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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения в формате PNG

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Азаров М.С.

Группа 0382

Тема работы : Обработка изображения в формате PNG

Вариант 17

Программа должна реализовывать следующий функционал по обработке PNG-файла

1. Рисование окружности. Окружность определяется:
   - либо координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, либо координатами ее центра и радиусом
   - толщиной линии окружности
   - цветом линии окружности
   - окружность может быть залитой или нет
   - цветом которым залита сама окружность, если пользователем выбрана залитая окружность

2. Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
   - Какую компоненту требуется изменить
   - В какой значение ее требуется изменить

3. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:
   - Количество частей по “оси” Y
   - Количество частей по “оси” X
   - Толщина линии
   - Цвет линии
   - Либо путь куда сохранить кусочки

4. Рисование квадрата с диагоналями. Квадрат определяется:
   - Координатами левого верхнего угла
   - Размером стороны

- Толщиной линий
- Цветом линий
- Может быть залит или нет (диагонали располагаются “поверх” заливки)
- Цветом которым он залит, если пользователем выбран залитый

Дата выдачи задания: 25.04.2021

Дата сдачи реферата: 30.05.2021

Дата защиты реферата: 01.06.2021

Студент _____ Азаров М.С.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

Курсовая работа заключается в выполнении поставленного задания и отработки полученный знаний. В процессе выполнения работы , были использованы следующие инструменты : для обработки изображения формата PNG типа цвета RGBA была использована библиотека libpng; для создания графического интерфейса был использован фреймворк Qt.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсовой работы заключается в создание программы , для обработки изображения формата PNG с типом цвета PNG. В ходе работы были задействованы язык C++, библиотека libpng, среда разработки Qt.

1.ЦЕЛЬ

Освоить работу с библиотекой libpng. Научится применять язык программирования C++ и его ООП парадигму для написания программ. Также освоить создание графической оболочки программы в среде разработки Qt.

2.ХОД РАБОТЫ

2.1. Чтение

Для работы с изображением был написан класс Image_PNG в котором было реализован с помощью методов класса API для работы с изображением . Для чтения используется метод ***readFromDisk()*** который принимает путь к файлу считывает изображения с помощью средств библиотеки libpng , и возвращает в случае удачи считывания 1 , иначе 0.

2.2. Запись

Для записи используется метод ***writeOnDisk()*** которая , если не передать путь , сохраняет изображение в тот же файл с которого считало, если передать путь записывает изображение на диск по этому пути. Функция также использует функции libpng для записи .

2.2.Решение подзадачи 1

Данная подзадача заключалась в рисовании окружности на выбранном изображении по заданным пользователем параметрам. Для реализации жтой задачи был написан метод drawCircle(), который принимает координаты центра окружности , радиус , толщину линии и цвета линии и заливки . Причем если цвет заливки не isValid() то окружность заливаться не будет.

2.2.Решение подзадачи 2

Во второй подзадаче требовалось сделать rgba-фильтр. То есть для всего изображения в заданный канал цвета установить заданное значение . Это реализуется с помощью метода **filterRGBA** () , в который передается значение

в которое нужно установить в канал для всего изображения и сам канал ч с помощью перечисления enum **RGBA**.

2.3.Решение подзадачи 3

В этой подзадаче нужно было разрезать изображение N*M частей , где N и M вводит пользователь , и сохранить все кусочки в указаную папку пользователем. Реализация исполнена с помощью функции **cutIntoPieces(),** которая в свою очередь использует функцию **getPieceImg(),** которая возвращает один кусочек изображения по введенным координатам.

2.3.Решение подзадачи 4

В этой задаче требовалось нарисовать прямоугольник с диагоналями по введённым координатам и с выбранным цветом , а также с возможностью выбора с заливкой или без нее. Реализация этой задачи находится в функции **drawRectWithDiag()**, которая использует функцию **drawWithDiag()** для рисования прямоугольника без диагоналей и функцию **drawLine()** ,для рисования диагоналей. Причем сама функция **drawWithDiag()** тоже использует функцию рисования линий **drawLine()** для рисования сторон прямоугольника

ЗАКЛЮЧЕНИЕ

В ходе работы была создана программа удовлетворяющая поставленным требованиям. А именно программа считывает , обрабатывает и записывает изображение формата PNG типа цвета RGBA. Как и задумывалось у неё присутствует графический интерфейс и реализована работа поставленных подзадач на обработку изображения.

