYPE_RGBA || bitDepth != 8) {
        qDebug("Error with converting image to RGBA 8 bit");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 6;
    }

    if (setjmp(png_jmpbuf(pngStructPtr))) {
        qDebug("Error in ImagePNG::load with setjmp(png_jmpbuf()) #2");
        png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
        fclose(file);
        return 7;
    }

    pixelArray = new png_bytep[height];

    for (int y = 0; y < height; y++) {
        pixelArray[y] = new png_byte[png_get_rowbytes(pngStructPtr,
pngInfoStructPtr)];
    }

    png_read_image(pngStructPtr, pixelArray);
    qDebug("The image is successfully loaded: %s",
imagePath.toStdString().c_str());

    png_destroy_read_struct(&pngStructPtr, &pngInfoStructPtr, NULL);
    fclose(file);

    loaded = true;
    fileInfo.setFile(imagePath);

    return 0;
}

int ImagePNG::save(const QString& imagePath) {
    FILE* file = fopen(imagePath.toStdString().c_str(), "wb");

    if (!file) {
        qDebug("Error in ImagePNG::save with fopen()");
        return 1;
    }

    png_structp pngStructPtr =
png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (!pngStructPtr) {
        qDebug("Error in ImagePNG::save with png_create_write_struct()");
        fclose(file);
        return 2;
    }

    png_infop pngInfoStructPtr = png_create_info_struct(pngStructPtr);
    if (!pngInfoStructPtr) {
        qDebug("Error in ImagePNG::save with png_create_info_struct()");
        png_destroy_write_struct(&pngStructPtr, NULL);
```

```
        fclose(file);
        return 3;
    }

    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf())");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 4;
    }

    png_init_io(pngStructPtr, file);

    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #2");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 5;
    }

    png_set_IHDR(pngStructPtr, pngInfoStructPtr, width, height, bitDepth,
colorType, interlaceType, compressionType, filteringMethod);
    png_write_info(pngStructPtr, pngInfoStructPtr);

    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #3");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 6;
    }

    png_write_image(pngStructPtr, pixelArray);

    if (setjmp(png_jmpbuf(pngStructPtr))){
        qDebug("Error in ImagePNG::save with setjmp(png_jmpbuf()) #4");
        png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
        fclose(file);
        return 7;
    }

    png_write_end(pngStructPtr, NULL);

    qDebug("The image is successfully saved: %s",
imagePath.toStdString().c_str());

    png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
    fclose(file);

    changed = false;

    return 0;
}

ImagePNG::ImagePart* ImagePNG::copyPart(const QPoint& point1, const
QPoint& point2) const {
    if (isLoaded() && isCoordinatesCorrect(point1) &&
isCoordinatesCorrect(point2)) {
```

```
        ImagePart* part = new ImagePart(point2.x() - point1.x() + 1,
point2.y() - point1.y() + 1);

        for (int y = 0; y < part->height; y++) {
            for (int x = 0; x < part->width; x++) {
                setPixel(part->pixelArray[y] + 4 * x,
getPixel(pixelArray[point1.y() + y] + 4 * (point1.x() + x)));
            }
        }

        qDebug("Part of the image in area %s to %s was successfully
copied", pointToString(point1).toStdString().c_str(),
pointToString(point2).toStdString().c_str());

        return part;
    }

    return nullptr;
}

void ImagePNG::pastePart(const ImagePNG::ImagePart* part, const QPoint&
point) {
    if (isLoaded() && part != nullptr && isCoordinatesCorrect(point)) {
        int endY = getCorrectY(point.y() + part->height);
        int endX = getCorrectX(point.x() + part->width);

        for (int y = point.y(); y < endY; y++) {
            for (int x = point.x(); x < endX; x++) {
                setPixel(pixelArray[y] + 4 * x, getPixel(part-
>pixelArray[y - point.y()] + 4 * (x - point.x())));
            }
        }

        qDebug("Part of the image was successfully pasted in %s",
pointToString(point).toStdString().c_str());
    }
}

int ImagePNG::getWidth() const {
    return width;
}

int ImagePNG::getHeight() const {
    return height;
}

png_byte ImagePNG::getBitDepth() const {
    return bitDepth;
}

png_byte ImagePNG::getColorType() const {
    return colorType;
}

QFileInfo ImagePNG::getFileInfo() const {
    return fileInfo;
}
```

```
QString ImagePNG::getColorModelName() const {
    switch (colorType) {
    case PNG_COLOR_TYPE_RGB:
        return "RGB";
    case PNG_COLOR_TYPE_GRAY:
        return "Gray";
    case PNG_COLOR_TYPE_RGBA:
        return "RGBA";
    case PNG_COLOR_TYPE_PALETTE:
        return "Palette";
    case PNG_COLOR_TYPE_GRAY_ALPHA:
        return "Gray alpha";
    default:
        return "Unknown";
    }
}

