**Файл “correctioncommand.h”**

#ifndef CORRECTIONCOMMAND\_H

#define CORRECTIONCOMMAND\_H

#include <QWidget>

#include <QImage>

#include <QUndoCommand>

namespace Draw

{

class CorrectionCommand : public QUndoCommand

{

public:

void undo() override;

void redo() override;

explicit CorrectionCommand(QWidget \*drawingArea, QImage \*image, double gamma);

~CorrectionCommand();

private:

void adjust();

private:

QWidget \*\_drawingArea;

QImage \*\_image;

QImage \_undoImage;

double \_gamma;

};

} // namespace Draw

#endif // CORRECTIONCOMMAND\_H

**Файл “correctioncommand.cpp”**

#include "CorrectionCommand.h"

#include <cmath>

namespace Draw

{

CorrectionCommand::CorrectionCommand(QWidget \*drawingArea,QImage \*image, double gamma)

: \_drawingArea(drawingArea),\_image(image), \_gamma(gamma)

{

this->\_gamma = (this->\_gamma > 0)? this->\_gamma : 1;

}

void CorrectionCommand::adjust()

{

int height = this->\_image->height();

int width = this->\_image->width();

for (int i = 0; i < height - 1; ++i)

{

QRgb \*line = reinterpret\_cast<QRgb\*>(this->\_image->scanLine(i));

for (int j = 0; j < width - 1; ++j)

{

QRgb pixelColor = line[j];

int r = qRed(pixelColor);

int g = qGreen(pixelColor);

int b = qBlue(pixelColor);

r = std::pow(double(r)/255, this->\_gamma) \* 255;

g = std::pow(double(g)/255, this->\_gamma) \* 255;

b = std::pow(double(b)/255, this->\_gamma) \* 255;

line[j] = qRgb(r,g,b);

}

}

}

void CorrectionCommand::undo()

{

\*this->\_image = this->\_undoImage.copy();

this->\_drawingArea->update();

}

void CorrectionCommand::redo()

{

this->\_undoImage = this->\_image->copy();

adjust();

this->\_drawingArea->update();

}

CorrectionCommand::~CorrectionCommand()

{

}

} // namespce Draw

**Файл “drawcommand.h”**

#ifndef DRAWCOMMAND\_H

#define DRAWCOMMAND\_H

#include <QUndoCommand>

#include <QWidget>

namespace Draw

{

class DrawCommand : public QUndoCommand

{

public:

virtual void undo() override;

virtual void redo() override;

explicit DrawCommand(QWidget \*drawingArea, QImage \*image, QImage \_afterDrawingImage, QImage \_beforeDrawingImage);

~DrawCommand();

private:

QWidget \*\_drawingArea;

QImage \*\_image;

QImage \_afterDrawingImage;

QImage \_beforeDrawingImage;

};

} // namespace Draw

#endif // DRAWCOMMAND\_H

**Файл “drawcommand.cpp”**

#include "DrawCommand.h"

namespace Draw

{

DrawCommand::DrawCommand(QWidget \*drawingArea, QImage \*image, QImage afterDrawingImage, QImage beforeDrawingImage) :

\_drawingArea(drawingArea),

\_image(image),

\_afterDrawingImage(afterDrawingImage),

\_beforeDrawingImage(beforeDrawingImage)

{

}

void DrawCommand::undo()

{

\*this->\_image = this->\_afterDrawingImage;

this->\_drawingArea->update();

}

void DrawCommand::redo()

{

\*this->\_image = this->\_beforeDrawingImage;

this->\_drawingArea->update();

}

DrawCommand::~DrawCommand() {}

} // namespace Draw

**Файл “drawingarea.h”**

#ifndef DRAW\_H

#define DRAW\_H

#include "Shape.h"

#include <qevent.h>

#include <QPainter>

#include <QUndoStack>

#include <QEvent>

#include <QWidget>

namespace Draw

{

class DrawingArea : public QWidget

{

Q\_OBJECT

public:

bool isModified();

void createNewImage();

bool openImage(const QString &fileName);

bool saveImage(const QString &fileName, const char \*fileFormat);

void setPenColor(const QColor &newColor);

void setPenWidth(int newWidth);

QColor getPenColor();

int getPenWidth();

QSize getImageSize();

void resizeIamge(const QSize newSize);

void flip(bool horizontal, bool vertical);

void rotate(qreal deg);

void toggleEraserMode();

void drawEllipses(QMouseEvent \*event);

void drawRectangle(QMouseEvent \*event);

void ColorPicker(QMouseEvent \*event);

void fillShape(QMouseEvent \*event);

void medianFilter(const int ratio);

void gammaCorrection(const double gamma);

void drawMouseLine(const QPoint &endPoint);

DrawingArea(QUndoStack \*undoStack, QWidget \*parent = 0);

~DrawingArea();

public slots:

void setCreatePen();

void setCreateEllipse();

void setCreateRectangle();

void setColorPicker();

void setCreateEraser();

void setCreateFilledShape();

protected:

void mousePressEvent(QMouseEvent \*event);

void mouseReleaseEvent(QMouseEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void paintEvent(QPaintEvent \*event);

private:

QUndoStack \*\_undoStack;

QImage \_image;

int \_penWidth;

QPoint lastPoint;

QColor \_penColor;

QColor \_oldPenColor;

std::string \_currentShape;

bool isDrawing;