Также были приобретены новые знания в написании программ на языке C++, работы со сторонними библиотеками , такими как libpng и отработаны уже имеющиеся знания , полученные в течение семестра.

КОД ПРОГРАММЫ

Файл **Image_PNG.h**:

```
/* заголовочный файл класса Image_PNG.h */
/* интерфейс класса*/
/* работает только с RGBA*/

#ifndef IMAGE_PNG_H
#define IMAGE_PNG_H

#include <png.h>
#include <QString>
#include <QVector>
#include <QColor>

const int SIZE_BIT_CHECK_PNG = 8;

enum RGBA{//канал
   R, //=0
   G, //=1
   B, //=2
   A  //=3
};



/* объявление класса*/
class Image_PNG {
private:
   int m_width, m_height;//ширина и высота в изб. в пикселях
   png_byte **m_arr_pixel = NULL;//масив пикселей  изображение (трех мерный массив)
   png_byte m_color_type; //тип кодирования цвета (может быть только RGB)
```

```
    png_byte m_bit_depth;  //бит глубины
```

QString m_path_to_img = "";// путь к изоб. (включая имя файла) // "" - означает что структура не инициализированна.

```
    png_structp m_png_ptr;
    png_infop m_info_ptr;
    int m_number_of_passes;

    void set_first_small_second_big(int *small_p, int *big_p);//в первую переменую устанавливает меньшее,
                                                              //во второую большее значение из переданых переменых
    void freeImage();   //вспомогательная фнкция для деструктора
    void deepCopy(const Image_PNG &img);      //вспомогательная фнкция для конструктора копирования и operator=
    void setColorInPixel(int x, int y,QColor color);  //если кординаты вне картинки ничего не делает
public:
    //Конструктор
    Image_PNG();

    ~Image_PNG();//деконструктор

    Image_PNG(const Image_PNG &img);//конструктор копирования

    Image_PNG& operator=(const Image_PNG &img);//и так понятно

    int readFromDisk(QString path); //считывание изб. с диска

    void writeOnDisk(QString path ); // записываем изб. на диск
```

void writeOnDisk();//записываем изб. на диск по умл. от куда и было считано

void writeTempImgOnDisk(QString temp_path);//записывает на диск временное изображение

//отличается от writeOnDisk() тем что не изменяет m_path_to_img, то есть не превязывает екземпляр к новому файлу

void filterRGBA (int value,RGBA channel);//устанавливает згачение value в канале channal во всех пикселях изб.

bool wasInitialized(); //определяет была ли инициализирована структура

int getHeight();//получить высоту изб. в пикс.

int getWidth();//получить ширину в пикс.

QVector<QVector<Image_PNG>> cutIntoPieces(int N, int M);// разрезать изображение на N*M частей

Image_PNG getPieceImg(int x, int y, int width, int height); //получить один кусочек изображения

//x и y координаты верхнего левого кусочка

//рисует окружность круг(включяая r , не включая r+d)// если color_in isValid() == true заливает внут. окружности

void drawCircle(int centr_x, int centr_y, int r, int d, QColor color, QColor color_in = QColor::Invalid);

//нарисовать отрезок

void drawLine(int x1, int y1, int x2, int y2,int d , QColor color);

```
    //нарисовать прямоугольник //если color_in isValid то ресуется залитый
        void drawRectangle(int x, int y,int width,int height,int d ,QColor color_out, QColor color_in = QColor::Invalid);

    //нарисовать примоугольник с диагоналями
      void drawRectWithDiag(int x, int y,int width,int height,int d ,QColor color_out, QColor color_in = QColor::Invalid);

};

#endif // IMAGE_PNG_H
```