QPixmap ImagePNG::getPixmap() const {
    QPixmap pixmap;

    if (isLoaded())  {
        QImage image(width, height, QImage::Format_RGBA64);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                png_bytep pixel = pixelArray[y] + 4 * x;
                image.setPixelColor(x, y, QColor(pixel[0], pixel[1],
pixel[2], pixel[3]));
            }
        }

        pixmap = QPixmap::fromImage(image);
    }

    return pixmap;
}

bool ImagePNG::isLoaded() const {
    return loaded && pixelArray != nullptr;
}

bool ImagePNG::isChanged() const {
    return changed;
}

bool ImagePNG::isCoordinatesCorrect(const QPoint& point) const {
    return (point.x() >= 0 && point.y() >= 0 && point.x() < width &&
point.y() < height);
}

void ImagePNG::updateFileInfo(const QString& path) {
    if (path == "") {
        fileInfo = QFileInfo(fileInfo.filePath());
    } else {
        fileInfo = QFileInfo(path);
    }
```

```
}

int ImagePNG::getCorrectY(int y) const {
    if (y < 0) {
        return 0;
    }

    if (y >= height) {
        return height - 1;
    }

    return y;
}

int ImagePNG::getCorrectX(int x) const {
    if (x < 0) {
        return 0;
    }

    if (x >= width) {
        return width - 1;
    }

    return x;
}

QPoint& ImagePNG::correctPoint(QPoint& point) const {
    if (!isCoordinatesCorrect(point)) {
        point.setX(getCorrectX(point.x()));
        point.setY(getCorrectY(point.y()));
    }

    return point;
}

QPoint ImagePNG::correctPoint(const QPoint& point) const {
    return QPoint(getCorrectX(point.x()), getCorrectY(point.y()));
}

void ImagePNG::paintSquare(const QPoint& point, int lineWidth, int
squareSide, const QColor& color1, FillType fillType, const QColor&
color2) {
    if (!isLoaded()) {
        return;
    }

    qDebug("Starting drawing a square with %d line width and %d square
side:", lineWidth, squareSide);
    qDebug("Starting point: %s;",
pointToString(point).toStdString().c_str());
    qDebug("Filling type: %d;", fillType);
    qDebug("Colors: %s and %s.",
colorToString(color1).toStdString().c_str(),
colorToString(color2).toStdString().c_str());

    int xRight = getCorrectX(point.x() + squareSide - 1);
    int xInsideRight = point.x() + squareSide - lineWidth - 1;
```

```
    int xInsideLeft = point.x() + lineWidth;
    int yDown = getCorrectY(point.y() + squareSide - 1);
    int yIndiseUp = point.y() + lineWidth;
    int yIndiseDown = point.y() + squareSide - lineWidth - 1;


    for (int y = point.y(); y <= yDown; y++) {
        for (int x = point.x(); x <= xRight; x++) {
            if (isCoordinatesCorrect(QPoint(x, y))) {
                if (x < xInsideLeft || x > xInsideRight || y < yIndiseUp
|| y > yIndiseDown) {
                    setPixel(pixelArray[y] + 4 * x, color1);
                } else if (fillType == FillType::FILLED) {
                    setPixel(pixelArray[y] + 4 * x, color2);
                }
            }
        }
    }

    qDebug("Drawing a square finished");

    changed = true;
}

void ImagePNG::changeImageAreas(const QPoint& point1, const QPoint&
point2, ExchangingMethod exchangingMethod) {
    if (!isLoaded()) {
        return;
    }

    QPoint fixedPoint1 = correctPoint(point1);
    QPoint fixedPoint2 = correctPoint(point2);

    if ((fixedPoint2.x() - fixedPoint1.x()) % 2 == 0) {
        fixedPoint2.setX(point2.x() - 1);
    }

    if ((fixedPoint2.y() - fixedPoint1.y()) % 2 == 0) {
        fixedPoint2.setY(point2.y() - 1);
    }

    int deltaX = fixedPoint2.x() - fixedPoint1.x();
    int deltaY = fixedPoint2.y() - fixedPoint1.y();
    int partWidth = deltaX / 2 + 1;
    int partHeight = deltaY / 2 + 1;
    int middleX = fixedPoint1.x() + partWidth;
    int middleY = fixedPoint1.y() + partHeight;

    if (partWidth < 2 || partHeight < 2) {
        qDebug("Area is too small!");
        return;
    }