QImage \_afterDrawingImage;

bool \_isEraser;

static const QSize \_startSize;

static const QSize \_maxSize;

};

} // namespace Draw

#endif //DRAW\_H

**Файл “drawingarea.cpp”**

#include "DrawingArea.h"

#include "CreateFunctions.h"

#include "ImageSizeException.h"

#include "DrawCommand.h"

#include "ResizeCommand.h"

#include "FlipCommand.h"

#include "CorrectionCommand.h"

#include "FiltraringCommand.h"

namespace Draw

{

const QSize DrawingArea::\_startSize(1920,1080);

const QSize DrawingArea::\_maxSize(5000,5000);

DrawingArea::DrawingArea(QUndoStack \*undoStack, QWidget \*parent) :

QWidget(parent),

\_undoStack(undoStack),

\_penWidth(5),

\_penColor(Qt::black),

\_currentShape("1")

{

const QImage \_image = QImage();

isDrawing = false;

ResizeCommand(this, &this->\_image, \_startSize).redo();

}

DrawingArea::~DrawingArea()

{

}

bool DrawingArea::isModified()

{

return !(this->\_undoStack->isClean());

}

void DrawingArea::resizeIamge(const QSize newSize)

{

this->\_undoStack->push(new ResizeCommand(this, &this->\_image,

newSize));

}

void DrawingArea::flip(bool horizontal, bool vertical)

{

this->\_undoStack->push(new FlipCommand(this, &this->\_image,

horizontal, vertical));

}

void DrawingArea::medianFilter(const int ratio)

{

this->\_undoStack->push(new FiltraringCommand(this, &this->\_image,

ratio));

}

void DrawingArea::gammaCorrection(const double gamma)

{

this->\_undoStack->push(new CorrectionCommand(this, &this->\_image,

gamma));

}

void DrawingArea::createNewImage()

{

ResizeCommand(this, &this->\_image, \_startSize).redo();

this->\_image.fill(Qt::white);

this->\_undoStack->clear();

}

bool DrawingArea::openImage(const QString &fileName)

{

QImage loadedImage;

if (!loadedImage.load(fileName))

return false;

const QSize newSize = loadedImage.size();

if(newSize.width() > this->\_maxSize.width() ||

newSize.height() > this->\_maxSize.height())

throw ImageSizeException("Bad image resolution!");

this->\_image = loadedImage.convertToFormat(QImage::Format\_ARGB32);

ResizeCommand(this, &this->\_image, newSize).redo();

this->\_undoStack->clear();

return true;

}

bool DrawingArea::saveImage(const QString &fileName, const char \*fileFormat)

{

if (this->\_image.save(fileName, fileFormat))

{

this->\_undoStack->clear();

return true;

}

else

return false;

}

void DrawingArea::setPenColor(const QColor &newColor)

{

this->\_penColor = newColor;

}

void DrawingArea::setPenWidth(int newWidth)

{

this->\_penWidth = newWidth;

}

void DrawingArea::setCreatePen()

{

this->\_currentShape = "Pen";

}

void DrawingArea::setCreateEllipse()

{

this->\_currentShape = "Ellipse";

}

void DrawingArea::setCreateRectangle()

{

this->\_currentShape = "Rectangle";

}

void DrawingArea::setColorPicker()

{

this->\_currentShape = "ColorPicker";

}

void DrawingArea::setCreateEraser()

{

this->\_currentShape = "Eraser";

this->\_oldPenColor = this->\_penColor;

this->\_penColor = Qt::white;

this->\_isEraser = true;

}

void DrawingArea::setCreateFilledShape()

{

this->\_currentShape = "FilledShape";

}

QColor DrawingArea::getPenColor()

{

return this->\_penColor;

}

int DrawingArea::getPenWidth()

{

return this->\_penWidth;

}

QSize DrawingArea::getImageSize()

{

return this->\_image.size();

}

void DrawingArea::toggleEraserMode()

{

if(this->\_isEraser)

{

this->\_penColor = this->\_oldPenColor;

this->\_isEraser = false;

}

}

void DrawingArea::drawEllipses(QMouseEvent \*event)

{

toggleEraserMode();

QPainter painter(&this->\_image);

Ellipse ellipse(&this->\_image, event->pos(), this->getPenWidth(), this->getPenColor());

ellipse.draw(painter);

update();

}

void DrawingArea::drawRectangle(QMouseEvent \*event)

{

toggleEraserMode();

QPainter painter(&this->\_image);

Rectangle rect(&this->\_image, event->pos(), this->getPenWidth(), this->getPenColor());

rect.draw(painter);

update();

}

void DrawingArea::ColorPicker(QMouseEvent \*event)

{

this->\_penColor = this->\_image.pixelColor(event->pos());

}

void DrawingArea::fillShape(QMouseEvent \*event)

{

toggleEraserMode();

FilledShape fillShape(&this->\_image, event->pos(), this->\_penColor);

QPainter painter(&this->\_image);

fillShape.draw(painter);

update();

}

void DrawingArea::mousePressEvent(QMouseEvent \*event)

{

if (event->button() == Qt::LeftButton)

{

lastPoint = event->pos();

this->isDrawing = true;

\_afterDrawingImage = \_image.copy();

}

}

void DrawingArea::drawMouseLine(const QPoint &endPoint)

{

if(this->\_currentShape == "Pen")

{

toggleEraserMode();

}

QPainter painter(&this->\_image);

painter.setPen(QPen(\_penColor, \_penWidth, Qt::SolidLine, Qt::RoundCap,

Qt::RoundJoin));

painter.drawLine(lastPoint, endPoint);

int rad = (\_penWidth / 2) + 2;

update(QRect(lastPoint, endPoint).normalized()

.adjusted(-rad, -rad, +rad, +rad));

lastPoint = endPoint;

}

void DrawingArea::mouseMoveEvent(QMouseEvent \*event)

{

if(this->isDrawing & this->\_currentShape == "Pen" || this->\_currentShape == "Eraser")

{

drawMouseLine(event->pos());

}

}

void DrawingArea::mouseReleaseEvent(QMouseEvent \*event)

{

if(event->button() == Qt::LeftButton && this->isDrawing) {

this->isDrawing = false;

lastPoint = event->pos();

update();

this->\_undoStack->push(new DrawCommand(this, &this->\_image, \_afterDrawingImage, \_image.copy()));

if(this->\_currentShape == "Ellipse")

{

return this->drawEllipses(event);

}

if(this->\_currentShape == "Rectangle")

{

return this->drawRectangle(event);

}

if(this->\_currentShape == "ColorPicker")

{

return this->ColorPicker(event);

}

if(this->\_currentShape == "FilledShape")

{

return this->fillShape(event);

}

}

}

void DrawingArea::paintEvent(QPaintEvent \*event)

{

const QRect paintRect = event->rect();

QPainter painter(this);

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

painter.drawImage(paintRect, this->\_image, paintRect);

}

} // namespace Draw

**Файл “ellipse.h”**

#ifndef ELLIPSE\_H

#define ELLIPSE\_H

#include "Shape.h"

namespace Draw

{

class Ellipse : public Shape

{

public:

void draw(QPainter &painter) override;

void update(const QPoint &toPoint) override;

QRect rect() override;

Ellipse(QImage \*image,

const QPoint &eventPoint,

int penWidth,

const QColor& penColor);

~Ellipse();

private:

QRect \_rectangle;

};

} // namespace Draw

#endif // ELLIPSE\_H

**Файл “ellipse.cpp”**

#include "Ellipse.h"

namespace Draw

{

Ellipse::Ellipse(QImage \*image, const QPoint &eventPoint, int penWidth, const QColor &penColor) :

Shape(image, penWidth, penColor), \_rectangle(eventPoint.x(), eventPoint.y(), 100 ,100)

{

}

Ellipse::~Ellipse()

{

}

void Ellipse::draw(QPainter &painter)

{

if (!this->\_rectangle.isNull())

{

painter.setPen(QPen(this->getPenColor(), this->getPenWidth(),

Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter.drawEllipse(this->\_rectangle);

}

}

void Ellipse::update(const QPoint &toPoint)

{

this->\_rectangle.setBottomRight(toPoint);

}

QRect Ellipse::rect()

{

const int correction = this->getPenWidth() / 2 + 2;

QRect shapeRect = this->\_rectangle.normalized();

shapeRect = shapeRect.adjusted(-correction, -correction,

+correction, +correction);

return shapeRect;

}

} // namespace Draw

**Файл “filledshape.h”**

#ifndef FILLEDSHAPE\_H

#define FILLEDSHAPE\_H

#include "Shape.h"

namespace Draw {

class FilledShape : public Shape

{

public:

void draw(QPainter &painter) override;

void update(const QPoint &toPoint) override;

QRect rect() override;

FilledShape(QImage\* image,

const QPoint &eventPoint,

const QColor& penColor);

~FilledShape();

private:

std::vector<QPoint> floodFill(const QPoint &pos, const QRgb &newColor);

private:

std::vector<QPoint> \_points;

QRect \_rectangle;

};

} // namespace Draw

#endif // FILLEDSHAPE\_H

**Файл “filledshape.cpp”**

#include "FilledShape.h"

#include <queue>

#include <QPoint>

#include <vector>

namespace Draw

{

std::vector<QPoint>

FilledShape::floodFill(const QPoint &pos, const QRgb &newColor)

{

std::vector<QPoint> modified;

QImage image = this->\_image->copy();

QRgb oldColor = image.pixel(pos);

if (oldColor == newColor)

{

return modified;

}

std::queue<QPoint> nodeQ;

nodeQ.push(QPoint(pos.x(), pos.y()));

while(!nodeQ.empty())

{

QPoint currNode = nodeQ.front();

nodeQ.pop();

if(image.pixel(currNode) == oldColor)

{

image.setPixel(currNode.x(),currNode.y(), newColor);

modified.emplace\_back(currNode);

if(currNode.x() > 0)

nodeQ.push(QPoint(currNode.x()-1, currNode.y()));

if(currNode.x() < (image.width() - 1))

nodeQ.push(QPoint(currNode.x()+1, currNode.y()));

if(currNode.y() > 0)

nodeQ.push(QPoint(currNode.x(), currNode.y()-1));

if(currNode.y() < (image.height() - 1))

nodeQ.push(QPoint(currNode.x(), currNode.y()+1));

}

}

return modified;

}

FilledShape::FilledShape(QImage\* image, const QPoint &topLeft, const QColor &penColor) :

Shape(image, 1, penColor),

\_points(FilledShape::floodFill(topLeft, penColor.rgb())),

\_rectangle(topLeft, topLeft)

{

foreach (const QPoint point, this->\_points)

{

if (point.x() < this->\_rectangle.left())

this->\_rectangle.setLeft(point.x());

else if (point.x() > this->\_rectangle.right())

this->\_rectangle.setRight(point.x());

if (point.y() < this->\_rectangle.top())

this->\_rectangle.setTop(point.y());

else if (point.y() > this->\_rectangle.bottom())

this->\_rectangle.setBottom(point.y());

}

}

void FilledShape::draw(QPainter &painter)

{

if (!this->\_rectangle.isNull())

{

painter.setPen(QPen(this->getPenColor(), this->getPenWidth(),

Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter.drawPoints(this->\_points.data(), this->\_points.size());

}

}

QRect FilledShape::rect()

{

const int correction = this->getPenWidth() / 2 + 2;

QRect shapeRect = this->\_rectangle.normalized();

shapeRect = shapeRect.adjusted(-correction, -correction,

+correction, +correction);

return shapeRect;

}

void FilledShape::update(const QPoint &)

{

}

FilledShape::~FilledShape()

{

}

} // namespace Draw

**Файл “filtraringcommand.h”**

#ifndef FILTRARINGCOMMAND\_H

#define FILTRARINGCOMMAND\_H

#include <QWidget>

#include <QImage>

#include <QUndoCommand>

namespace Draw

{

class FiltraringCommand : public QUndoCommand

{

public:

void undo() override;

void redo() override;

explicit FiltraringCommand(QWidget \*drawingArea, QImage \*image, const int ratio);

~FiltraringCommand();

private:

int findMedian(int\* Array, const int size);

void filtrate();

private:

QWidget \*\_drawingArea;

QImage \*\_image;

QImage \_undoImage;

int \_ratio;

};

} // namespace Draw

#endif // FILTRARINGCOMMAND\_H

**Файл “filtraringcommand.cpp”**

#include "FiltraringCommand.h"

#include <QProgressDialog>

#include <QApplication>

namespace Draw

{

FiltraringCommand::FiltraringCommand(QWidget \*drawingArea, QImage \*image, const int ratio):

\_drawingArea(drawingArea), \_image(image), \_ratio(ratio)

{

}

int FiltraringCommand::findMedian(int\* Array, const int size)

{

int i;

int j;

int median\_index;

int temp;

int median;

for (i = 1; i < size; i++)

{

j = i - 1;

temp = Array[i];

while (j >= 0 && temp < Array[j])

{

Array[j + 1] = Array[j];

j--;

}

Array[j + 1] = temp;

}

median\_index = (int)(size / 2);

median = Array[median\_index];

return median;

}

void FiltraringCommand::filtrate()

{

const int array\_size = 9;

int row = 0;

int coloumn = 0;

int size = 0;

int row\_limit = 3;

int coloumn\_limit = 3;

int red\_array[array\_size];

int green\_array[array\_size];

int blue\_array[array\_size];

for (int i = 0; i < this->\_image->height(); i++)

{

for (int j = 0; j < this->\_image->width(); j++)

{

row = i - 1;

row\_limit = 3;

if (i == this->\_image->height() - 1 || i == 0)

{

row\_limit = 2;

if(i == 0) row = i;

}

for (int y = row, k = 0; row\_limit--; y++)

{

coloumn = j - 1;

coloumn\_limit = 3;

if (j == this->\_image->width() - 1 || j == 0)

{

coloumn\_limit = 2;

if (j == 0) coloumn = j;

}

for (int x = coloumn; coloumn\_limit--; x++)

{

red\_array[k] = qRed(this->\_image->pixel(x, y));

green\_array[k] = qGreen(this->\_image->pixel(x, y));

blue\_array[k] = qBlue(this->\_image->pixel(x, y));

k++;

size = k;

}

}

int r = findMedian(red\_array, size);

int g = findMedian(green\_array, size);

int b = findMedian(blue\_array, size);

QRgb pixelColor = qRgb(r, g ,b);

this->\_image->setPixel(j, i, pixelColor);

memset(red\_array, 0, array\_size \* sizeof(int));

memset(green\_array, 0, array\_size \* sizeof(int));

memset(blue\_array, 0, array\_size \* sizeof(int));

size = 0;

}

}

}

void FiltraringCommand::undo()

{

\*this->\_image = this->\_undoImage.copy();

this->\_drawingArea->update();

}

void FiltraringCommand::redo()

{

this->\_undoImage = this->\_image->copy();

QProgressDialog \*progressDialog = new QProgressDialog("Filtrating...",

QString(),

0, this->\_ratio,

this->\_drawingArea,

Qt::WindowTitleHint);

progressDialog->setWindowTitle("Please Wait");

progressDialog->setWindowModality(Qt::WindowModal);

progressDialog->setMinimumDuration(0);

progressDialog->setMinimumHeight(70);

progressDialog->setMinimumWidth(250);

progressDialog->setValue(0);

QApplication::processEvents();

for(int i = 0; i < this->\_ratio; i++)

{

progressDialog->setValue(i);

QApplication::processEvents();

filtrate();

}

progressDialog->setValue(this->\_ratio);

delete progressDialog;

this->\_drawingArea->update();

}

FiltraringCommand::~FiltraringCommand()

{

}

} // namespace Draw

**Файл “flipcommand.h”**

#ifndef FLIPCOMMAND\_H

#define FLIPCOMMAND\_H

#include <QUndoCommand>

#include <QWidget>

namespace Draw

{

class FlipCommand : public QUndoCommand

{

public:

virtual void undo() override;

virtual void redo() override;

explicit FlipCommand(QWidget \*drawingArea, QImage \*image,

bool horizontal, bool vertical);

~FlipCommand();

private:

QWidget \*\_drawingArea;

QImage \*\_image;

bool \_horizontal;

bool \_vertical;

};

} // namespace Draw

#endif // FLIPCOMMAND\_H

**Файл “flipcommand.cpp”**

#include "FlipCommand.h"

namespace Draw

{

FlipCommand::FlipCommand(QWidget \*drawingArea, QImage \*image,

bool horizontal, bool vertical) :

\_drawingArea(drawingArea), \_image(image),

\_horizontal(horizontal), \_vertical(vertical)

{

}

void FlipCommand::undo()

{

redo();

}

void FlipCommand::redo()

{

\*this->\_image = this->\_image->mirrored(this->\_vertical, this->\_horizontal);

this->\_drawingArea->update();

}

FlipCommand::~FlipCommand()

{

}

} // namespace Draw

**Файл “imagesizeexception.h”**

#ifndef IMAGESIZEEXCEPTION\_H

#define IMAGESIZEEXCEPTION\_H

#include <stdexcept>

namespace Draw

{

class ImageSizeException : public std::domain\_error

{

public:

explicit ImageSizeException(const char\* message);

~ImageSizeException();

};

} // namespace Draw

#endif // IMAGESIZEEXCEPTION\_H

**Файл “imagesizeexception.cpp”**

#include "ImageSizeException.h"

namespace Draw

{

ImageSizeException::ImageSizeException(const char\* message) : domain\_error(message)

{

}

ImageSizeException::~ImageSizeException()

{

}

} // namespace Draw

**Файл “mainwindow.h”**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "DrawingArea.h"

#include <QActionGroup>

#include <QMainWindow>

#include <QScrollArea>

#include <QUndoStack>

#include <QList>

namespace Draw

{

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = 0);

~MainWindow();

protected:

void closeEvent(QCloseEvent \*event);

private slots:

void open();

void save();

void createNew();

void penColor();

void penWidth();

void flipHorizontal();

void flipVertical();

void changeSize();

void filtrate();

void correction();

void about();

private:

void createDrawActionGruop();

void createEffectsActionGruop();

void createActions();

void createMenus();

QToolBar\* createToolBar();

bool maybeSave();

bool saveFile(const QByteArray &fileFormat);

QMenu \*\_saveAsMenu;

QMenu \*\_fileMenu;

QMenu \*\_doRedoMenu;

QMenu \*\_brushMenu;

QMenu \*\_effectsMenu;

QMenu \*\_addOptionsMenu;

QMenu \*\_helpMenu;

QToolBar\* \_drawToolBar;

QAction \*\_openAct;

QList<QAction \*> \_saveAsActs;

QAction \*\_exitAct;

QAction \*\_newAct;

QAction \*\_undoAct;

QAction \*\_redoAct;

QAction \*\_penColorAct;

QAction \*\_penWidthAct;

QAction \*\_resizeAct;

QAction \*\_correctionAct;

QAction \*\_filtrateAct;

QAction \*\_aboutAct;

QAction \*\_drawPen;

QActionGroup \_drawActionGroup;

QActionGroup \_effectsActionGroup;

QUndoStack \_undoStack;

DrawingArea \*\_workingSpace;

QScrollArea \*\_scrollingArea;

};

} // namespace Draw

#endif // MAINWINDOW\_H

**Файл “mainwindow.cpp”**

#include "mainwindow.h"

#include "ImageSizeException.h"

#include "DrawingArea.h"

#include <QWidget>

#include <QtWidgets>

namespace Draw

{

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), \_drawActionGroup(this), \_effectsActionGroup(this)

{

this->\_workingSpace = new DrawingArea(&this->\_undoStack);

this->\_workingSpace->setMinimumSize(100,100);

createActions();

createMenus();

this->\_drawToolBar = createToolBar();

addToolBar(Qt::TopToolBarArea, this->\_drawToolBar);

setWindowIcon(QIcon("D:/qt projects/paintCourse/icons/MainIcon.ico"));

this->\_scrollingArea = new QScrollArea;

this->\_scrollingArea->setWidget(this->\_workingSpace);

setCentralWidget(this->\_scrollingArea);

setWindowTitle("Lightning Paint");

resize(500, 500);

this->\_drawActionGroup.actions().first()->trigger();

}

MainWindow::~MainWindow()

{

delete this->\_drawToolBar;

delete this->\_newAct;

delete this->\_openAct;

delete this->\_exitAct;

delete this->\_undoAct;

delete this->\_redoAct;

delete this->\_penColorAct;

delete this->\_penWidthAct;

delete this->\_resizeAct;

delete this->\_correctionAct;

delete this->\_filtrateAct;

delete this->\_aboutAct;

delete this->\_drawPen;

foreach (QAction \*action, \_saveAsActs)

delete action;

foreach (QAction \*action, \_drawActionGroup.actions())

delete action;

foreach (QAction \*action, \_effectsActionGroup.actions())

delete action;

delete this->\_saveAsMenu;

delete this->\_fileMenu;

delete this->\_doRedoMenu;

delete this->\_brushMenu;

delete this->\_effectsMenu;

delete this->\_addOptionsMenu;

delete this->\_helpMenu;

delete this->\_workingSpace;

delete this->\_scrollingArea;

}

void MainWindow::closeEvent(QCloseEvent \*event)

{

if (maybeSave())

event->accept();

else

event->ignore();

}

void MainWindow::createNew()

{

if (maybeSave())

this->\_workingSpace->createNewImage();

}

void MainWindow::open()

{

if (maybeSave())

{

try

{

const QString fileName =

QFileDialog::getOpenFileName(this, "Open File",

QDir::currentPath());

if (!fileName.isEmpty())

this->\_workingSpace->openImage(fileName);

}

catch (ImageSizeException &excepetion)

{

QString message;

message.insert(message.size(), excepetion.what());

message.insert(message.size(), "\nImage's resolution can't be bigger than 5000x5000");

QMessageBox::warning(this, "Lightning Paint",

message,

QMessageBox::Ok);

}

}

}

void MainWindow::save()

{

QAction \*action = qobject\_cast<QAction \*>(sender());

QByteArray fileFormat = action->data().toByteArray();

saveFile(fileFormat);

}

void MainWindow::penColor()

{

const QColor newColor = QColorDialog::getColor(this->\_workingSpace->getPenColor());

if (newColor.isValid())

this->\_workingSpace->setPenColor(newColor);

}

void MainWindow::penWidth()

{

bool ok;

const int newWidth = QInputDialog::getInt(this, "Pen's options",

"Select pen width:",

this->\_workingSpace->getPenWidth(),

1, 50, 1, &ok,

Qt::WindowTitleHint | Qt::WindowCloseButtonHint);

if (ok)

this->\_workingSpace->setPenWidth(newWidth);

}

void MainWindow::flipHorizontal()

{

this->\_workingSpace->flip(true, false);

}

void MainWindow::flipVertical()

{

this->\_workingSpace->flip(false, true);

}

void MainWindow::changeSize()

{

bool ok\_1, ok\_2;

const int newWidth = QInputDialog::getInt(this, "Resize",

"Input image's width:",

this->\_workingSpace->getImageSize().width(),

50, 5000, 10, &ok\_1,

Qt::WindowTitleHint | Qt::WindowCloseButtonHint);

if(ok\_1)

{

const int newHeight = QInputDialog::getInt(this, "Resize",

"Input image's height:",

this->\_workingSpace->getImageSize().height(),

50, 5000, 10, &ok\_2,

Qt::WindowTitleHint | Qt::WindowCloseButtonHint);

if (ok\_2)

{

const QSize newSize(newWidth, newHeight);

this->\_workingSpace->resizeIamge(newSize);

}

}

}

void MainWindow::correction()

{

bool ok;

const double gamma = QInputDialog::getDouble(this, "Gamma Correction",

"Input required gamma valure",

1.0,

0.05, 50.0, 2, &ok,

Qt::WindowTitleHint | Qt::WindowCloseButtonHint);

if (ok)

this->\_workingSpace->gammaCorrection(gamma);

}

void MainWindow::filtrate()

{

bool ok;

const int ratio = QInputDialog::getInt(this, "Median Filtrating",

"Select filtrating ratio:",

1, 1, 20, 1, &ok,

Qt::WindowTitleHint | Qt::WindowCloseButtonHint);

if (ok)

this->\_workingSpace->medianFilter(ratio);

}

void MainWindow::about()

{

QMessageBox::about(this, "About Lightning Paint",

"Lightning Paint:)");

}

bool MainWindow::maybeSave()

{

if (this->\_workingSpace->isModified())

{

const QMessageBox::StandardButton clicked =

QMessageBox::warning(this, "Lightning Paint",

"The image has been modified.\n"

"Do you want to save your changes?",

QMessageBox::Save |

QMessageBox::Discard |

QMessageBox::Cancel);

if (clicked == QMessageBox::Save)

return saveFile("png");

else if (clicked == QMessageBox::Cancel)

return false;

}

return true;

}

void MainWindow::createDrawActionGruop()

{

QAction \*penAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/pen.ico"), "&Pen");

connect(penAct, SIGNAL(triggered()), \_workingSpace, SLOT(setCreatePen()));

penAct->setCheckable(true);

penAct->setActionGroup(&\_drawActionGroup);

QAction \*ellipseAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/ellipse.ico"), "&Ellipse");

connect(ellipseAct, SIGNAL(triggered()), \_workingSpace, SLOT(setCreateEllipse()));

ellipseAct->setCheckable(true);

ellipseAct->setActionGroup(&\_drawActionGroup);

QAction \*rectangleAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/rectangle.ico"), "&Rectangle");

connect(rectangleAct, SIGNAL(triggered()), \_workingSpace, SLOT(setCreateRectangle()));

rectangleAct->setCheckable(true);

rectangleAct->setActionGroup(&\_drawActionGroup);

QAction \*lineAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/ColorPicker.ico"), "&ColorPicker");

connect(lineAct, SIGNAL(triggered()), \_workingSpace, SLOT(setColorPicker()));

lineAct->setCheckable(true);

lineAct->setActionGroup(&\_drawActionGroup);

QAction \*eraserAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/eraser.ico"), "&Eraser");

connect(eraserAct, SIGNAL(triggered()), \_workingSpace, SLOT(setCreateEraser()));

eraserAct->setCheckable(true);

eraserAct->setActionGroup(&\_drawActionGroup);

QAction \*fillAct = new QAction(QIcon("D:/qt projects/paintCourse/icons/fill.ico"), "&Fill");

connect(fillAct, SIGNAL(triggered()), \_workingSpace, SLOT(setCreateFilledShape()));

fillAct->setCheckable(true);

fillAct->setActionGroup(&\_drawActionGroup);

\_drawActionGroup.setExclusive(true);

}

void MainWindow::createEffectsActionGruop()

{

QAction \*flipHorizontalAct = new QAction("Flip Horizontal");

connect(flipHorizontalAct, SIGNAL(triggered()), this, SLOT(flipHorizontal()));

flipHorizontalAct->setActionGroup(&\_effectsActionGroup);

QAction \*flipVerticalAct = new QAction("Flip Vertical");

connect(flipVerticalAct, SIGNAL(triggered()), this, SLOT(flipVertical()));

flipVerticalAct->setActionGroup(&\_effectsActionGroup);

\_effectsActionGroup.setExclusive(true);

}

void MainWindow::createActions()

{

\_newAct = new QAction("Create New...", this);

\_newAct->setShortcuts(QKeySequence::New);

connect(\_newAct, SIGNAL(triggered()), this, SLOT(createNew()));

\_openAct = new QAction("&Open...", this);

\_openAct->setShortcuts(QKeySequence::Open);

connect(\_openAct, SIGNAL(triggered()), this, SLOT(open()));

foreach (QByteArray format, QImageWriter::supportedImageFormats())

{

QString text = tr("%1...").arg(QString(format).toUpper());

QAction \*action = new QAction(text, this);

action->setData(format);

connect(action, SIGNAL(triggered()), this, SLOT(save()));

\_saveAsActs.append(action);

}

\_exitAct = new QAction("E&xit", this);

\_exitAct->setShortcuts(QKeySequence::Quit);

connect(\_exitAct, SIGNAL(triggered()), this, SLOT(close()));

\_undoAct = \_undoStack.createUndoAction(nullptr, "&Undo");

\_undoAct->setShortcut(QKeySequence::Undo);

\_redoAct = \_undoStack.createRedoAction(nullptr, "&Redo");

\_redoAct->setShortcut(QKeySequence::Redo);

\_penColorAct = new QAction("&Pen Color...", this);

connect(\_penColorAct, SIGNAL(triggered()), this, SLOT(penColor()));

\_penWidthAct = new QAction("Pen &Width...", this);

connect(\_penWidthAct, SIGNAL(triggered()), this, SLOT(penWidth()));

\_resizeAct = new QAction("&Resize image...", this);

connect(\_resizeAct, SIGNAL(triggered()), this, SLOT(changeSize()));

\_correctionAct = new QAction("&Gamma Correction", this);

connect(\_correctionAct, SIGNAL(triggered()), this, SLOT(correction()));

\_filtrateAct = new QAction("&Median Filtration", this);

connect(\_filtrateAct, SIGNAL(triggered()), this, SLOT(filtrate()));

createDrawActionGruop();

createEffectsActionGruop();

\_aboutAct = new QAction("&About", this);

connect(\_aboutAct, SIGNAL(triggered()), this, SLOT(about()));

}

QToolBar\* MainWindow::createToolBar()

{

QToolBar\* toolBar = new QToolBar("Drawing ToolBar");

toolBar->addActions(this->\_drawActionGroup.actions());

toolBar->addSeparator();

toolBar->addAction(this->\_penWidthAct);

toolBar->addAction(this->\_penColorAct);

toolBar->setIconSize(QSize(25,25));

toolBar->setMovable(false);

return toolBar;

}

void MainWindow::createMenus()

{

\_saveAsMenu = new QMenu("&Save As", this);

foreach (QAction \*action, \_saveAsActs)

\_saveAsMenu->addAction(action);

\_fileMenu = new QMenu("&File", this);

\_fileMenu->addAction(\_newAct);

\_fileMenu->addAction(\_openAct);

\_fileMenu->addMenu(\_saveAsMenu);

\_fileMenu->addSeparator();

\_fileMenu->addAction(\_exitAct);

\_doRedoMenu = new QMenu("&Edit", this);

\_doRedoMenu->addAction(\_undoAct);

\_doRedoMenu->addAction(\_redoAct);

\_brushMenu = new QMenu("&Brush", this);

\_brushMenu->addAction(\_penColorAct);

\_brushMenu->addAction(\_penWidthAct);

\_brushMenu->addSeparator();

foreach (QAction \*action, \_drawActionGroup.actions())

\_brushMenu->addAction(action);

\_effectsMenu = new QMenu("&Effects");

foreach (QAction \*action, \_effectsActionGroup.actions())

\_effectsMenu->addAction(action);

\_addOptionsMenu = new QMenu("&Advanced Image Processing", this);

\_addOptionsMenu->addAction(\_resizeAct);

\_addOptionsMenu->addAction(\_correctionAct);

\_addOptionsMenu->addAction(\_filtrateAct);

\_helpMenu = new QMenu("&Help", this);

\_helpMenu->addAction(\_aboutAct);

menuBar()->addMenu(\_fileMenu);

menuBar()->addMenu(\_doRedoMenu);

menuBar()->addMenu(\_brushMenu);

menuBar()->addMenu(\_effectsMenu);

menuBar()->addMenu(\_addOptionsMenu);

menuBar()->addMenu(\_helpMenu);

}

bool MainWindow::saveFile(const QByteArray &fileFormat)

{

const QString initialPath = QDir::currentPath() + "/untitled." + fileFormat;

const QString fileName =

QFileDialog::getSaveFileName(this, "Save As",

initialPath,

tr("%1 Files (\*.%2);;All Files (\*)")

.arg(QString::fromLatin1(

fileFormat.toUpper()))

.arg(QString::fromLatin1(fileFormat)));

if (fileName.isEmpty())

return false;

else

return this->\_workingSpace->saveImage(fileName, fileFormat.constData());

}

} // namespace Draw

**Файл “rectangle.h”**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include "Shape.h"

namespace Draw

{

class Rectangle : public Shape

{

public:

void draw(QPainter &painter) override;

void update(const QPoint &toPoint) override;

QRect rect() override;

Rectangle(QImage \*image,

const QPoint &eventPoint,

int penWidth,

const QColor &penColor);

~Rectangle();

private:

QRect \_rectangle;

};

} // namespace Draw

#endif // RECTANGLE\_H

**Файл “rectangle.cpp”**

#include "Rectangle.h"

namespace Draw

{

Rectangle::Rectangle(QImage \*image,const QPoint &topLeft,

int penWidth, const QColor &penColor) :

Shape(image ,penWidth, penColor), \_rectangle(topLeft.x(), topLeft.y(), 25, 25)

{

}

Rectangle::~Rectangle()

{

}

void Rectangle::draw(QPainter &painter)

{

if (!this->\_rectangle.isNull())

{

painter.setPen(QPen(this->getPenColor(), this->getPenWidth(),

Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter.drawRect(this->\_rectangle);

}

}

void Rectangle::update(const QPoint &toPoint)

{

this->\_rectangle.setBottomRight(toPoint);

}

QRect Rectangle::rect()

{

const int correction = (this->getPenWidth() / 2) + 2;

QRect shapeRect = this->\_rectangle.normalized();

shapeRect = shapeRect.adjusted(-correction, -correction,

+correction, +correction);

return shapeRect;

}

} // namespace Draw

**Файл “resizecommand.h”**

#ifndef RESIZECOMMAND\_H

#define RESIZECOMMAND\_H

#include "Shape.h"

#include <QUndoCommand>

#include <QWidget>

namespace Draw

{

class ResizeCommand : public QUndoCommand

{

public:

virtual void undo() override;

virtual void redo() override;

ResizeCommand(QWidget \*drawingArea, QImage \*image,

const QSize &size);

~ResizeCommand();

private:

QWidget \*\_drawingArea;

QImage \*\_image;

QImage \_undoImage;

QSize \_oldSize;

QSize \_newSize;

};

} // namespace Draw

#endif // RESIZECOMMAND\_H

**Файл “resizecommand.cpp”**

#include "ResizeCommand.h"

namespace Draw

{

ResizeCommand::ResizeCommand(QWidget \*drawingArea,

QImage \*image,

const QSize &size) :

\_drawingArea(drawingArea), \_image(image),

\_oldSize(image->size()), \_newSize(size)

{

}

void ResizeCommand::undo()

{

\*this->\_image = this->\_undoImage.copy();

this->\_drawingArea->setGeometry(this->\_image->rect());

this->\_drawingArea->update();

}

void ResizeCommand::redo()

{

this->\_undoImage = this->\_image->copy();

if (this->\_image->size() == this->\_newSize)

{

this->\_drawingArea->setGeometry(this->\_image->rect());

this->\_drawingArea->update();

return;

}

QImage newImage(this->\_newSize, QImage::Format\_ARGB32);

newImage.fill(Qt::white);

\*this->\_image = this->\_image->scaled(this->\_newSize);

QPainter painter(&newImage);

painter.drawImage(QPoint(0, 0), \*this->\_image);

\*this->\_image = newImage;

this->\_drawingArea->setGeometry(this->\_image->rect());

this->\_drawingArea->update();

}

ResizeCommand::~ResizeCommand()

{

}

} // namespace Draw

**Файл “shape.h”**

#ifndef Shape\_H

#define Shape\_H

#include <QPainter>

namespace Draw

{

class Shape

{

public:

typedef Shape\* ShapePointer;

virtual void draw(QPainter &painter) = 0;

virtual void update(const QPoint &toPoint) = 0;

virtual QRect rect() = 0;

virtual ~Shape();

protected:

Shape(QImage\* image, int penWidth, const QColor &penColor);

int getPenWidth();

QColor getPenColor();

protected:

QImage\* \_image;

private:

int \_penWidth;

QColor \_penColor;

};

} // namespace Draw

#endif // Shape\_H

**Файл “shape.cpp”**

#include "Shape.h"

namespace Draw

{

Shape::Shape(QImage\* image,int penWidth, const QColor& penColor):

\_image(image),

\_penWidth(penWidth),

\_penColor(penColor)

{

}

int Shape::getPenWidth()

{

return this->\_penWidth;

}

QColor Shape::getPenColor()

{

return this->\_penColor;

}

Shape::~Shape()

{

}

} // namespace Draw

**Файл “main.cpp”**

#include "MainWindow.h"

#include <QApplication>

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

{

QApplication app(argc, argv);

Draw::MainWindow window;

window.show();

return app.exec();

}