Файл **Image_PNG.cpp**:

```
#include "Image_PNG.h"
#include <QMessageBox>
#include <QFile>
#include <QtMath>


void Image_PNG::writeTempImgOnDisk(QString temp_path){
    std::string tmp_str =   temp_path.toStdString();
    const char *c_path = tmp_str.c_str();

    /* Открываем файл для бин. чтения*/
    FILE *fp = fopen(c_path, "wb");
    if (fp == NULL){
        // Some error handling: file could not be opened
        QMessageBox::critical(nullptr,"Ошибка","Файл с таким именем не найден");
```

```
    return ;
  }

  /* Выделение дин. памяти */
    m_png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);

  if (m_png_ptr == NULL){
    fclose(fp);
    // Some error handling: png_create_write_struct failed

     QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин. память для структуры png_ptr");
    return ;
  }

  m_info_ptr = png_create_info_struct(m_png_ptr);
  if (m_info_ptr == NULL){
    png_destroy_write_struct(&m_png_ptr, NULL);
    fclose(fp);
    // Some error handling: png_create_info_struct failed

     QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин. память для структуры info_ptr");
    return ;
  }

  if (setjmp(png_jmpbuf(m_png_ptr))){
    png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
    fclose(fp);
    // Some error handling: error during init_io
```

```
        QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при записи информации о файле");
        return ;
    }

    png_init_io(m_png_ptr, fp);


    /* write header */
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during writing header
    }

    png_set_IHDR(m_png_ptr, m_info_ptr, m_width, m_height,
                 m_bit_depth, m_color_type, PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(m_png_ptr, m_info_ptr);


    /* write bytes */
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during writing bytes

        QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при записи изображения");
```

```
        return ;
    }

    png_write_image(m_png_ptr, m_arr_pixel);


    /* end write */
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during end of write

            QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при записи конца файла");
        return ;
    }

    png_write_end(m_png_ptr, NULL);

    png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
    fclose(fp);
}


bool Image_PNG::wasInitialized(){
    if (m_path_to_img == ""){
        return false;
    }else {
        return true;
    }
}
```

```
void  Image_PNG::freeImage(){
   if (m_arr_pixel != NULL){
      for (int i = 0; i < m_height; i++)
         free(m_arr_pixel[i]);
      free(m_arr_pixel);
   }
   m_arr_pixel = NULL;
}


//Деструктор
Image_PNG::~Image_PNG() {
   freeImage();
}

//Конструктор
Image_PNG::Image_PNG() {    }


//вспомогательная фнкция для конструктора копирования и operator=
void Image_PNG::deepCopy(const Image_PNG &img){

   if (m_arr_pixel != NULL){
      freeImage();
   }

   if (img.m_arr_pixel == NULL){
      m_arr_pixel = NULL;
      return;
```

```
    } else {
        m_width = img.m_width;
        m_height = img.m_height;
        m_color_type = img.m_color_type;
        m_bit_depth = img.m_bit_depth;
        m_path_to_img = img.m_path_to_img;

        m_arr_pixel = (png_bytep *) malloc(sizeof(png_bytep) * m_height);
        for (int i = 0; i < m_height; i++)
            m_arr_pixel[i] = (png_byte *) malloc(sizeof(png_byte) * m_width * 4);// так как канолов 4


        for (int y = 0; y < m_height; y++) {
            for (int x = 0; x < (m_width*4); x++) {//4 - количество каналов (в RGBA их 4)
                m_arr_pixel[y][x] = img.m_arr_pixel[y][x];
            }
        }
    }

}


//Конструктор копирования
Image_PNG::Image_PNG(const Image_PNG &img) {
    deepCopy(img);
}


Image_PNG& Image_PNG::operator=(const Image_PNG &img){
```

```
    if (this != &img){//проверка на самоприсвоение(тк мы очищаем память в deepCopy)
        deepCopy(img);
    }

    return *this;
}


//запись на диск по умл.(в файл с которого было считано)
void Image_PNG::writeOnDisk() {
    Image_PNG::writeOnDisk(m_path_to_img);
}


//чтение изображения с диска
int Image_PNG::readFromDisk( QString path){
    char header[SIZE_BIT_CHECK_PNG];     // 8 is the maximum size that can be checked

    std::string tmp_str =   path.toStdString();
    const char *c_path = tmp_str.c_str();

    /* open file and test for it being a png */
    FILE *fp = fopen(c_path, "rb");
    if (fp == NULL){
        // Some error handling: file could not be opened
        QMessageBox::critical(nullptr,"Ошибка","Файл с таким именем не найден");
        return 1;
    }
```

```
        fread(header,sizeof (header[0]), SIZE_BIT_CHECK_PNG, fp);//считывает массив эл из файла
                                if (png_sig_cmp((png_const_bytep)header, 0, SIZE_BIT_CHECK_PNG)){//проверка что файл это png
        fclose(fp);
        // Some error handling: file is not recognized as a PNG

            QMessageBox::critical(nullptr,"Ошибка","Этот файл не PNG\n\nДанная прогрмма потдерживает только файлы формата *.png, с типом цвета RGBA");
        return 1;
    }


    /* выделение памяти для структур */
        m_png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (m_png_ptr == NULL){
        fclose(fp);
        // Some error handling: png_create_read_struct failed
         QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин. память для структуры png_ptr");
        return 1;
    }

    m_info_ptr = png_create_info_struct(m_png_ptr);
    if (m_info_ptr == NULL){
        png_destroy_read_struct(&m_png_ptr,NULL, NULL);//очистка массива
        fclose(fp);
        // Some error handling: png_create_info_struct failed
```

```
        QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин. память для структуры info_ptr");
        return 1;
    }

    if (setjmp(png_jmpbuf(m_png_ptr))){//обработка ошибок
        png_destroy_read_struct(&m_png_ptr,&m_info_ptr, NULL);//очистка массива
        fclose(fp);
        // Some error handling: error during init_io

            QMessageBox::critical(nullptr,"Ошибка","Не удалось считать данные об изображении");
        return 1;
    }

    png_init_io(m_png_ptr, fp); //настройка функция ввода/вывода
     png_set_sig_bytes(m_png_ptr, 8); //сообщить библиотеке что первые 8 байт отстутствуют

    png_read_info(m_png_ptr, m_info_ptr);//считывание файла с изб. в структуру

    //проверяем что файл png RGBA
                        if (png_get_color_type(m_png_ptr, m_info_ptr) != PNG_COLOR_TYPE_RGB_ALPHA){
                png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка структуры
        fclose(fp);

            QMessageBox::critical(nullptr,"Ошибка","У этого png файла тип цвета не RGBA\n\nДанная программа работает только с файлами png  , с типом цвета RGBA!!!");
```

```
    return 1;
}

//очищаем память от прошлого изображения(если оно было)
freeImage();


m_width = png_get_image_width(m_png_ptr, m_info_ptr);
m_height = png_get_image_height(m_png_ptr, m_info_ptr);
m_color_type = png_get_color_type(m_png_ptr, m_info_ptr);
m_bit_depth = png_get_bit_depth(m_png_ptr, m_info_ptr);

m_number_of_passes = png_set_interlace_handling(m_png_ptr);
png_read_update_info(m_png_ptr, m_info_ptr);



/* read file */
if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка структуры
    fclose(fp);
    // Some error handling: error during read_image

        QMessageBox::critical(nullptr,"Ошибка","Не удалось считать само изображение");
    return 1;
}


// выделение памяти
```

```
    m_arr_pixel = (png_bytep *) malloc(sizeof(png_bytep) * m_height);
    for (int i = 0; i < m_height; i++)
                m_arr_pixel[i] = (png_byte *) malloc(png_get_rowbytes(m_png_ptr,
m_info_ptr));

    // наконец считывание изображения
    png_read_image(m_png_ptr, m_arr_pixel);

            png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка
структуры
    fclose(fp);


    m_path_to_img = path;
    return 0;

}


//запись изображения на диск
void Image_PNG::writeOnDisk(QString path) {
    Image_PNG::writeTempImgOnDisk(path);
    m_path_to_img = path;

}


//установливает в vall значение заданного канала channel во всем изб.
void Image_PNG::filterRGBA (int value,RGBA channel){
    int x,y;
```

```
    for (y = 0; y < m_height; y++) {
        for (x = 0; x < m_width; x++) {//4 - количество каналов (в RGBA их 4)
            png_byte *pixel = &(m_arr_pixel[y][x*4]);
            pixel[channel] = value;
        }
    }
    //m_height+=100;

}

int Image_PNG::getHeight() {
    return this->m_height;
}

int Image_PNG::getWidth() {
    return this->m_width;
}


Image_PNG Image_PNG::getPieceImg(int p_x, int p_y, int width, int height){
    Image_PNG p_img;

    p_img.m_height = height;
    p_img.m_width = width;

    //выделяем память
    p_img.m_arr_pixel = (png_bytep *) malloc(sizeof(png_bytep) * p_img.m_height);
    for (int i = 0; i < p_img.m_height; i++)
        p_img.m_arr_pixel[i] = (png_byte *) malloc(sizeof(png_byte) * p_img.m_width
* 4);// так как канолов 4
```

```
    p_img.m_color_type = m_color_type;
    p_img.m_bit_depth = m_bit_depth;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < (width*4); x++) {//4 - количество каналов (в RGBA их 4)
            p_img.m_arr_pixel[y][x] = m_arr_pixel[y+p_y][x+p_x*4];
        }
    }

    return p_img;

}


QVector<QVector<Image_PNG>> Image_PNG::cutIntoPieces(int N, int M){
    QVector<QVector<Image_PNG>> arr_pieces_img;
    int x,y;


    //выделяем память
    arr_pieces_img.resize(N);
    for (int x = 0; x < N ; x++){
        arr_pieces_img[x].resize(M);
    }

    /*борьба с не кратностью изображения на N и на M*/
    //средняя длина и выстона  кусков
    int width_piece = m_width / N; //целочисленное деление
    int height_piece = m_height / M;

    //остатки
```

```
int remainder_x = m_width % N;
int remainder_y = m_height % M;

//прирост x
int inc_x  ;
int inc_y ;

//координаты текущего куска
int coordinate_x = 0;
int coordinate_y = 0;

//разрезание
inc_x = 1;
for (x = 0; x < N; x++) {
    if (remainder_x == x) {//сколько остатков столько и изображений которые нужно изменить по x
        inc_x = 0;//убираем надбавку ширины
    }

    inc_y = 1;

    for ( y = 0; y < M ; y++){
            if (remainder_y == y) {//сколько остатков столько и изображений которые нужно изменить по y
            inc_y = 0;//убираем надбавку высоты
        }

        arr_pieces_img[x][y] = getPieceImg(coordinate_x ,coordinate_y,width_piece + inc_x, height_piece + inc_y);
        coordinate_y += height_piece + inc_y;//ноыве коорд для след куска по y
    }
```

```
        coordinate_y = 0;

        coordinate_x += width_piece + inc_x;//ноыве коорд для след куска по х
    }



    return  arr_pieces_img;
}


//нарисовать окружность
void Image_PNG::drawCircle(int centr_x, int centr_y, int r, int d,    QColor color, QColor color_in) {
    int x,y;

    for (y = centr_y-r-d; y <= centr_y+r+d; y++) {//так как круг зажат в квадрате
        for (x = centr_x-r-d; x <= centr_x+r+d; x++) {//с коорд.  лев верх (centr_x-r-d,centr_y-r-d)
                                        //и низ прав (centr_x+r+d,centr_y+r+d)

            int crcl_in = qPow(x-centr_x,2) +qPow(y-centr_y,2) - qPow(r,2);
            int crcl_out = qPow(x-centr_x,2) +qPow(y-centr_y,2) - qPow(r+d,2);

            if (  crcl_in >= 0 && crcl_out < 0 ){
                setColorInPixel(x, y, color);
            }

        }
    }
```

```
    if (color_in.isValid()) {
        drawCircle(centr_x,centr_y,0,r,color_in);
    }

}


void Image_PNG::drawLine(int x1, int y1, int x2, int y2,int d , QColor color ) {
    int x,y;

    if (x1 == x2){ //вертикальная линия (tg равен inf)
        //уравнение вида x == const

        int inc_left = trunc((float)(d-1)/2);//окр в меньшую
        int inc_right = round((float)(d-1)/2);//окр в большую

        set_first_small_second_big(&y1,&y2);
        for (y = y1; y <= y2; y++){
            for(x = x1 - inc_left; x <= x1+inc_right; x++){ //создание тольщины линии
              setColorInPixel(x, y, color);
            }
        }
        return;
    }

    float tg_a = (float)(y1 - y2)/(x1 - x2);
    float b = y1 - tg_a * x1;

    if (abs(tg_a) <= 1) {   //тогда строим y = y(x)
        set_first_small_second_big(&x1,&x2);
```

```
        int inc_up = trunc((float)(d-1)/2);//окр в меньшую
        int inc_down = round((float)(d-1)/2);//окр в большую
        int y_main;

        for ( x = x1; x <= x2; x++) {
           y_main = round(tg_a*x + b);  // y = kx+b

                for(y = y_main - inc_up; y <= y_main+inc_down; y++){ //создание
тольщины линии
              setColorInPixel(x, y, color);
           }
        }

    } else { //строим x = x(y)
        set_first_small_second_big(&y1,&y2);

        int inc_left = trunc((float)(d-1)/2);//окр в меньшую
        int inc_right = round((float)(d-1)/2);//окр в большую
        int x_main;

        for ( y = y1; y <= y2; y++) {
           x_main = round( (y - b) / tg_a );  // x = (y-b)/k

                for(x = x_main - inc_left; x <= x_main+inc_right; x++){ //создание
тольщины линии
              setColorInPixel(x, y, color);
           }
        }
    }
}
```

```
void Image_PNG::set_first_small_second_big(int *small_p, int *big_p){
    int small = *small_p;
    int big = *big_p;

    if (small > big) {//нужно наоборот
        *small_p = big;
        *big_p = small;
    }
}

void Image_PNG::setColorInPixel(int x, int y,QColor color) {
    png_byte *pixel;

    if (x >= 0 && x < m_width && y >= 0 && y < m_height){
        pixel = &(m_arr_pixel[y][x*4]);
        pixel[R] = color.red();
        pixel[G] = color.green();
        pixel[B] = color.blue();
        pixel[A] = color.alpha();
    }
}


void Image_PNG::drawRectangle(int x, int y,int width,int height,int d ,QColor color_out,  QColor color_in ){

    int inc_left = (d-1)/2;//окр в меньшую
    int inc_right = round((float)(d-1)/2);//окр в большую
    int inc_up = inc_left;//окр в меньшую
    int inc_down = inc_right;//окр в большую
```

```
    drawLine(x , y + inc_up, x + width-1, y + inc_up, d, color_out);    // up ---
      drawLine(x+width-inc_right-1, y+d, x+width-inc_right-1, y + height - d-1, d, color_out);    // right |
        drawLine(x, y+height-inc_down-1, x + width-1, y+height-inc_down-1, d, color_out);   // down ---
    drawLine(x+inc_left, y+d, x+inc_left, y + height - d-1, d, color_out);    // left |

    if (color_in.isValid()) {
       for(int i = x+d; i < x+width-d; i++) {
          for (int j = y+d; j < y+height-d;j++ ) {
             setColorInPixel(i,j,color_in);
          }
       }
    }
}

void Image_PNG::drawRectWithDiag(int x, int y,int width,int height,int d ,QColor color_out,  QColor color_in ) {
    drawRectangle(x,y,width,height,d,color_out,color_in);
    drawLine(x+d,y+d,x+width-d-1,y+height-d-1,d,color_out);
    drawLine(x+d,y+height-d-1,x+width-d-1,y+d,d,color_out);
}
```