    QPoint partPoint1 = fixedPoint1;
    QPoint partPoint2 = QPoint(middleX, fixedPoint1.y());
    QPoint partPoint3 = QPoint(fixedPoint1.x(), middleY);
    QPoint partPoint4 = QPoint(middleX, middleY);
```

```
    ImagePart* part1 = copyPart(partPoint1, QPoint(middleX - 1, middleY -
1));
    ImagePart* part2 = copyPart(partPoint2, QPoint(fixedPoint2.x(),
middleY - 1));
    ImagePart* part3 = copyPart(partPoint3, QPoint(middleX - 1,
fixedPoint2.y()));
    ImagePart* part4 = copyPart(partPoint4, fixedPoint2);

    if (exchangingMethod == ExchangingMethod::CLOCKWISE_METHOD) {
        pastePart(part1, partPoint2);
        pastePart(part2, partPoint4);
        pastePart(part3, partPoint1);
        pastePart(part4, partPoint3);
    } else if (exchangingMethod ==
ExchangingMethod::ANTICLOCKWISE_METHOD) {
        pastePart(part1, partPoint3);
        pastePart(part2, partPoint1);
        pastePart(part3, partPoint4);
        pastePart(part4, partPoint2);
    } else {
        pastePart(part1, partPoint4);
        pastePart(part2, partPoint3);
        pastePart(part3, partPoint2);
        pastePart(part4, partPoint1);
    }

    delete part1;
    delete part2;
    delete part3;
    delete part4;

    changed = true;
}

// Needs for QMap working with QColor
bool operator<(const QColor& a, const QColor& b) {
   return a.redF() < b.redF() || a.greenF() < b.greenF() || a.blueF() <
b.blueF() || a.alphaF() < b.alphaF();
}

void ImagePNG::replaceCommonColor(const QColor& newColor) {
    if (!isLoaded()) {
        return;
    }

    QMap<QColor, int> dictionary;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            QColor color = getColorFromPixel(pixelArray[y] + 4 * x);

            if (dictionary.contains(color)) {
                dictionary[color] += 1;
            } else {
                dictionary[color] = 1;
            }
```

```
        }
    }

    QColor commonColor = dictionary.begin().key();
    int max = dictionary.begin().value();

    for (auto pair : dictionary.toStdMap()) {
        if (pair.second > max) {
            max = pair.second;
            commonColor = pair.first;
        }
    }

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            png_bytep pixel = pixelArray[y] + 4 * x;
            QColor color = getColorFromPixel(pixel);

            if (color == commonColor) {
                setPixel(pixel, newColor);
            }
        }
    }

    changed = true;
}

void ImagePNG::invertColors(const QPoint& point1, const QPoint& point2) {
    if (!isLoaded()) {
        return;
    }

    QPoint fixedPoint1 = correctPoint(point1);
    QPoint fixedPoint2 = correctPoint(point2);

    for (int y = fixedPoint1.y(); y < fixedPoint2.y(); y++) {
        for (int x = fixedPoint1.x(); x < fixedPoint2.x(); x++) {
            invertPixelColor(pixelArray[y] + 4 * x);
        }
    }

    changed = true;
}

ImagePNG::~ImagePNG() {
    if (pixelArray != nullptr) {
        for(int y = 0; y < height; y++) {
            delete[] pixelArray[y];
        }

        delete[] pixelArray;
    }
}

ImagePNG::ImagePart::ImagePart(int width, int height) {
    this->width = width;
    this->height = height;
```

```
    pixelArray = new png_bytep[height];

    for (int y = 0; y < height; y++) {
        pixelArray[y] = new png_byte[4 * width];
    }
}

ImagePNG::ImagePart::~ImagePart() {
    if (pixelArray != nullptr) {
        for (int y = 0; y < height; y++) {
            delete[] pixelArray[y];
        }

        delete[] pixelArray;
    }
}
```

Файл *src/helpbrowser.h*:

```
#ifndef HELPBROWSER_H
#define HELPBROWSER_H

#include <QTextBrowser>
#include <QHelpEngine>
#include <QDockWidget>


class HelpBrowser: public QTextBrowser {
    Q_OBJECT

public:
    HelpBrowser(const QString& collectionPath, QWidget* parent =
nullptr);

    QVariant loadResource(int type, const QUrl &name);
    QHelpContentWidget* getContentWidget();
    QHelpIndexWidget* getIndexWidget();

    ~HelpBrowser();

private:
    QHelpEngine* helpEngine = nullptr;
};


class HelpDockWidget: public QDockWidget {
    Q_OBJECT

public:
    HelpDockWidget(QWidget* parent = nullptr);

private:
    HelpBrowser* helpBrowser = nullptr;
};
```

```
#endif // HELPBROWSER_H
```

Файл *src/helpbrowser.cpp*:

```
#include "helpbrowser.h"
#include <QTabWidget>
#include <QSplitter>
#include <QHelpContentWidget>
#include <QHelpIndexWidget>


HelpBrowser::HelpBrowser(const QString& collectionPath, QWidget* parent):
QTextBrowser(parent) {
    helpEngine = new QHelpEngine(collectionPath);
    helpEngine->setupData();
}

QVariant HelpBrowser::loadResource(int type, const QUrl &name) {
    if (name.scheme() == "qthelp")
        return QVariant(helpEngine->fileData(name));
    else
        return QTextBrowser::loadResource(type, name);
}

QHelpContentWidget* HelpBrowser::getContentWidget() {
    return helpEngine->contentWidget();
}

QHelpIndexWidget *HelpBrowser::getIndexWidget() {
    return helpEngine->indexWidget();
}

HelpBrowser::~HelpBrowser() {
    delete helpEngine;
}

HelpDockWidget::HelpDockWidget(QWidget* parent): QDockWidget(parent) {
    QSplitter* horizontalSplitter = new QSplitter(Qt::Horizontal, this);
    QTabWidget* tabWidget = new QTabWidget(horizontalSplitter);
    HelpBrowser* helpBrowser = new HelpBrowser("docs/help.qhc",
horizontalSplitter);

    tabWidget->setMaximumWidth(320);
    tabWidget->addTab(helpBrowser->getContentWidget(), "Contents");
    tabWidget->addTab(helpBrowser->getIndexWidget(), "Index");
    horizontalSplitter->insertWidget(0, tabWidget);
    horizontalSplitter->insertWidget(1, helpBrowser);
    horizontalSplitter->hide();

    helpBrowser-
>setSource(QUrl("qthelp://coursework.help/docs/index.html"));

    setWidget(horizontalSplitter);

    connect(helpBrowser->getContentWidget(), SIGNAL(linkActivated(const
QUrl&)), helpBrowser, SLOT(setSource(const QUrl&)));
```

```
    connect(helpBrowser->getIndexWidget(), SIGNAL(linkActivated(const
QUrl&, const QString&)), helpBrowser, SLOT(setSource(const QUrl&)));

    hide();
}
```

Файл *src/enumerations.h*:

```
#ifndef ENUMERATIONS_H
#define ENUMERATIONS_H


enum ToolType {
    NONE_TOOL = 0,
    SQUARE_PAINT_TOOL = 1,
    EXCHANGING_PARTS_TOOL = 2,
    CHANGE_COMMON_COLOR_TOOL = 3,
    INVERT_COLOR_TOOL = 4
};


enum ExchangingMethod {
    CLOCKWISE_METHOD = 1,
    ANTICLOCKWISE_METHOD = 2,
    DIAGONAL_METHOD = 3
};


enum FillType {
    UNFILLED = 1,
    FILLED = 2
};


#endif // ENUMERATIONS_H
```

КОММЕНТАРИИ ИЗ ПУЛЛ-РЕКВЕСТОВ

Исходный код:

```
HelpBrowser* helpBrowser = new HelpBrowser("docs/help.qhc",
horizontalSplitter);
    QTabWidget* tabWidget = new QTabWidget(horizontalSplitter);
    QSplitter* horizontalSplitter = new QSplitter(Qt::Horizontal,
this);
HelpDockWidget::HelpDockWidget(QWidget* parent): QDockWidget(parent) {

}
    delete helpEngine;
HelpBrowser::~HelpBrowser() {

}
    return helpEngine->indexWidget();
QHelpIndexWidget *HelpBrowser::getIndexWidget() {

}
    return helpEngine->contentWidget();
QHelpContentWidget* HelpBrowser::getContentWidget() {

}
        return QTextBrowser::loadResource(type, name);
    else
        return QVariant(helpEngine->fileData(name));
    if (name.scheme() == "qthelp")
QVariant HelpBrowser::loadResource(int type, const QUrl &name) {

}
    helpEngine->setupData();
    helpEngine = new QHelpEngine(collectionPath);
HelpBrowser::HelpBrowser(const QString& collectionPath, QWidget*
parent): QTextBrowser(parent) {


#include <QHelpIndexWidget>
#include <QHelpContentWidget>
#include <QSplitter>
#include <QTabWidget>
#include "helpbrowser.h"
```

Комментарии:

TatyanaBerlenko: вынесите все пути/url в строковые именованные константы

TheLifes08: Готово

Изменения: