«Частное учреждение дополнительного образования

«Новосибирская академия дизайна и программирования»»

Дипломная работа

На тему: «Создание графического редактора

на языке программирования C++ и Qt 5»

Выполнил: Смагин Максим Алексеевич, 7 класс

Руководитель: Пахомов И.В.

Новосибирск, 2020

**Содержание.**

Введение.............................................................................................................................3

Цель работы........................................................................................................................3

Ход работы.........................................................................................................................4

Заключение.......................................................................................................................15

Исходники........................................................................................................................16

**Введение.**

Персональный компьютер – универсальное и распространённое среди пользователей любого возраста техническое устройство. Создано множество программ, с помощью которых компьютер работает во многих сферах деятельности, например – в общении, развлечениях, обработки информации. Одной из многих сфер является создание цифровых изображений. Приложения, работающие с созданием и редактированием изображений, называются графические редакторы.

Растровый графический редактор — специализированная программа, предназначенная для создания и обработки растровых изображений.  
Растровые изображения хранятся при помощи специальных точек – пикселей, которые образуют матрицу. Графический редактор автоматизирует этот процесс.

**Цель работы.**

Написать графический редактор на языке программирования C++ вместе с графической оболочкой Qt. Посмотреть, как программа устроена, её работу и основной функционал.

**Ход работы.**

Программа поделена для удобства на разные модели, находящиеся в разных файлах:

1. **main.cpp** – место хранения функции main(), которая запускается при открытии программы.
2. **mainmenu –** модуль отображения элементов интерфейса главного окна за исключением области рисования (см. п. 3).
3. **Модули GraphicsDisplay** позволяют взаимодействовать, рисовать на “воображаемом холсте”. Группы модулей таковы:
   1. **core** – инициализирует и хранит информацию об окне рисования и объектов – инструментов.
   2. **toolcore** содержит класс с общими свойствами для всех инструментов редактирования графики.
   3. **Сами инструменты (pencil, fill, vector и др.).**
4. **Файл внешних ресурсов –** содержит иконки для программы в формате .png.
5. **Файл проекта Qt** содержит всю информацию о сборке, библиотеках и исходниках.
6. **dialogcreatefile** рисует окно создания файла.

Все модули написанные мною записаны на Заголовочные файлы (.h) и файлы Исходников (.cpp), как по стандарту C++.

**main.cpp**: в этом компоненте начинается запуск самой программы, создаётся объект главного меню (MainMenu файлов mainmenu), вызывая функцию, которая возвращает значение только по завершению работы программы.

**mainmenu**: содержит класс MainMenu, который отвечает за отображения главного окна:

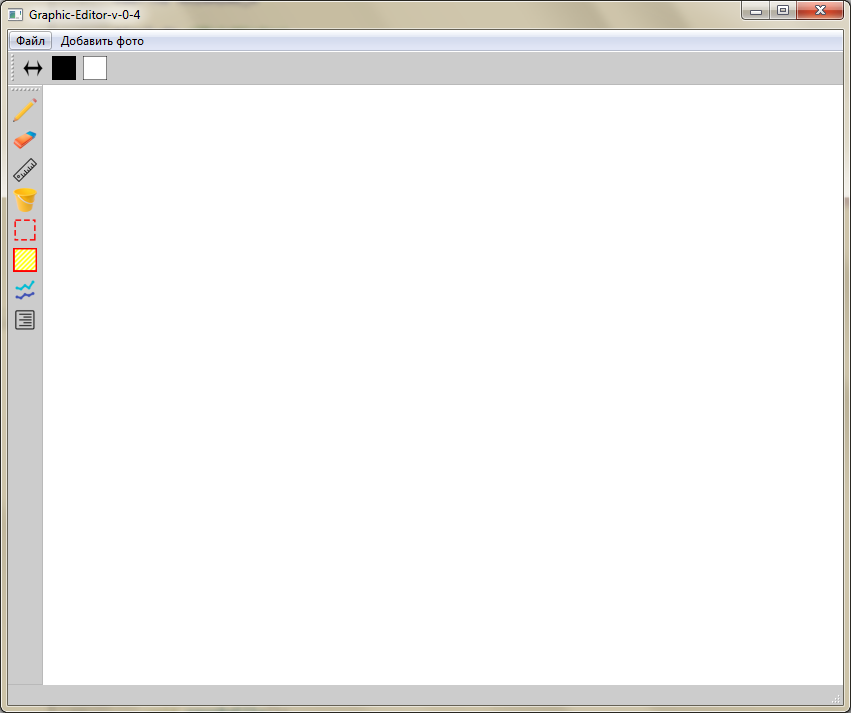


Рисунок 1 – основное окно программы.

Т.е. все кнопки, область рисования, меню инициализировано в конструкторе этого класса. Функции создания, открытия, сохранении файлов, также написаны здесь.

**Модули инструментов GraphicsDisplay** до компиляции хранятся в отдельной папке. Они отвечают за работу инструментов графики.

**core:** инициализирует и хранит данные, к примеру создаёт объекты инструментов, которые описаны ниже, хранит размеры и содержимое изображения.

**toolcore:** базовый класс – модуль, созданный для предотвращения повторения кода в написании классов инструментов.

**классы инструментов:**

**pencil:** эмуляция карандаша, сплошная линия одного из двух цветов, заданные пользователем и используемыми правой и левой кнопкой мыши.



Рисунок 2 – простой результат работы карандаша.

Сначала мы установили размер 12, нажав на иконку "двойную стрелочку", которая показала простое диалоговое окно выбора и предлагает выбрать нужный размер:

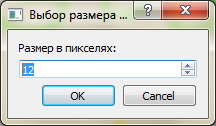


Рисунок 3 – диалоговое окно выбора цвета.

Теперь выберем главный и вторичный цвет, нажав на иконки верхнего меню C:\Users\Макс\Desktop\Снимок.PNG, по умолчанию которые чёрные и белые, вызвав стандартное меню выбора цвета Windows.

Модуль состоит из разных событий:

1. **MousePressEvent:** срабатывает при нажатии любой клавиши мыши. Позволяют программе понять, какая клавиша мыши нажата сохранив в переменную, заполняет первый пиксель и обновляет экран для показа изменений.
2. **MouseMoveEvent:** срабатывает при движении мышки с любой нажатой кнопкой мыши.
3. **paintEvent:** обновляет область рисования для показа изменений.

Все они участвуют в работе данного инструмента. Без первого клик бы по области (не движение с нажатием мыши) не приводил бы к изменениям. Без второго наоборот - только при движении мыши. А третье базовое для всего, что должно на области рисования обновиться.

**eraser:** старательная резинка. принцип работы как у карандаша (pencil), но нет возможности выбора цвета, вне зависимости от клавиши мыши цвет всегда белый.

**ruller:** эмуляция линейки, измеряющая расстояние между двумя точками - пикселями.

Координаты первой точки - место нажатия, которая сохраняется до отпускания мыши, а координаты второй - место курсора при удерживании и перемещении.

Само расстояние между двумя точками в двумерном пространстве высчитывается по формуле:

p1 - первая точка; p2 - вторая точка. x1 и y1 соответственно координаты первой точки, а x2 и y2 второй.

Координаты точек и расстояние между ними выводится в statusBar.



Рисунок 4 – демонстрация работы линейки.

**fill:** заливка. Заливка позволяет совершить заливку смежных пикселей, имеющие одинаковые коды RGB.

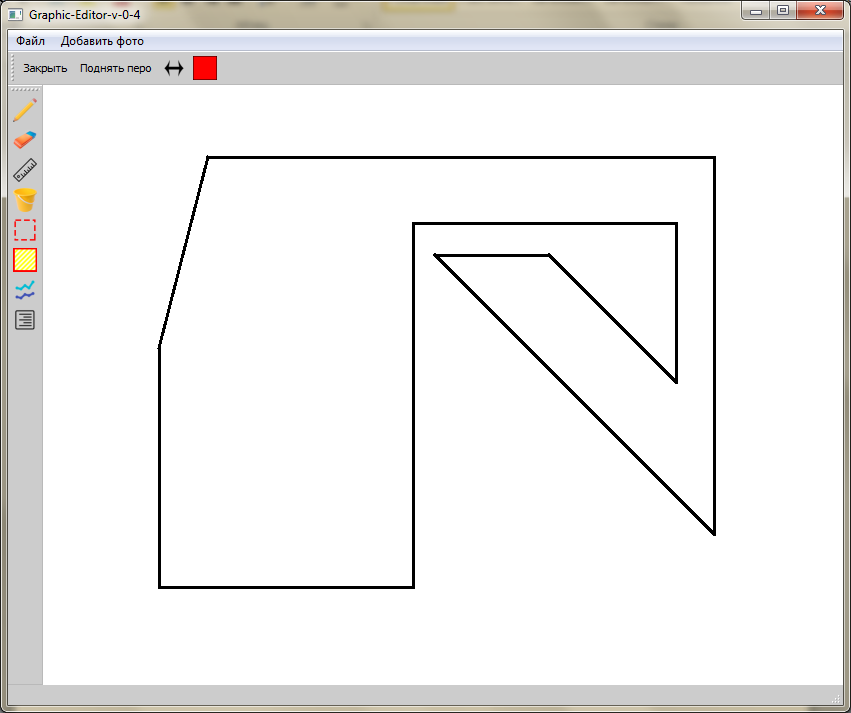


Рисунок 5 – исходный вид многоугольника.

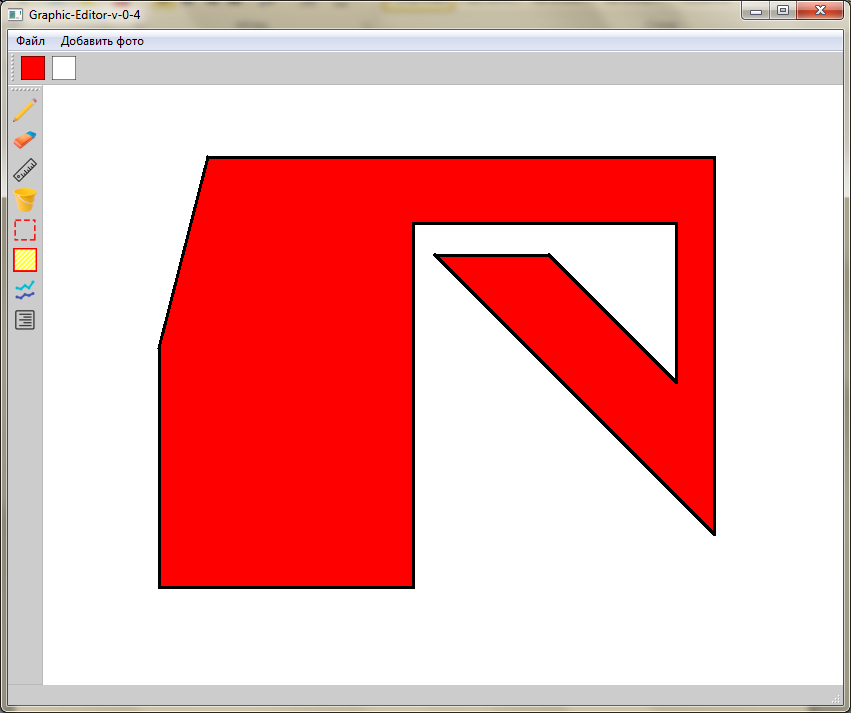


Рисунок 6 – внешний вид после применения заливки, кликнув внутри фигуры.

Работа данного инструмента основана на алгоритме **Scanline Flood Fill:**

**Заливка** (элемент, заменяемый цвет, цвет заливки):

1. Присвоить *Q* пустую очередь.

2. Если цвет *элемент*а - не *заменяемый цвет*, возврат.

3. Поместить *элемент* в *Q*.

4. Для каждого *N* из элементов *Q*:

5. Если цвет *N* - *заменяемый цвет*:

6. Присвоить *w* и *e* тот же элемент, что и *N*.

7. Смещать *w* на запад до тех пор, пока цвет *w* не станет отличаться от цвета "*заменяемый цвет*" .

8. Смещать *e* на восток до тех пор, пока цвет *e* не станет отличаться от цвета "*заменяемый цвет*".

9. Всем элементам между *w* и *e* придать *цвет заливки*.

10. Для каждого *n* между *w* и *e*:

11. Если цвет элемента к северу от *n* - *заменяемый цвет*, поместить этот элемент в *Q*.

Если цвет элемента к югу от *n* - *заменяемый цвет*, поместить этот элемент в *Q*.

12. Продолжать цикл, пока в *Q* останутся элементы.

13. Возврат. *{разумеется, в пп. 7, 8, а также 11 можно встретить края массива}*

**figures:** инструмент рисования простых геометрических фигур. Имеет выбираемый цвет контура и заполнения, а так же ширину контура.

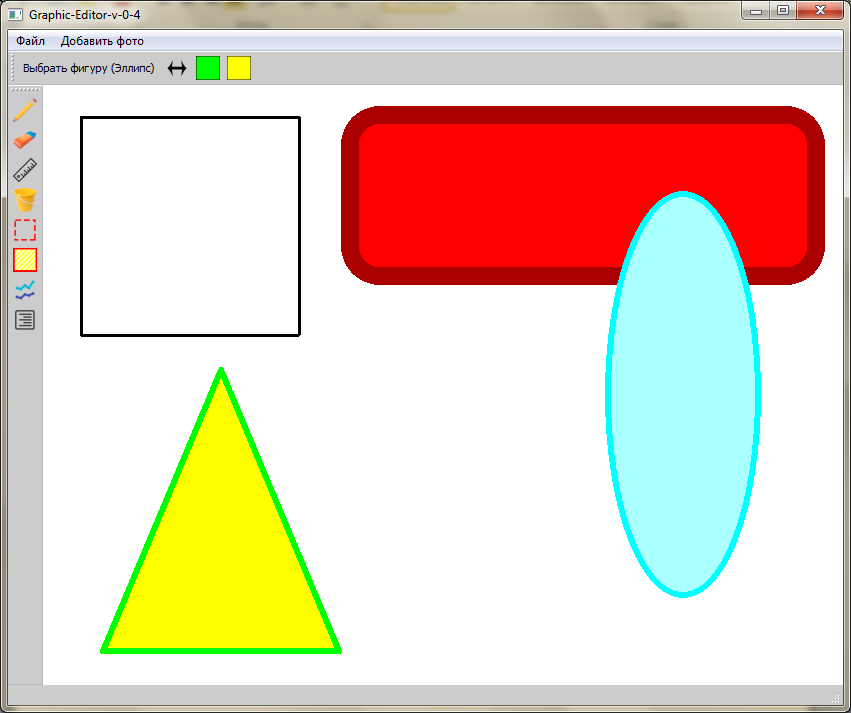


Рисунок 7 – пример рисование различных фигур.

Фигуры можно без проблем накладывать друг на друга, как голубой эллипс на красный прямоугольник с закруглёнными углами.

Рассмотрим, как рассчитываются точки – вершины треугольника, имея первую точку - место зажатие мышки, и вторую - место отпускания мышки.

На рисунке 8 точка в верхнем левом углу – первая точка, а в нижнем правом – вторая.

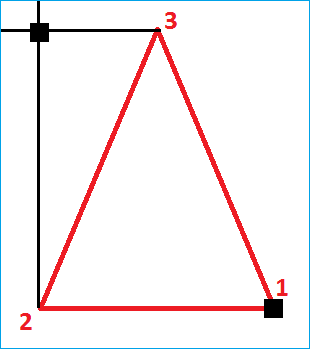


Рисунок 8 – треугольник с обозначенными точками выделения и вершинами.

Первая точка треугольника, не трудно догататься, соответствует точке отпускания мыши, вторая – координате X места нажимания мыши и координате Y места отжатия. А у третьей X равен разности X-ов точек отжатия и нажатия мыши, а Y точке нажатия Y.

**vector:** рисование ломанной (не соответствует названию). Кликами мыши определяются точки ломаной. Её можно закрыть, нажав на кнопку или оставить незакрытой (подняв перо).

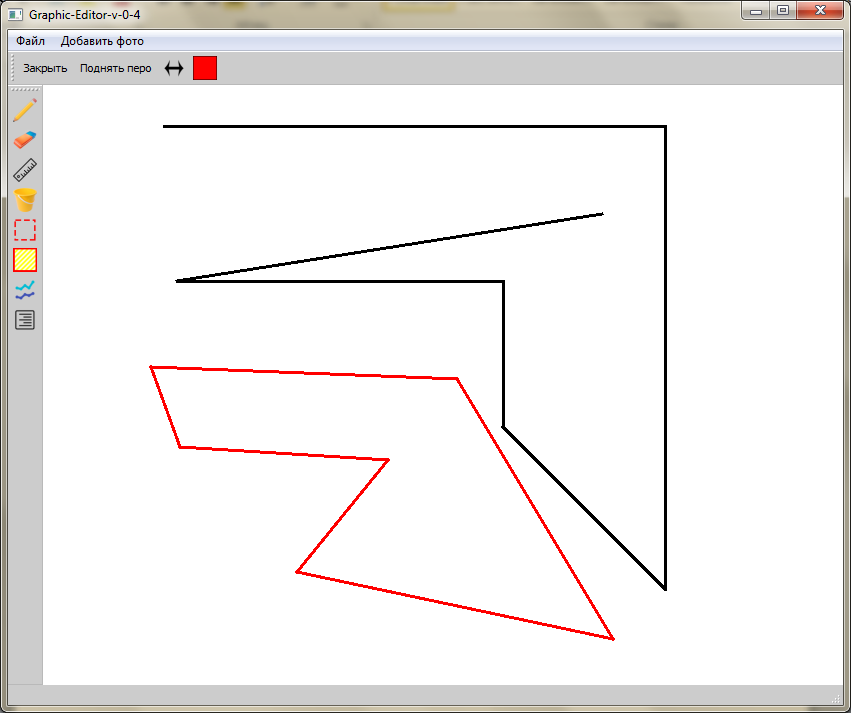


Рисунок 9 – демонстрация рисования ломаной. Красная закрытая, чёрная незакрытая.

**highlighting:** перемещение выделенной пользователем области.

Пользователь выделяет прямоугольную область с ПКМ (с нажатым Shift квадратную), затем когда выделил перемещает область в место клика указателя мыши.

Для удобства, между разными операциями, были создано разные состояния, такие как *выделение, движение выделенной области, сохранение изменений в ОЗУ и неактивное состояние.* Они были созданы того, что бы программа понимала, когда ждать от пользователя выделить область, а когда перемещать.

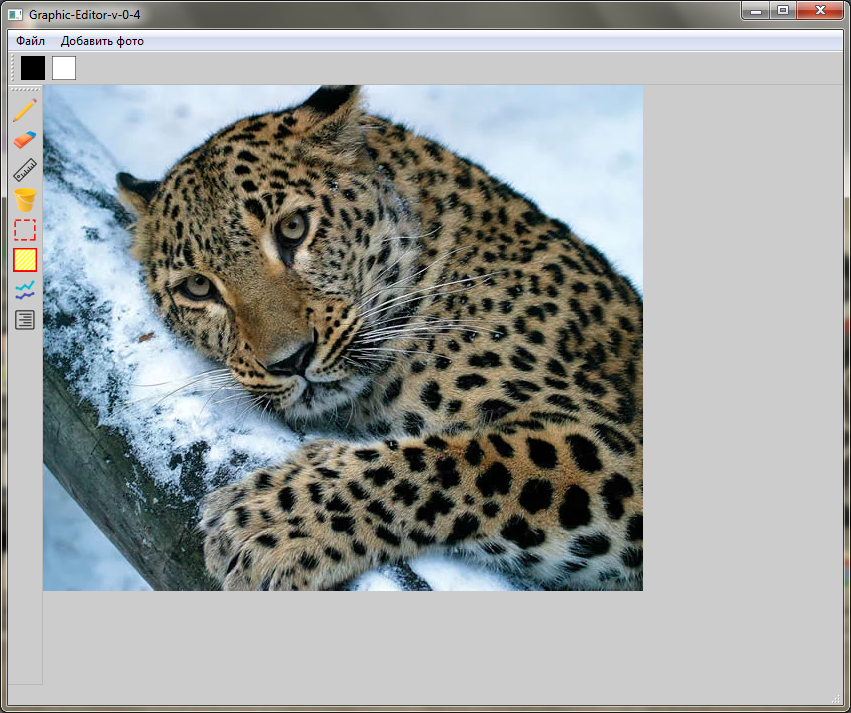


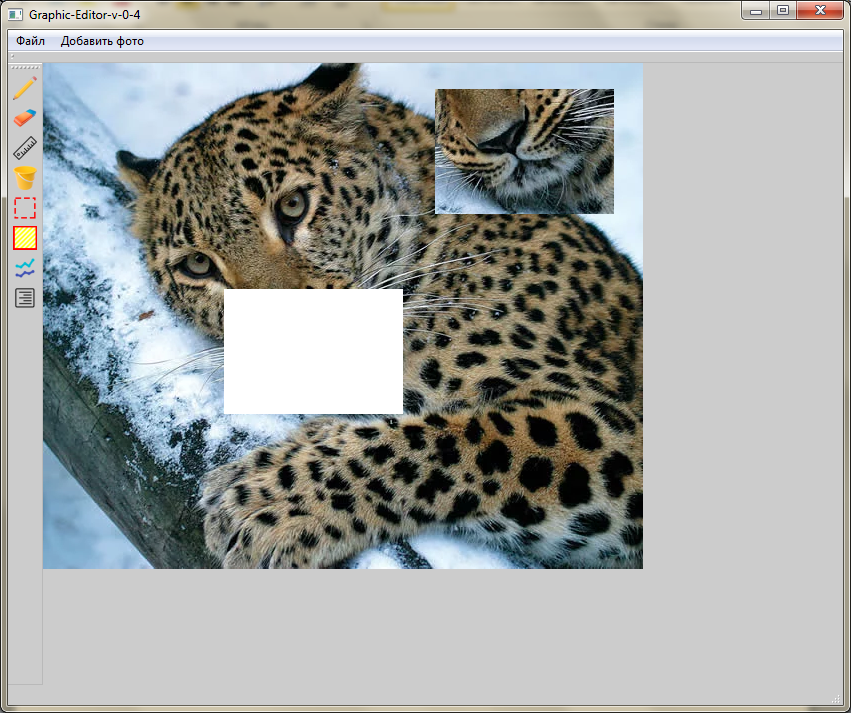
Рисунок 10 – исходный вид изображения, скаченного из интернета.

Рисунок 11 – демонстрация работы перемещения выделенной области.

**Заключение.**

В результате работы было изучено создание программ на C++ с графикой на Qt. Программу можно использовать для простой обработки графики.

Во время написании программы возникло немало вопросов и проблем, которые решились с помощью форумов, справочников и с помощью моего преподавателя Пахомова И.В.. На работу было потрачено примерно 40 часов.

**Исходники.**

**main.cpp:**

#include "mainmenu.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainMenu \*w = new MainMenu();

w->show();

return a.exec();

}

**mainmenu.h:**

#ifndef MAINMENU\_H

#define MAINMENU\_H

#include <QMainWindow>

#include "dialogcreatefile.h"

#include "GraphicDisplay/core.h"

class **MainMenu** : public QMainWindow

{

Q\_OBJECT

QMenuBar \*mainMenu;

QMenu \*menuFile;

QToolBar \*mainTools;

enum **selectToolIndex** { PENCIL, BRUSH, ERASSER, FILL, RULLER, HIGHLIGHTING, FIGURES, VECTOR, TEXT, INSERTPHOTO };

int selectedtoolindex;

QToolBar \*upTool;

QStatusBar \*statusInfo;

QString fileWay;

private:

virtual void ***closeEvent***(QCloseEvent \*event);

public:

GraphicDisplayCore \*graphicDisplay;

**MainMenu**(QWidget \*parent = nullptr);

~***MainMenu***();

public slots:

void **selectPensil**();

void **selectBrush**();

void **selectEraser**();

void **selectFill**();

void **selectRuller**();

void **selectHighlighting**();

void **selectFigures**();

void **selectVector**();

void **selectText**();

void **selectInsertPhoto**();

void **createFile**();

void **openFile**();

void **saveFile**();

void **saveAsFile**();

void **closeFile**();

void **addPhotoInDisplay**();

void **exitProgram**();

};

#endif // MAINMENU\_H

**mainmenu.cpp:**

#include "mainmenu.h"

MainMenu::MainMenu(QWidget \*parent)

: QMainWindow(parent)

{

//Инициализация:

upTool = new QToolBar("Меню инструмета");

mainMenu = new QMenuBar();

menuFile = new QMenu("Файл");

statusInfo = new QStatusBar();

graphicDisplay = new GraphicDisplayCore(&selectedtoolindex, upTool, statusInfo);

mainTools = new QToolBar("Базовые инструметы");

selectedtoolindex = 0;

fileWay = "";

//mainMenu:

mainMenu->addMenu(menuFile);

mainMenu->addAction("Добавить фото", this, &MainMenu::selectInsertPhoto);

menuFile->addAction(QIcon(), "Создать...", this, &MainMenu::createFile);

menuFile->addAction(QIcon(), "Открыть...", this, &MainMenu::openFile);

menuFile->addSeparator();

menuFile->addAction(QIcon(), "Сохранить", this, &MainMenu::saveFile);

menuFile->addAction(QIcon(), "Сохранить как...", this, &MainMenu::saveAsFile);

menuFile->addSeparator();

menuFile->addAction(QIcon(), "Закрыть файл", this, &MainMenu::closeFile);

menuFile->addAction(QIcon(), "Выйти", this, &MainMenu::exitProgram);

//mainTools:

mainTools->addAction(QIcon(":/Icons/Pencil.png"), "Карандаш", this, &MainMenu::selectPensil);

mainTools->addAction(QIcon(":/Icons/Erasser.png"), "Ластик", this, &MainMenu::selectEraser);

mainTools->addAction(QIcon(":/Icons/Ruller.png"), "Линейка", this, &MainMenu::selectRuller);

mainTools->addAction(QIcon(":/Icons/Fill.png"), "Заполнение", this, &MainMenu::selectFill);

mainTools->addAction(QIcon(":/Icons/Highlighting"), "Выделение", this, &MainMenu::selectHighlighting);

mainTools->addAction(QIcon(":/Icons/Figures.png"), "Фигуры", this, &MainMenu::selectFigures);

mainTools->addAction(QIcon(":/Icons/Vector.png"), "Вектора", this, &MainMenu::selectVector);

mainTools->addAction(QIcon(":/Icons/Text.png"), "Текст", this, &MainMenu::selectText);

//this:

setStyleSheet("QMainWindow {background: #CCCCCC;}");

setCentralWidget(graphicDisplay->checkedToolPtr);

centralWidget()->setMinimumSize(800, 600);

resize(800, 600);

setMenuBar(mainMenu);

addToolBar(Qt::LeftToolBarArea, mainTools);

addToolBar(Qt::TopToolBarArea, upTool);

setStatusBar(statusInfo);

selectPensil();

}

MainMenu::~MainMenu()

{

}

/\*virtual\*/ void MainMenu::closeEvent(QCloseEvent \*event) {

QMessageBox \*question = new QMessageBox(QMessageBox::Icon::Question, "Выход из программы", "Сохранить файл перед закрытием?", QMessageBox::StandardButton::Ok | QMessageBox::StandardButton::No | QMessageBox::StandardButton::Cancel);

int answer = question->exec();

if(answer == QMessageBox::StandardButton::Ok) {

saveFile();

}

else if(answer == QMessageBox::StandardButton::No);

else event->ignore();

}

//Слоты выбора базового инструмета:

void MainMenu::selectPensil() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetPencil);

graphicDisplay->widgetPencil->isSelect();

selectedtoolindex = PENCIL;

}

void MainMenu::selectBrush() {

if(auto\* p=centralWidget()){ p->setParent(nullptr); }

setCentralWidget(graphicDisplay->widgetBrush);

graphicDisplay->widgetBrush->isSelect();

selectedtoolindex = BRUSH;

}

void MainMenu::selectEraser() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetErasser);

graphicDisplay->widgetErasser->isSelect();

selectedtoolindex = ERASSER;

}

void MainMenu::selectFill() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetFill);

graphicDisplay->widgetFill->isSelect();

selectedtoolindex = FILL;

}

void MainMenu::selectRuller() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetRuller);

graphicDisplay->widgetRuller->isSelect();

selectedtoolindex = RULLER;

}

void MainMenu::selectHighlighting() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetHighlighting);

graphicDisplay->widgetHighlighting->isSelect();

selectedtoolindex = HIGHLIGHTING;

}

void MainMenu::selectFigures() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetFigures);

graphicDisplay->widgetFigures->isSelect();

selectedtoolindex = FIGURES;

}

void MainMenu::selectVector() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetVector);

graphicDisplay->widgetVector->isSelect();

selectedtoolindex = VECTOR;

}

void MainMenu::selectText() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetText);

graphicDisplay->widgetText->isSelect();

selectedtoolindex = TEXT;

}

void MainMenu::selectInsertPhoto() {

if(auto\* p=centralWidget()){p->setParent(nullptr);}

setCentralWidget(graphicDisplay->widgetInsertPhoto);

graphicDisplay->widgetInsertPhoto->isSelect();

selectedtoolindex = INSERTPHOTO;

}

void MainMenu::createFile() {

if(fileWay == "") {

int width, height;

DialogCreateFile dialog(&width, &height, this);

if(dialog.exec() == QDialog::Accepted) {

\*graphicDisplay->display = graphicDisplay->display->scaled(width, height);

centralWidget()->resize(width, height);

centralWidget()->setMinimumSize(width, height);

resize(width, height);

for(int ind = 0; ind < graphicDisplay->display->width(); ind++) {

for(int jnd = 0; jnd < graphicDisplay->display->height(); jnd++) {

graphicDisplay->display->setPixelColor(ind, jnd, Qt::white);

}

}

}

}

else {

QMessageBox \*question = new QMessageBox(QMessageBox::Icon::Question, "Создание изображения", "Сохранить предыдущую фотографию перед закрытием?", QMessageBox::StandardButton::Ok | QMessageBox::StandardButton::No | QMessageBox::StandardButton::Cancel, this);

int answer = question->exec();

if(answer == QMessageBox::StandardButton::Ok) {

saveFile();

int width, height;

DialogCreateFile dialog(&width, &height, this);

if(dialog.exec() == QDialog::Accepted) {

\*graphicDisplay->display = graphicDisplay->display->scaled(width, height);

centralWidget()->resize(width, height);

for(int ind = 0; ind < graphicDisplay->display->width(); ind++) {

for(int jnd = 0; jnd < graphicDisplay->display->height(); jnd++) {

graphicDisplay->display->setPixelColor(ind, jnd, Qt::white);

}

}

}

}

else if(answer == QMessageBox::StandardButton::No) {

int width, height;

DialogCreateFile dialog(&width, &height, this);

if(dialog.exec() == QDialog::Accepted) {

\*graphicDisplay->display = graphicDisplay->display->scaled(width, height);

centralWidget()->resize(width, height);

for(int ind = 0; ind < graphicDisplay->display->width(); ind++) {

for(int jnd = 0; jnd < graphicDisplay->display->height(); jnd++) {

graphicDisplay->display->setPixelColor(ind, jnd, Qt::white);

}

}

}

}

delete question;

}

}

void MainMenu::openFile() {

QMessageBox \*question = new QMessageBox(QMessageBox::Icon::Question, "Создание изображения", "Сохранить предыдущую фотографию перед закрытием?", QMessageBox::StandardButton::Ok | QMessageBox::StandardButton::No | QMessageBox::StandardButton::Cancel, this);

int answer = question->exec();

if(answer == QMessageBox::StandardButton::Ok) {

saveFile();

QString buf\_way = QFileDialog::getOpenFileName(this, "Открыть файл...", "", "Images (\*.png, \*.jpg, \*.bmp)");

if(buf\_way != "") {

graphicDisplay->display->load(buf\_way);

graphicDisplay->checkedToolPtr->update();

fileWay = buf\_way;

}

}

else if(answer == QMessageBox::StandardButton::No) {

QString buf\_way = QFileDialog::getOpenFileName(this, "Открыть файл...", "", "Images (\*.png \*.jpg \*.bmp)");

if(buf\_way != "") {

graphicDisplay->display->load(buf\_way);

graphicDisplay->checkedToolPtr->update();

fileWay = buf\_way;

}

}

delete question;

}

void MainMenu::saveFile() {

if(fileWay != "") graphicDisplay->display->save(fileWay.replace("/", "//"));

else saveAsFile();

}

void MainMenu::saveAsFile() {

QString buf\_way = QFileDialog::getSaveFileName(this, "Сохранить как...", "", "\*.png;;\*.jpg;;\*.bmp)");

if(buf\_way != "") {

graphicDisplay->display->save(buf\_way);

fileWay = buf\_way;

}

}

void MainMenu::closeFile() {

fileWay = "";

for(int ind = 0; ind < graphicDisplay->display->width(); ind++) {

for(int jnd = 0; jnd < graphicDisplay->display->height(); jnd++) {

graphicDisplay->display->setPixelColor(ind, jnd, Qt::white);

}

}

graphicDisplay->checkedToolPtr->update();

}

void MainMenu::addPhotoInDisplay() {

}

void MainMenu::exitProgram() {

QMessageBox \*question = new QMessageBox(QMessageBox::Icon::Question, "Выход из программы", "Сохранить файл перед закрытием?", QMessageBox::StandardButton::Ok | QMessageBox::StandardButton::No | QMessageBox::StandardButton::Cancel);

int answer = question->exec();

if(answer == QMessageBox::StandardButton::Ok) {

saveFile();

}

else if(answer == QMessageBox::StandardButton::No);

else return;

close();

}

**GraphicsDisplay/core.h:**

#ifndef GRAPHICDISPLAYCORE\_H

#define GRAPHICDISPLAYCORE\_H

#include <QtWidgets>

#include "GraphicDisplay/pencil.h"

#include "GraphicDisplay/brush.h"

#include "GraphicDisplay/erasser.h"

#include "GraphicDisplay/fill.h"

#include "GraphicDisplay/ruller.h"

#include "GraphicDisplay/highlighting.h"

#include "GraphicDisplay/figures.h"

#include <GraphicDisplay/vector.h>

#include "GraphicDisplay/text.h"

#include "GraphicDisplay/insertphoto.h"

class GraphicDisplayCore : public QObject

{

Q\_OBJECT

public:

QSize imageSize;

QImage \*display;

QColor \*selectedColor1, \*selectedColor2;

QVector<QColor> colorList;

int pointBeginX, pointBeginY, pointEndX, pointEndY;

int \*toolIndexPtr;

QToolBar \*uptool;

QStatusBar \*statusInfo;

int penWidth;

enum selectToolIndex { PENCIL, BRUSH, ERASSER, RULLER, HIGHLIGHTING, FIGURES, VECTOR, TEXT };

QWidget \*checkedToolPtr;

ToolPencil \*widgetPencil;

ToolBrush \*widgetBrush;

ToolErasser \*widgetErasser;

ToolFill \*widgetFill;

ToolRuller \*widgetRuller;

ToolHighlighting \*widgetHighlighting;

ToolFigures \*widgetFigures;

ToolVector \*widgetVector;

ToolText \*widgetText;

ToolInsertPhoto \*widgetInsertPhoto;

GraphicDisplayCore(int \*toolIndexPtr, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent = nullptr);

};

#endif // GRAPHICDISPLAYCORE\_H

**GraphicsDisplay/core.cpp:**

#include "core.h"

GraphicDisplayCore::**GraphicDisplayCore**(int \*toolIndexPtr, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent) : QObject(parent)

{

//Инициализация:

imageSize = QSize(800, 600);

display = new QImage(imageSize.width(), imageSize.height(), QImage::Format\_RGB32);

selectedColor1 = nullptr;

selectedColor2 = nullptr;

colorList << QColor(0, 0, 0) << QColor(255, 0, 0) << QColor(0, 255, 0) << QColor(0, 255, 0) << QColor(0, 0, 255) << QColor(255, 255, 255);

selectedColor1 = &colorList[0];

selectedColor2 = &colorList[colorList.count()-1];

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

this->toolIndexPtr = toolIndexPtr;

this->uptool = uptool;

this->statusInfo = statusInfo;

penWidth = 3;

widgetPencil = new ToolPencil(display, selectedColor1, selectedColor2, uptool, statusInfo, &penWidth);

widgetBrush = new ToolBrush(display, selectedColor1, selectedColor2, uptool, statusInfo, &penWidth);

widgetErasser = new ToolErasser(display, uptool, statusInfo, &penWidth);

widgetFill = new ToolFill(display, selectedColor1, selectedColor2, uptool, statusInfo);

widgetRuller = new ToolRuller(display, uptool, statusInfo);

widgetHighlighting = new ToolHighlighting(display, uptool, statusInfo);

widgetFigures = new ToolFigures(display, selectedColor1, selectedColor2, uptool, statusInfo, &penWidth);

widgetVector = new ToolVector(display, selectedColor1, selectedColor2, uptool, statusInfo, &penWidth);

widgetText = new ToolText(display, selectedColor1, selectedColor2, uptool, statusInfo, &penWidth);

widgetInsertPhoto = new ToolInsertPhoto(display, uptool, statusInfo);

checkedToolPtr = widgetPencil;

//this:

}

**GraphicsDisplay/toolcore.h:**

#ifndef TOOLCORE\_H

#define TOOLCORE\_H

#include <QtWidgets>

class ToolCore : public QWidget

{

public:

bool pressedLeftButton;

QImage \*display;

QColor \*selectedColor1, \*selectedColor2;

bool isrepaint;

QToolBar \*uptool;

QStatusBar \*statusInfo;

int \*penWidthPtr;

bool manualPaint;

int iconSize;

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

virtual void resizeEvent(QResizeEvent \*event);

ToolCore(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent = nullptr);

void isSelect();

ToolCore& operator=(ToolCore& right);

public slots:

void uptool\_selectResize();

void uptool\_selectColor1();

void uptool\_selectColor2();

};

#endif // TOOLCORE\_H

**GraphicsDisplay/toolcore.cpp:**

#include "toolcore.h"

ToolCore::ToolCore(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent) : QWidget(parent)

{

pressedLeftButton = 0;

this->display = display;

this->selectedColor1 = selectedColor1;

this->selectedColor2 = selectedColor2;

isrepaint = 0;

this->uptool = uptool;

this->statusInfo = statusInfo;

this->penWidthPtr = penWidthPtr;

manualPaint = 0;

iconSize = 50;

for(int ind = 0; ind < display->width(); ind++) {

for(int jnd = 0; jnd < display->height(); jnd++) {

display->setPixelColor(ind, jnd, Qt::white);

}

}

resize(display->width(), display->height());

}

ToolCore& ToolCore::operator=(ToolCore& right) {

return right;

}

/\*virtual\*/ void ToolCore::mousePressEvent(QMouseEvent \*event) { }

/\*virtual\*/ void ToolCore::mouseMoveEvent(QMouseEvent \*event) { }

/\*virtual\*/ void ToolCore::mouseReleaseEvent(QMouseEvent \*event) { }

/\*virtual\*/ void ToolCore::paintEvent(QPaintEvent \*) { }

/\*virtual\*/ void ToolCore::resizeEvent(QResizeEvent \*event) { }

void ToolCore::isSelect() {

uptool->clear();

QPixmap \*icon = new QPixmap(50, 50);

QPainter paint(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor1);

paint.end();

uptool->addAction(QIcon(":/Icons/Width.png"), "Размер пера", this, &ToolCore::uptool\_selectResize);

uptool->addAction(QIcon(\*icon), "Цвет(1)", this, &ToolCore::uptool\_selectColor1);

icon = new QPixmap(50, 50);

paint.begin(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor2);

paint.end();

uptool->addAction(QIcon(\*icon), "Цвет(2)", this, &ToolCore::uptool\_selectColor2);

}

void ToolCore::uptool\_selectResize() {

bool npressedCancel;

int width = QInputDialog::getInt(this, "Выбор размера инструмента:", "Размер в пикселях:", \*penWidthPtr, 0, 1000, 1, &npressedCancel);

if(!npressedCancel) {

return;

}

\*penWidthPtr = width;

}

void ToolCore::uptool\_selectColor1() {

QColor buf;

{

QColor color = QColorDialog::getColor(\*selectedColor1, this, "Выбор цвета(1)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

buf = color;

}

\*selectedColor1 = buf;

isSelect();

}

void ToolCore::uptool\_selectColor2() {

{

QColor color = QColorDialog::getColor(\*selectedColor2, this, "Выбор цвета(2)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

\*selectedColor2 = color;

isSelect();

}

}

**GraphicsDisplay/brush.h:**

#ifndef TOOLBRUSH\_H

#define TOOLBRUSH\_H

#include <QtWidgets>

#include "pencil.h"

class ToolBrush : public ToolPencil

{

Q\_OBJECT

protected:

virtual void paintEvent(QPaintEvent \*event);

public:

ToolBrush(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent = nullptr);

};

#endif // TOOLBRUSH\_H

**GraphicsDisplay/brush.cpp:**

#include "brush.h"

ToolBrush::ToolBrush(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent) : ToolPencil(display, selectedColor1, selectedColor2, uptool, statusInfo, penWidthPtr, parent)

{

}

/\*virtual\*/ void ToolBrush::paintEvent(QPaintEvent \*event) {

if(pressedLeftButton) {

QPainter paint(display);

paint.setRenderHint(QPainter::Antialiasing, true);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

if(!isrepaint) {

if(!(pointBeginX == pointEndX && pointBeginY == pointEndY)) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else { paint.drawPoint(pointBeginX, pointBeginY); }

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

else {

QPainter paint(display);

paint.setRenderHint(QPainter::Antialiasing, true);

paint.setPen(QPen(\*selectedColor2, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

if(!isrepaint) {

if(!(pointBeginX == pointEndX && pointBeginY == pointEndY)) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else { paint.drawPoint(pointBeginX, pointBeginY); }

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

}

**GraphicsDisplay/eraser.h:**

#ifndef TOOLERASSER\_H

#define TOOLERASSER\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolErasser : public ToolCore

{

Q\_OBJECT

int pointBeginX, pointBeginY, pointEndX, pointEndY;

int \*penWidthPtr;

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

void uptool\_selectColor1();

void uptool\_selectColor2();

public:

void isSelect();

ToolErasser(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent = nullptr);

};

#endif // TOOLERASSER\_H

**GraphicsDisplay/eraser.cpp:**

#include "erasser.h"

ToolErasser::ToolErasser(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent) : ToolCore(display, nullptr, nullptr, uptool, statusInfo, penWidthPtr, parent)

{

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

this->display = display;

this->penWidthPtr = penWidthPtr;

}

/\*virtual\*/ void ToolErasser::mousePressEvent(QMouseEvent \*event) {

manualPaint = 1;

if(event->buttons() == Qt::LeftButton) {

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

pressedLeftButton = 1;

}

else {

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

pressedLeftButton = 0;

}

update();

}

/\*virtual\*/ void ToolErasser::mouseMoveEvent(QMouseEvent \*event) {

manualPaint = 1;

if(event->buttons() == Qt::LeftButton) {

if(pointBeginX == pointEndX) {

pointEndX = event->x();

pointEndY = event->y();

}

pressedLeftButton = 1;

}

else {

if(pointBeginX == pointEndX) {

pointEndX = event->x();

pointEndY = event->y();

}

pressedLeftButton = 0;

}

update();

}

/\*virtual\*/ void ToolErasser::mouseReleaseEvent(QMouseEvent \*event) {

}

/\*virtual\*/ void ToolErasser::paintEvent(QPaintEvent \*event) {

if(manualPaint) {

QPainter paint(display);

paint.setPen(QPen(Qt::white, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

if(!isrepaint) {

if(!(pointBeginX == pointEndX && pointBeginY == pointEndY)) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else { paint.drawPoint(pointBeginX, pointBeginY); }

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.end();

}

manualPaint = 0;

}

void ToolErasser::isSelect() {

uptool->clear();

uptool->addAction(QIcon(":/Icons/Width.png"), "Размер пера", this, &ToolCore::uptool\_selectResize);

}

void ToolErasser::uptool\_selectColor1() {

QColor buf;

{

QColor color = QColorDialog::getColor(\*selectedColor1, this, "Выбор цвета(1)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

buf = color;

}

\*selectedColor1 = buf;

isSelect();

}

void ToolErasser::uptool\_selectColor2() {

{

QColor color = QColorDialog::getColor(\*selectedColor2, this, "Выбор цвета(2)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

\*selectedColor2 = color;

isSelect();

}

}

**GraphicsDisplay/figures.h:**

#ifndef TOOLFIGURES\_H

#define TOOLFIGURES\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolFigures : public ToolCore

{

Q\_OBJECT

QStringList figureList;

int figureSelectIndex;

short figureIsDraw;

QPoint pointStart, pointEnd;

bool directMod;

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

virtual void keyPressEvent(QKeyEvent \*event);

virtual void keyReleaseEvent(QKeyEvent \*event);

void uptool\_selectColor1();

void uptool\_selectColor2();

QPoint directCoordinates(int x1, int y1, int x2, int y2);

public:

enum figuresIndexes { RECTANGLE, ROUNDEDRECTANGLE, ELLIPSE, TRIANGLE };

ToolFigures(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent = nullptr);

void isSelect();

void figureSelect();

};

#endif // TOOLFIGURES\_H

struct FillInTheArea {

};

**GraphicsDisplay/figures.cpp:**

#include "figures.h"

ToolFigures::**ToolFigures**(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent) : ToolCore(display, selectedColor1, selectedColor2, uptool, statusInfo, penWidthPtr, parent)

{

figureList << "Квадрат" << "Квадрат с закруглёнными краями" << "Эллипс" << "Треугольник";

figureSelectIndex = 0;

figureIsDraw = 0;

directMod = 0;

setFocusPolicy(Qt::StrongFocus);

}

void ToolFigures::**isSelect**() {

uptool->clear();

QPixmap \*icon = new QPixmap(50, 50);

QPainter paint(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor1);

paint.end();

uptool->addAction(QIcon(), QString("Выбрать фигуру (%1)").arg(figureList.at(figureSelectIndex)), this, &ToolFigures::figureSelect);

uptool->addAction(QIcon(":/Icons/Width.png"), "Размер обводки", this, &ToolFigures::uptool\_selectResize);

uptool->addAction(QIcon(\*icon), "Цвет обводки", this, &ToolFigures::uptool\_selectColor1);

icon = new QPixmap(50, 50);

paint.begin(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor2);

paint.end();

uptool->addAction(QIcon(\*icon), "Цвет заполнения", this, &ToolFigures::uptool\_selectColor2);

}

void ToolFigures::**figureSelect**() {

QString buf\_figure = QInputDialog::getItem(this, "Выбрать фигуру", "Выберите фигуру из списка:", figureList);

for(int ind = 0; ind < figureList.count(); ind++) {

if(figureList.at(ind) == buf\_figure) {

figureSelectIndex = ind;

isSelect();

}

}

}

/\*virtual\*/ void ToolFigures::***mousePressEvent***(QMouseEvent \*event) {

figureIsDraw = 1;

pointStart.setX(event->x());

pointStart.setY(event->y());

update();

}

/\*virtual\*/ void ToolFigures::***mouseMoveEvent***(QMouseEvent \*event) {

pointEnd.setX(event->x());

pointEnd.setY(event->y());

update();

}

/\*virtual\*/ void ToolFigures::***mouseReleaseEvent***(QMouseEvent \*event) {

figureIsDraw = 2;

pointEnd.setX(event->x());

pointEnd.setY(event->y());

repaint();

figureIsDraw = 0;

}

/\*virtual\*/ void ToolFigures::***paintEvent***(QPaintEvent \*) {

QPoint point = directCoordinates(pointStart.x(), pointStart.y(), pointEnd.x(), pointEnd.y());

if(figureIsDraw == 0) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

}

else if(figureIsDraw == 1) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

paint.setBrush(\*selectedColor2);

if(directMod) {

switch (figureSelectIndex) {

case 0:

paint.drawRect(QRect(pointStart, point));

break;

case 1:

paint.drawRoundedRect(QRect(pointStart, point), 30, 30);

break;

case 2:

paint.drawEllipse(QRect(pointStart, point));

break;

case 3:

QPoint p1(pointStart.x(), point.y());

QPoint p2(point);

QPoint p3((pointStart.x()+point.x())/2, pointStart.y());

QVector<QPoint> lines;

lines << p1 << p2 << p3;

paint.drawPolygon(QPolygon(lines));

break;

}

}

else {

switch (figureSelectIndex) {

case 0:

paint.drawRect(QRect(pointStart, pointEnd));

break;

case 1:

paint.drawRoundedRect(QRect(pointStart, pointEnd), 30, 30);

break;

case 2:

paint.drawEllipse(QRect(pointStart, pointEnd));

break;

case 3:

QPoint p1(pointStart.x(), pointEnd.y());

QPoint p2(pointEnd);

QPoint p3((pointStart.x()+pointEnd.x())/2, pointStart.y());

QVector<QPoint> lines;

lines << p1 << p2 << p3;

paint.drawPolygon(QPolygon(lines));

break;

}

}

}

else if(figureIsDraw == 2) {

QPainter paint(display);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

paint.setBrush(\*selectedColor2);

if(directMod) {

switch (figureSelectIndex) {

case 0:

paint.drawRect(QRect(pointStart, point));

break;

case 1:

paint.drawRoundedRect(QRect(pointStart, point), 30, 30);

break;

case 2:

paint.drawEllipse(QRect(pointStart, point));

break;

case 3:

QPoint p1(pointStart.x(), point.y());

QPoint p2(point);

QPoint p3((pointStart.x()+point.x())/2, pointStart.y());

QVector<QPoint> lines;

lines << p1 << p2 << p3;

paint.drawPolygon(QPolygon(lines));

break;

}

}

else {

qDebug() << "=";

switch (figureSelectIndex) {

case 0:

paint.drawRect(QRect(pointStart, pointEnd));

break;

case 1:

paint.drawRoundedRect(QRect(pointStart, pointEnd), 30, 30);

break;

case 2:

paint.drawEllipse(QRect(pointStart, pointEnd));

break;

case 3:

QPoint p1(pointStart.x(), pointEnd.y());

QPoint p2(pointEnd);

QPoint p3((pointStart.x()+pointEnd.x())/2, pointStart.y());

QVector<QPoint> lines;

lines << p1 << p2 << p3;

paint.drawPolygon(QPolygon(lines));

break;

}

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

}

}

void ToolFigures::**uptool\_selectColor1**() {

QColor buf;

{

QColor color = QColorDialog::getColor(\*selectedColor1, this, "Выбор цвета(1)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

buf = color;

}

\*selectedColor1 = buf;

isSelect();

}

void ToolFigures::**uptool\_selectColor2**() {

{

QColor color = QColorDialog::getColor(\*selectedColor2, this, "Выбор цвета(2)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

\*selectedColor2 = color;

isSelect();

}

}

QPoint ToolFigures::**directCoordinates**(int x1, int y1, int x2, int y2) {

double direction = (atan2(y2 - y1, x2 - x1) / acos(-1) \* 180)+90;

if(direction > 180 && direction <= 270) direction = -(270-direction+90); //170 в 100; 160 в 110

int DC = 180/2;

if(direction < -DC) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1+xd) : QPoint(x1-yd, y1+yd); }

else if(direction < 0) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1-xd) : QPoint(x1-yd, y1-yd); }

else if(direction < DC) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1-xd) : QPoint(x1+yd, y1-yd); }

else if(direction <= DC\*2) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1+xd) : QPoint(x1+yd, y1+yd); }

return QPoint(0, 0);

}

/\*virtual\*/ void ToolFigures::***keyPressEvent***(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 1; update(); }

}

/\*virtual\*/ void ToolFigures::***keyReleaseEvent***(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 0; update(); }

}

**GraphicsDisplay/fill.h:**

#ifndef TOOLFILL\_H

#define TOOLFILL\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolFill : public ToolCore

{

Q\_OBJECT

QImage \*display;

int colorTolerance;

QColor seachColor, fillColor;

void paintProcess(QPoint point);

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

public:

void isSelect();

ToolFill(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent = nullptr);

};

#endif // TOOLFILL\_H

**GraphicsDisplay/fill.cpp:**

#include "fill.h"

ToolFill::ToolFill(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent) : ToolCore(display, selectedColor1, selectedColor2, uptool, statusInfo, nullptr, parent)

{

this->display = display;

colorTolerance = 1;

fillColor = qRgb(255, 0, 0);

}

/\*virtual\*/ void ToolFill::mousePressEvent(QMouseEvent \*event) {

if(event->buttons() == Qt::LeftButton) fillColor = \*selectedColor1;

else if(event->buttons() == Qt::RightButton) fillColor = \*selectedColor2;

else return;

seachColor = display->pixelColor(event->x(), event->y());

paintProcess(event->pos());

update();

}

void ToolFill::paintProcess(QPoint point) {

QStack<QPoint> pixels;

if(seachColor == fillColor) return;

pixels.push(point);

while(!pixels.empty()) {

if(display->pixelColor(pixels.top()) == seachColor) {

QPoint w, e;

w = e = pixels.top();

while(display->pixelColor(w) == seachColor && display->pixelColor(w.x()-1, w.y()) == seachColor) w.setX(w.x()-1);

while(display->pixelColor(e) == seachColor && display->pixelColor(e.x()+1, e.y()) == seachColor) e.setX(e.x()+1);

for(int ind = w.x(); ind <= e.x(); ind++) display->setPixelColor(ind, w.y(), fillColor);

for(int width = w.x(); width <= e.x(); width++) {

if(display->pixelColor(width, w.y()-1) == seachColor) { pixels.push(QPoint(width, w.y()-1)); }

if(display->pixelColor(width, w.y()+1) == seachColor) { pixels.push(QPoint(width, w.y()+1)); }

}

}

else pixels.pop();

}

update();

}

/\*virtual\*/ void ToolFill::paintEvent(QPaintEvent \*event) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.end();

}

void ToolFill::isSelect() {

uptool->clear();

QPixmap \*icon = new QPixmap(50, 50);

QPainter paint(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor1);

paint.end();

uptool->addAction(QIcon(\*icon), "Цвет(1)", this, &ToolCore::uptool\_selectColor1);

icon = new QPixmap(50, 50);

paint.begin(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor2);

paint.end();

uptool->addAction(QIcon(\*icon), "Цвет(2)", this, &ToolCore::uptool\_selectColor2);

}

**GraphicsDisplay/highlighting.h:**

#ifndef TOOLHIGHLIGHTING\_H

#define TOOLHIGHLIGHTING\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolHighlighting : public ToolCore

{

Q\_OBJECT

short clicksNum;

QPoint pointBegin, pointEnd, pointCursor;

QImage \*highlightingZone;

bool directMod;

void reset();

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

virtual void keyPressEvent(QKeyEvent \*event);

virtual void keyReleaseEvent(QKeyEvent \*event);

QPoint directCoordinates(int x1, int y1, int x2, int y2);

enum state { OFF, HIGHLIGHTING, CLEAR, MOVING, SET };

public:

ToolHighlighting(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent = nullptr);

void isSelect();

};

#endif // TOOLHIGHLIGHTING\_H

**GraphicsDisplay/highlighting.cpp:**

#include "highlighting.h"

ToolHighlighting::ToolHighlighting(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent) : ToolCore(display, nullptr, nullptr, uptool, statusInfo, nullptr, parent)

{

clicksNum = OFF;

pointBegin = pointEnd = pointCursor = QPoint(0, 0);

directMod = 0;

setMouseTracking(true);

setFocusPolicy(Qt::StrongFocus);

}

/\*virtual\*/ void ToolHighlighting::mousePressEvent(QMouseEvent \*event) {

if(clicksNum == OFF) {

pointBegin.setX(event->x());

pointBegin.setY(event->y());

pointEnd = pointBegin;

clicksNum = HIGHLIGHTING;

update();

}

else if(clicksNum == MOVING) {

pointCursor.setX(event->x());

pointCursor.setY(event->y());

clicksNum = SET;

update();

}

}

/\*virtual\*/ void ToolHighlighting::mouseMoveEvent(QMouseEvent \*event) {

if(clicksNum == HIGHLIGHTING) {

pointEnd.setX(event->x());

pointEnd.setY(event->y());

update();

}

if(clicksNum == MOVING) {

pointCursor.setX(event->x());

pointCursor.setY(event->y());

update();

}

}

/\*virtual\*/ void ToolHighlighting::mouseReleaseEvent(QMouseEvent \*event) {

if(clicksNum == HIGHLIGHTING) {

pointEnd.setX(event->x());

pointEnd.setY(event->y());

int width = pointEnd.x()-pointBegin.x()+1, height = pointEnd.y()-pointBegin.y()+1;

highlightingZone = new QImage(width, height, QImage::Format::Format\_RGB32);

for(int ind = 0; ind < width; ind++) {

for(int jnd = 0; jnd < height; jnd++) {

highlightingZone->setPixelColor(ind, jnd, display->pixel(pointBegin.x()+ind, pointBegin.y()+jnd));

}

}

clicksNum = CLEAR;

update();

}

}

/\*virtual\*/ void ToolHighlighting::paintEvent(QPaintEvent \*) {

if(clicksNum == OFF) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.end();

}

else if(clicksNum == HIGHLIGHTING) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.setPen(QPen(QBrush(qRgba(255, 0, 0, 128)), 3, Qt::PenStyle::DashLine));

if(directMod) paint.drawRect(QRect(pointBegin, directCoordinates(pointBegin.x(), pointBegin.y(), pointEnd.x(), pointEnd.y())));

else paint.drawRect(QRect(pointBegin, pointEnd));

paint.end();

}

else if(clicksNum == CLEAR) {

QPainter paint(display);

paint.fillRect(QRect(pointBegin, pointEnd), Qt::white);

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.drawImage(pointCursor, \*highlightingZone);

paint.end();

clicksNum = MOVING;

}

else if(clicksNum == MOVING) {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.drawImage(pointCursor, \*highlightingZone);

}

else if(clicksNum == SET) {

QPainter paint(display);

paint.drawImage(pointCursor, \*highlightingZone);

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

reset();

}

}

void ToolHighlighting::isSelect() {

uptool->clear();

}

void ToolHighlighting::reset() {

clicksNum = OFF;

pointBegin = pointEnd = pointCursor = QPoint(0, 0);

delete highlightingZone;

}

/\*virtual\*/ void ToolHighlighting::keyPressEvent(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 1; update(); }

}

/\*virtual\*/ void ToolHighlighting::keyReleaseEvent(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 0; update(); }

}

QPoint ToolHighlighting::directCoordinates(int x1, int y1, int x2, int y2) {

double direction = (atan2(y2 - y1, x2 - x1) / acos(-1) \* 180)+90;

if(direction > 180 && direction <= 270) direction = -(270-direction+90); //170 в 100; 160 в 110

int DC = 180/4;

if(direction < -DC\*3.5) { return QPoint(x1, y2);}

else if(direction < -DC\*2.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1+xd) : QPoint(x1-yd, y1+yd); }

else if(direction < -DC\*1.5) { return QPoint(x2, y1); }

else if(direction < -DC\*0.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1-xd) : QPoint(x1-yd, y1-yd); }

else if(direction < DC\*0.5) { return QPoint(x1, y2); }

else if(direction < DC\*1.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1-xd) : QPoint(x1+yd, y1-yd); }

else if(direction < DC\*2.5) { return QPoint(x2, y1);}

else if(direction < DC\*3.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1+xd) : QPoint(x1+yd, y1+yd); }

else if(direction <= DC\*4.0) { return QPoint(x1, y2); }

return QPoint(0, 0);

}

**GraphicsDisplay/insertphoto.h:**

#ifndef TOOLINSERTPHOTO\_H

#define TOOLINSERTPHOTO\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolInsertPhoto : public ToolCore

{

Q\_OBJECT

QString insertPhotoWay;

QImage insertPhotoFile;

int mouseCoordinateX, mouseCoordinateY;

bool mouseClick;

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

public:

ToolInsertPhoto(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent = nullptr);

void isSelect();

public slots:

void insertPhoto();

};

#endif // TOOLINSERTPHOTO\_H

**GraphicsDisplay/insertphoto.cpp:**

#include "insertphoto.h"

ToolInsertPhoto::**ToolInsertPhoto**(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent) : ToolCore(display, nullptr, nullptr, uptool, statusInfo, nullptr, parent)

{

this->display = display;

this->uptool = uptool;

this->statusInfo = statusInfo;

insertPhotoWay = "";

setMouseTracking(true);

mouseCoordinateX = mouseCoordinateY = 0;

mouseClick = false;

}

/\*virtual\*/ void ToolInsertPhoto::***mousePressEvent***(QMouseEvent \*event) {

mouseClick = true;

update();

}

/\*virtual\*/ void ToolInsertPhoto::***mouseMoveEvent***(QMouseEvent \*event) {

if(insertPhotoWay != "") {

mouseCoordinateX = event->x();

mouseCoordinateY = event->y();

update();

}

}

/\*virtual\*/ void ToolInsertPhoto::***paintEvent***(QPaintEvent \*event) {

if(insertPhotoWay != "") {

if(!mouseClick) {

QPainter paint(this);

paint.setBrush(QBrush(QColor(0, 0, 0, 128)));

paint.drawImage(0, 0, \*display);

paint.drawImage(mouseCoordinateX, mouseCoordinateY, insertPhotoFile);

}

else {

QPainter paint(display);

paint.drawImage(mouseCoordinateX, mouseCoordinateY, insertPhotoFile);

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

mouseClick = 0;

insertPhotoWay = "";

}

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

}

}

void ToolInsertPhoto::**isSelect**() {

uptool->clear();

uptool->addAction(QIcon(), "Вставить фото", this, &ToolInsertPhoto::insertPhoto);

}

void ToolInsertPhoto::**insertPhoto**() {

if(insertPhotoWay == "") {

QString buf\_way = QFileDialog::getOpenFileName(this, "Добавить фото из...", "", "Images (\*.png \*.jpg \*.bmp)");

if(buf\_way != "") {

insertPhotoWay = buf\_way;

insertPhotoFile.load(insertPhotoWay);

}

}

}

**GraphicsDisplay/pencil.h:**

#ifndef TOOLPENCIL\_H

#define TOOLPENCIL\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolPencil : public ToolCore

{

protected:

int pointBeginX, pointBeginY, pointEndX, pointEndY;

int \*penWidthPtr;

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

public:

ToolPencil(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent = nullptr);

public slots:

};

#endif // TOOLPENCIL\_H

**GraphicsDisplay/pencil.cpp:**

#include "pencil.h"

ToolPencil::ToolPencil(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidthPtr, QWidget \*parent) : ToolCore(display, selectedColor1, selectedColor2, uptool, statusInfo, penWidthPtr, parent)

{

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

pressedLeftButton = 0;

this->display = display;

this->selectedColor1 = selectedColor1;

this->selectedColor2 = selectedColor2;

this->uptool = uptool;

this->penWidthPtr = penWidthPtr;

}

/\*virtual\*/ void ToolPencil::mousePressEvent(QMouseEvent \*event) {

manualPaint = 1;

if(event->buttons() == Qt::LeftButton) {

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

display->setPixelColor(pointBeginX, pointBeginY, \*selectedColor1);

pressedLeftButton = 1;

}

else {

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

display->setPixelColor(pointBeginX, pointBeginY, \*selectedColor2);

pressedLeftButton = 0;

}

update();

}

/\*virtual\*/ void ToolPencil::mouseMoveEvent(QMouseEvent \*event) {

manualPaint = 1;

if(event->buttons() == Qt::LeftButton) {

if(pointBeginX == pointEndX) {

pointEndX = event->x();

pointEndY = event->y();

}

display->setPixelColor(pointBeginX, pointBeginY, \*selectedColor1);

pressedLeftButton = 1;

}

else {

if(pointBeginX == pointEndX) {

pointEndX = event->x();

pointEndY = event->y();

}

display->setPixelColor(pointBeginX, pointBeginY, \*selectedColor2);

pressedLeftButton = 0;

}

update();

}

/\*virtual\*/ void ToolPencil::mouseReleaseEvent(QMouseEvent \*event) {

}

/\*virtual\*/ void ToolPencil::paintEvent(QPaintEvent \*) {

if(pressedLeftButton) {

if(manualPaint) {

QPainter paint(display);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

if(!isrepaint) {

if(!(pointBeginX == pointEndX && pointBeginY == pointEndY)) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else { paint.drawPoint(pointBeginX, pointBeginY); }

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.end();

}

}

else {

if(manualPaint) {

QPainter paint(display);

paint.setPen(QPen(\*selectedColor2, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

if(!isrepaint) {

if(!(pointBeginX == pointEndX && pointBeginY == pointEndY)) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else { paint.drawPoint(pointBeginX, pointBeginY); }

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

paint.end();

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

paint.end();

}

}

manualPaint = 0;

}

**GraphicsDisplay/ruller.h:**

#ifndef TOOLRULLER\_H

#define TOOLRULLER\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolRuller : public ToolCore

{

Q\_OBJECT

int pointBeginX, pointBeginY, pointEndX, pointEndY;

bool isPressed;

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

public:

void isSelect();

ToolRuller(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent = nullptr);

};

#endif // TOOLRULLER\_H

**GraphicsDisplay/ruller.cpp:**

#include "ruller.h"

ToolRuller::ToolRuller(QImage \*display, QToolBar \*uptool, QStatusBar \*statusInfo, QWidget \*parent) : ToolCore(display, nullptr, nullptr, uptool, statusInfo, nullptr, parent)

{

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

this->statusInfo = statusInfo;

this->display = display;

this->uptool = uptool;

isPressed = 0;

}

/\*virtual\*/ void ToolRuller::mousePressEvent(QMouseEvent \*event) {

isPressed = 1;

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

statusInfo->showMessage(QString("p1(%1, %2), p2(%3, %4). Расстояние: %5.").arg(pointBeginX).arg(pointBeginY).arg(pointEndX).arg(pointBeginY).arg(sqrt((pointBeginX-pointEndX)\*(pointBeginX-pointEndX)+(pointBeginY-pointEndY)\*(pointBeginY-pointEndY))));

update();

}

/\*virtual\*/ void ToolRuller::mouseMoveEvent(QMouseEvent \*event) {

pointEndX = event->x();

pointEndY = event->y();

update();

statusInfo->showMessage(QString("p1(%1, %2), p2(%3, %4). Расстояние: %5.").arg(pointBeginX).arg(pointBeginY).arg(pointEndX).arg(pointEndY).arg(sqrt((pointBeginX-pointEndX)\*(pointBeginX-pointEndX)+(pointBeginY-pointEndY)\*(pointBeginY-pointEndY))));

update();

}

/\*virtual\*/ void ToolRuller::mouseReleaseEvent(QMouseEvent \*event) {

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

statusInfo->showMessage("");

update();

isPressed = 0;

}

/\*virtual\*/ void ToolRuller::paintEvent(QPaintEvent \*) {

QPainter paint(this);

paint.setPen(QPen(Qt::black, 3));

paint.drawImage(0, 0, \*display);

if(isPressed) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

}

void ToolRuller::isSelect() {

uptool->clear();

}

**GraphicsDisplay/text.h:**

#ifndef TOOLTEXT\_H

#define TOOLTEXT\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolText : public ToolCore

{

Q\_OBJECT

int pointBeginX, pointBeginY, pointEndX, pointEndY;

bool paintText;

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void mouseReleaseEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

public:

ToolText(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent = nullptr);

};

#endif // TOOLTEXT\_H

**GraphicsDisplay/text.cpp:**

#include "text.h"

ToolText::ToolText(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent) : ToolCore(display, selectedColor1, selectedColor2, uptool, statusInfo, penWidth, parent)

{

pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

paintText = 0;

this->statusInfo = statusInfo;

this->display = display;

this->uptool = uptool;

this->selectedColor1 = selectedColor1;

this->selectedColor2 = selectedColor2;

this->penWidthPtr = penWidth;

}

/\*virtual\*/ void ToolText::mousePressEvent(QMouseEvent \*event) {

manualPaint = 1;

if(event->button() == Qt::LeftButton) pressedLeftButton = 1;

else pressedLeftButton = 0;

pointBeginX = pointEndX = event->x();

pointBeginY = pointEndY = event->y();

update();

}

/\*virtual\*/ void ToolText::mouseMoveEvent(QMouseEvent \*event) {

pointEndX = event->x();

pointEndY = event->y();

update();

}

/\*virtual\*/ void ToolText::mouseReleaseEvent(QMouseEvent \*event) {

paintText = 1;

repaint();

paintText = manualPaint = 0;

pointBeginX = pointEndX = 0;

pointBeginY = pointEndY = 0;

}

/\*virtual\*/ void ToolText::paintEvent(QPaintEvent \*) {

if(manualPaint) {

if(pressedLeftButton) {

if(paintText) {

bool rs;

paintText = 0;

QString text = QInputDialog::getText(this, "Добавить текст", "Введите текст для добавления на картинку:", QLineEdit::EchoMode::Normal, QString(), &rs);

paintText = 1;

if(rs) {

QPainter paint(display);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

QFont fontinfo;

fontinfo.setPixelSize(\*penWidthPtr <= 10 ? 10 : \*penWidthPtr);

paint.setFont(fontinfo);

paint.drawText(pointBeginX, pointBeginY, pointEndX-pointBeginX, pointEndY-pointBeginY, 0, text);

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

}

else return;

}

else {

QPainter paint(this);

paint.setPen(QPen(Qt::red, 3, Qt::DashLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

paint.drawImage(0, 0, \*display);

paint.drawRect(pointBeginX, pointBeginY, pointEndX-pointBeginX, pointEndY-pointBeginY);

}

}

else {

if(paintText) {

bool rs;

paintText = 0;

QString text = QInputDialog::getText(this, "Добавить текст", "Введите текст для добавления на картинку:", QLineEdit::EchoMode::Normal, QString(), &rs);

paintText = 1;

if(rs) {

QPainter paint(display);

paint.setPen(QPen(\*selectedColor2, \*penWidthPtr, Qt::SolidLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

QFont fontinfo;

fontinfo.setPixelSize(\*penWidthPtr <= 10 ? 10 : \*penWidthPtr);

paint.setFont(fontinfo);

paint.drawText(pointBeginX, pointBeginY, pointEndX-pointBeginX, pointEndY-pointBeginY, 0, text);

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

}

else return;

}

else {

QPainter paint(this);

paint.setPen(QPen(Qt::red, 3, Qt::DashLine, Qt::PenCapStyle::RoundCap, Qt::PenJoinStyle::RoundJoin));

paint.drawImage(0, 0, \*display);

paint.drawRect(pointBeginX, pointBeginY, pointEndX-pointBeginX, pointEndY-pointBeginY);

}

}

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

}

}

**GraphicDisplay/vector.h:**

#ifndef TOOLVECTOR\_H

#define TOOLVECTOR\_H

#include <QtWidgets>

#include "toolcore.h"

class ToolVector : public ToolCore

{

Q\_OBJECT

bool isClicked, isStart, isStop;

int pointStartX, pointStartY, pointBeginX, pointBeginY, pointEndX, pointEndY;

bool directMod;

protected:

virtual void mousePressEvent(QMouseEvent \*event);

virtual void mouseMoveEvent(QMouseEvent \*event);

virtual void paintEvent(QPaintEvent \*event);

virtual void keyPressEvent(QKeyEvent \*event);

virtual void keyReleaseEvent(QKeyEvent \*event);

void uptool\_selectColor1();

void uptool\_selectColor2();

QPoint directCoordinates(int x1, int y1, int x2, int y2);

public:

ToolVector(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent = nullptr);

void isSelect();

public slots:

void vectorClose();

void vectorUp();

};

#endif // TOOLVECTOR\_H

**GraphicDisplay/vector.cpp:**

#include "vector.h"

ToolVector::ToolVector(QImage \*display, QColor \*selectedColor1, QColor \*selectedColor2, QToolBar \*uptool, QStatusBar \*statusInfo, int \*penWidth, QWidget \*parent) : ToolCore(display, selectedColor1, selectedColor2, uptool, statusInfo, penWidth, parent)

{

isClicked = isStart = isStop = 0;

pointStartX = pointStartY = pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

directMod = 0;

setMouseTracking(true);

setFocusPolicy(Qt::StrongFocus);

}

/\*virtual\*/ void ToolVector::mousePressEvent(QMouseEvent \*event) {

isClicked = 1;

if(!isStart) {

pointStartX = pointBeginX = pointEndX = event->x();

pointStartY = pointBeginY = pointEndY = event->y();

isStart = 1;

}

else {

pointEndX = event->x();

pointEndY = event->y();

}

update();

}

/\*virtual\*/ void ToolVector::mouseMoveEvent(QMouseEvent \*event) {

if(isStart) {

pointEndX = event->x();

pointEndY = event->y();

update();

}

}

/\*virtual\*/ void ToolVector::paintEvent(QPaintEvent \*) {

if(isStart) {

if(isClicked) {

qDebug() << pointBeginX << pointBeginY << " - " << pointEndX << pointEndY;

QPainter paint(display);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::SquareCap));

if(pointBeginX == pointEndX && pointBeginY == pointEndY) {

paint.drawPoint(pointBeginX, pointBeginY);

}

else {

if(!directMod) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else paint.drawLine(QPoint(pointBeginX, pointBeginY), directCoordinates(pointBeginX, pointBeginY, pointEndX, pointEndY));

}

paint.end();

paint.begin(this);

paint.drawImage(0, 0, \*display);

isClicked = 0;

if(directMod) {

QPoint point = directCoordinates(pointBeginX, pointBeginY, pointEndX, pointEndY);

pointBeginX = pointEndX = point.x();

pointBeginY = pointEndY = point.y();

}

else {

pointBeginX = pointEndX;

pointBeginY = pointEndY;

}

}

else {

QPainter paint(this);

paint.setPen(QPen(\*selectedColor1, \*penWidthPtr, Qt::SolidLine, Qt::SquareCap));

paint.drawImage(0, 0, \*display);

if(pointBeginX == pointEndX && pointBeginY == pointEndY) {

paint.drawPoint(pointBeginX, pointBeginY);

}

else {

if(!directMod) paint.drawLine(pointBeginX, pointBeginY, pointEndX, pointEndY);

else paint.drawLine(QPoint(pointBeginX, pointBeginY), directCoordinates(pointBeginX, pointBeginY, pointEndX, pointEndY));

}

}

}

else {

QPainter paint(this);

paint.drawImage(0, 0, \*display);

}

}

void ToolVector::isSelect() {

uptool->clear();

QPixmap \*icon = new QPixmap(50, 50);

QPainter paint(icon);

paint.fillRect(0, 0, 50, 50, qRgba(0, 0, 0, 255)) ;

paint.fillRect(0+1, 0+1, 50-2, 50-2, \*selectedColor1);

paint.end();

uptool->addAction(QIcon(), "Закрыть", this, &ToolVector::vectorClose);

uptool->addAction(QIcon(), "Поднять перо", this, &ToolVector::vectorUp);

uptool->addAction(QIcon(":/Icons/Width.png"), "Размер пера", this, &ToolVector::uptool\_selectResize);

uptool->addAction(QIcon(\*icon), "Цвет", this, &ToolVector::uptool\_selectColor1);

}

void ToolVector::vectorClose() {

pointEndX = pointStartX;

pointEndY = pointStartY;

isClicked = 1;

repaint();

isClicked = isStart = 0;

pointStartX = pointStartY = pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

}

void ToolVector::vectorUp() {

isClicked = isStart = 0;

pointStartX = pointStartY = pointBeginX = pointBeginY = pointEndX = pointEndY = 0;

update();

}

void ToolVector::uptool\_selectColor1() {

QColor buf;

{

QColor color = QColorDialog::getColor(\*selectedColor1, this, "Выбор цвета(1)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

buf = color;

}

\*selectedColor1 = buf;

isSelect();

}

void ToolVector::uptool\_selectColor2() {

{

QColor color = QColorDialog::getColor(\*selectedColor2, this, "Выбор цвета(2)");

//Если цвет не выбран (нажата Cancel)

if(!color.isValid()) {

return;

}

\*selectedColor2 = color;

isSelect();

}

}

/\*virtual\*/ void ToolVector::keyPressEvent(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 1; update(); }

}

/\*virtual\*/ void ToolVector::keyReleaseEvent(QKeyEvent \*event) {

if(event->key() == Qt::Key\_Shift) { directMod = 0; update(); }

}

QPoint ToolVector::directCoordinates(int x1, int y1, int x2, int y2) {

double direction = (atan2(y2 - y1, x2 - x1) / acos(-1) \* 180)+90;

if(direction > 180 && direction <= 270) direction = -(270-direction+90); //170 в 100; 160 в 110

int DC = 180/4;

if(direction < -DC\*3.5) { return QPoint(x1, y2);}

else if(direction < -DC\*2.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1+xd) : QPoint(x1-yd, y1+yd); }

else if(direction < -DC\*1.5) { return QPoint(x2, y1); }

else if(direction < -DC\*0.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1-xd, y1-xd) : QPoint(x1-yd, y1-yd); }

else if(direction < DC\*0.5) { return QPoint(x1, y2); }

else if(direction < DC\*1.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1-xd) : QPoint(x1+yd, y1-yd); }

else if(direction < DC\*2.5) { return QPoint(x2, y1);}

else if(direction < DC\*3.5) { int xd=x2-x1,yd=y2-y1; xd = abs(xd); yd = abs(yd); return +xd > +yd ? QPoint(x1+xd, y1+xd) : QPoint(x1+yd, y1+yd); }

else if(direction <= DC\*4.0) { return QPoint(x1, y2); }

return QPoint(0, 0);

}

**dialogcreatefile.h:**

#ifndef DIALOGCREATEFILE\_H

#define DIALOGCREATEFILE\_H

#include <QtWidgets>

class DialogCreateFile : public QDialog

{

Q\_OBJECT

QVBoxLayout \*mainBoxLayout;

QHBoxLayout \*fieldWidth, \*fieldHeight;

QLabel \*labelMain, \*labelWidthDisplay, \*labelHeightDisplay;

QLineEdit \*leWidthDisplay, \*leHeightDisplay;

QPushButton \*pbOk;

int \*widthPtr, \*heightPtr;

public:

DialogCreateFile(int \*widthPtr, int \*heightPtr, QWidget \*parent = nullptr);

~DialogCreateFile();

public slots:

void fOk();

};

#endif // DIALOGCREATEFILE\_H

**dialogcreatefile.cpp:**

#include "dialogcreatefile.h"

DialogCreateFile::DialogCreateFile(int \*widthPtr, int \*heightPtr, QWidget \*parent) : QDialog(parent)

{

mainBoxLayout = new QVBoxLayout();

fieldWidth = new QHBoxLayout();

fieldHeight = new QHBoxLayout();

labelMain = new QLabel("Установить размеры изображения:");

labelWidthDisplay = new QLabel("Ширина: ");

labelHeightDisplay = new QLabel("Высота: ");

leWidthDisplay = new QLineEdit();

leHeightDisplay = new QLineEdit();

pbOk = new QPushButton(QIcon(), "Создать");

this->widthPtr = widthPtr;

this->heightPtr = heightPtr;

//this:

QObject::connect(pbOk, &QPushButton::clicked, this, &DialogCreateFile::fOk);

leWidthDisplay->setValidator(new QIntValidator(1, 2000));

leHeightDisplay->setValidator(new QIntValidator(1, 2000));

mainBoxLayout->addWidget(labelMain);

mainBoxLayout->addLayout(fieldWidth);

mainBoxLayout->addLayout(fieldHeight);

mainBoxLayout->addWidget(pbOk);

fieldWidth->addWidget(labelWidthDisplay);

fieldWidth->addWidget(leWidthDisplay);

fieldHeight->addWidget(labelHeightDisplay);

fieldHeight->addWidget(leHeightDisplay);

setLayout(mainBoxLayout);

setWindowTitle("Создание изображения...");

show();

}

DialogCreateFile::~DialogCreateFile() {

}

void DialogCreateFile::fOk() {

\*widthPtr = leWidthDisplay->text().toInt();

\*heightPtr = leHeightDisplay->text().toInt();

accept();

}

**Graphic-Editor-v-0-4.pro:**

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

GraphicDisplay/brush.cpp \

GraphicDisplay/core.cpp \

GraphicDisplay/erasser.cpp \

GraphicDisplay/figures.cpp \

GraphicDisplay/fill.cpp \

GraphicDisplay/highlighting.cpp \

GraphicDisplay/insertphoto.cpp \

GraphicDisplay/pencil.cpp \

GraphicDisplay/ruller.cpp \

GraphicDisplay/text.cpp \

GraphicDisplay/toolcore.cpp \

GraphicDisplay/vector.cpp \

dialogcreatefile.cpp \

main.cpp \

mainmenu.cpp

HEADERS += \

GraphicDisplay/brush.h \

GraphicDisplay/core.h \

GraphicDisplay/erasser.h \

GraphicDisplay/figures.h \

GraphicDisplay/fill.h \

GraphicDisplay/highlighting.h \

GraphicDisplay/insertphoto.h \

GraphicDisplay/pencil.h \

GraphicDisplay/ruller.h \

GraphicDisplay/text.h \

GraphicDisplay/toolcore.h \

GraphicDisplay/vector.h \

dialogcreatefile.h \

mainmenu.h

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

DISTFILES +=

RESOURCES += \

resurses.qrc