**Министерство образования и науки Российской Федерации**

**Федеральное государственное образовательное учреждение высшего профессионального образования**

**«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИСТЕТ»**

**(ВолгГТУ)**

**Факультет Электроники и вычислительной техники**

**Кафедра ««САПР и ПК»»**

# Курсовая работа

На тему: «Графический редактор».

**Выполнил: студент группы ФЭВТ 3С**

**Кравченко А.А.**

**Проверил преподаватель:**

**Шабалина О.А.**

**Волгоград, 2013г.**

**Содержание**

[Постановка задачи 2](#_Toc375498364)

[Функции 2](#_Toc375498365)

[Проектирование интерфейса (с точки пользователя) 2](#_Toc375498366)

[Проектирование интерфейса (с точки разработчика) 3](#_Toc375498367)

[Тестовый пример 4](#_Toc375498368)

[Листинг 4](#_Toc375498369)

[Диаграмма вариантов использования 40](#_Toc375498370)

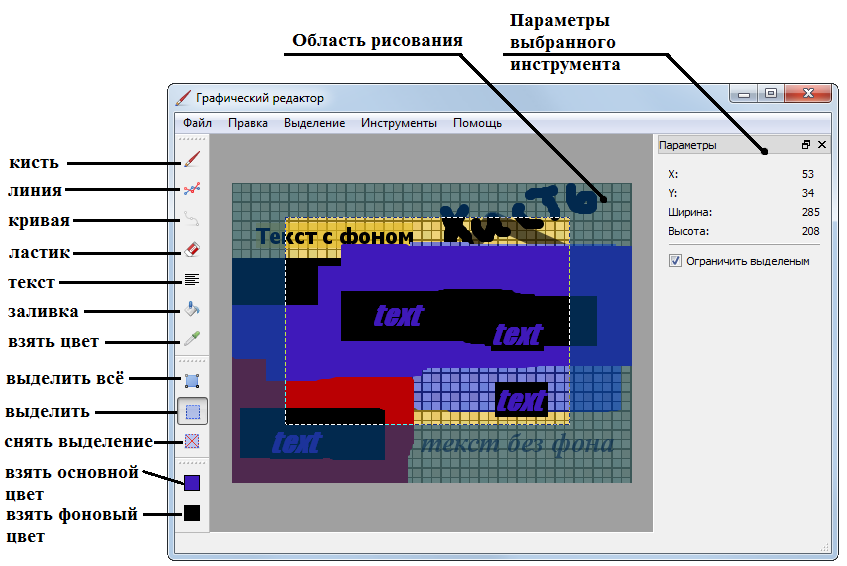
# Постановка задачи

Реализовать программу «Графический редактор».

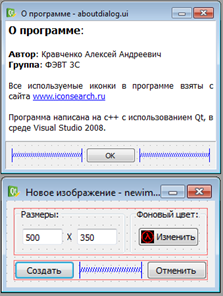
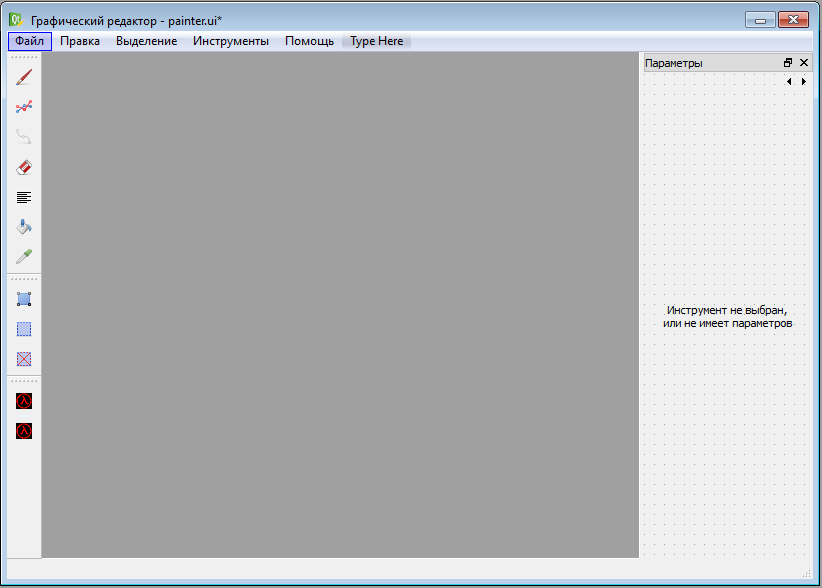
# Функции

1. Создание изображений;
2. Открытие изображений;
3. Сахранение изображений;
4. Печать изображений;
5. Вставка и копирование изображений (буфер обмена)
6. Выделение области
7. Поддержка инструментов:
   1. Кисть
   2. Линия
   3. Кривая
   4. Ластик
   5. Заливка
   6. Текст
   7. Взять цвет

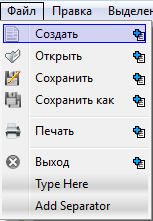
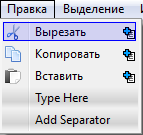
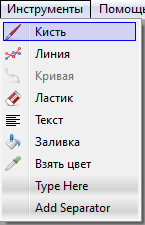
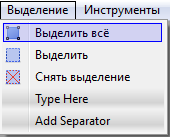
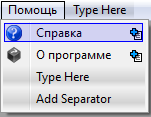
# Проектирование интерфейса (с точки пользователя)



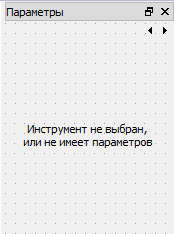
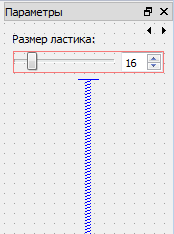
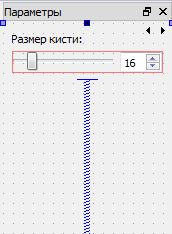
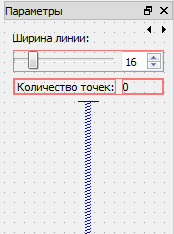
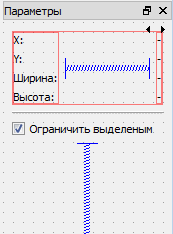
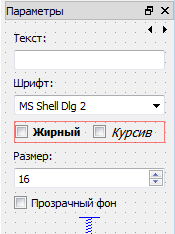
# Проектирование интерфейса (с точки разработчика)



***Рисунок - Основные формы***

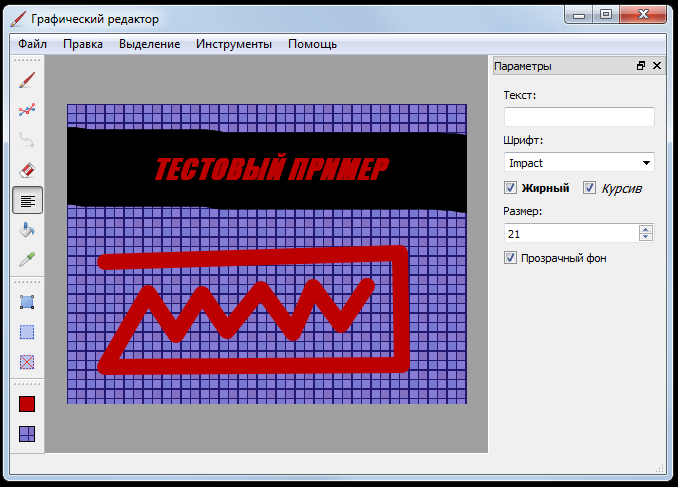
    

***Рисунок - Главное меню***

***Рисунок - Стек виджет***

# Тестовый пример



# Листинг

***main.cpp***

#include "stdafx.h"

#include "painter.h"

#include <QtGui/QApplication>

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

{

QTextCodec::setCodecForCStrings(QTextCodec::codecForName("Windows-1251"));

QApplication a(argc, argv);

//Указываем деректорию для плагинов

QStringList list\_path;

QDir dir =QDir(a.applicationDirPath()+"//plugins//");

list\_path << dir.absolutePath() << a.libraryPaths();

a.setLibraryPaths(list\_path);

//---------------------------------

Painter w;

w.show();

return a.exec();

}

***painter.h***

#ifndef PAINTER\_H

#define PAINTER\_H

#include <QtGui/QMainWindow>

#include <QActionGroup>

#include <QClipboard>

#include "ui\_painter.h"

#include "canvas.h"

#include "newimagedialog.h"

#include "aboutdialog.h"

#include "actioncolors.h"

class AbstractTool;

class BrushTool;

class LineTool;

class CurveTool;

class EraserTool;

class TextTool;

class FillTool;

class TakeColorTool;

class SelectTool;

class ImageTool;

class Painter : public QMainWindow

{

Q\_OBJECT

public:

Painter(QWidget \*parent = 0, Qt::WFlags flags = 0);

~Painter();

ActionColors\* colors;

Canvas \*canvas;

Ui::PainterClass ui;

QActionGroup \*groupActionTool;

//Инструменты

BrushTool\* brushTool;

LineTool\* lineTool;

CurveTool\* curveTool;

EraserTool\* eraserTool;

TextTool\* textTool;

FillTool\* fillTool;

TakeColorTool\* takeColorTool;

SelectTool\* selectTool;

ImageTool\* imageTool;

private:

QString currentFileName;

private slots:

void onNewFileClicked();

void onOpenFileClicked();

void onSaveFileClicked();

void onSaveAsFileClicked();

void printDocument(QPrinter\*);

void onPrintClicked();

void onAboutClicked();

void onCopyClicked();

void onPasteClicked();

void onCutClicked();

void onSelectAllClicked();

void onDeselectClicked();

void selectFgColor();

void selectBgColor();

void changeColors(int, const QColor&);

void selectedTool(QAction\* action);

void onRestrictSelection(bool state);

void changeSA();

};

#endif // PAINTER\_H

***painter.cpp***

#include "stdafx.h"

#include "painter.h"

#include "abstracttool.h"

#include "brushtool.h"

#include "linetool.h"

#include "curvetool.h"

#include "erasertool.h"

#include "texttool.h"

#include "filltool.h"

#include "takecolortool.h"

#include "selecttool.h"

#include "imagetool.h"

Q\_DECLARE\_METATYPE(AbstractTool\*);

Painter::Painter(QWidget \*parent, Qt::WFlags flags)

: QMainWindow(parent, flags)

{

ui.setupUi(this);

//создание области рисования

canvas = new Canvas(this);

ui.canvasLayout->addWidget(canvas);

canvas->newImage(QColor(Qt::white), QSize(500, 350));

//при изменение области выделения на канвасе

connect(canvas, SIGNAL(changeSA()), this, SLOT(changeSA()));

//инициализация и установка цвета по умолчанию

colors = new ActionColors(this);

connect(colors, SIGNAL(changeColors(int, const QColor&)), this, SLOT(changeColors(int, const QColor&)));

colors->setFG(Qt::black);

colors->setBG(Qt::white);

//создание и назначение инструментов

brushTool = new BrushTool(canvas, ui.pageBrushTool, &ui, colors);

ui.actionBrush->setData(QVariant::fromValue<AbstractTool\*>(brushTool));

lineTool = new LineTool(canvas, ui.pageLineTool, &ui, colors);

ui.actionLine->setData(QVariant::fromValue<AbstractTool\*>(lineTool));

curveTool = new CurveTool(canvas, ui.pageLineTool, &ui, colors);

ui.actionCurve->setData(QVariant::fromValue<AbstractTool\*>(curveTool));

eraserTool = new EraserTool(canvas, ui.pageEraserTool, &ui, colors);

ui.actionEraser->setData(QVariant::fromValue<AbstractTool\*>(eraserTool));

textTool = new TextTool(canvas, ui.pageTextTool, &ui, colors);

ui.actionText->setData(QVariant::fromValue<AbstractTool\*>(textTool));

fillTool = new FillTool(canvas, ui.pageNoTool, &ui, colors);

ui.actionFill->setData(QVariant::fromValue<AbstractTool\*>(fillTool));

takeColorTool = new TakeColorTool(canvas, ui.pageNoTool, &ui, colors);

ui.actionTakeColor->setData(QVariant::fromValue<AbstractTool\*>(takeColorTool));

selectTool = new SelectTool(canvas, ui.pageSelectTool, &ui, colors);

ui.actionSelect->setData(QVariant::fromValue<AbstractTool\*>(selectTool));

imageTool = new ImageTool(canvas, ui.pageNoTool, &ui, colors);

//обьединение actions

groupActionTool = new QActionGroup(this);

groupActionTool->addAction(ui.actionBrush);

groupActionTool->addAction(ui.actionEraser);

groupActionTool->addAction(ui.actionText);

groupActionTool->addAction(ui.actionFill);

groupActionTool->addAction(ui.actionLine);

groupActionTool->addAction(ui.actionCurve);

groupActionTool->addAction(ui.actionTakeColor);

groupActionTool->addAction(ui.actionSelect);

connect(groupActionTool, SIGNAL(triggered(QAction\*)), this, SLOT(selectedTool(QAction\*)));

//назначение событий меню "Файл"

connect(ui.actionNewFile, SIGNAL(triggered()), this, SLOT(onNewFileClicked()));

connect(ui.actionOpenFile, SIGNAL(triggered()), this, SLOT(onOpenFileClicked()));

connect(ui.actionSaveFile, SIGNAL(triggered()), this, SLOT(onSaveFileClicked()));

connect(ui.actionSaveAsFile, SIGNAL(triggered()), this, SLOT(onSaveAsFileClicked()));

connect(ui.actionPrint, SIGNAL(triggered()), this, SLOT(onPrintClicked()));

connect(ui.actionExit, SIGNAL(triggered()), this, SLOT(close()));

//назначение событий меню "Помощь"

connect(ui.actionAbout, SIGNAL(triggered()), this, SLOT(onAboutClicked()));

//назначение событий меню "Правка"

connect(ui.actionCopy, SIGNAL(triggered()), this, SLOT(onCopyClicked()));

connect(ui.actionPaste, SIGNAL(triggered()), this, SLOT(onPasteClicked()));

connect(ui.actionCut, SIGNAL(triggered()), this, SLOT(onCutClicked()));

//назначение событий меню "Выделение"

connect(ui.actionSelectAll, SIGNAL(triggered()), this, SLOT(onSelectAllClicked()));

connect(ui.actionDeselect, SIGNAL(triggered()), this, SLOT(onDeselectClicked()));

//назначение событий для выбора цвета

connect(ui.actionFgColor, SIGNAL(triggered()), this, SLOT(selectFgColor()));

connect(ui.actionBgColor, SIGNAL(triggered()), this, SLOT(selectBgColor()));

//событие "ограничить выделенным"

connect(ui.checkBoxRestrictSelection, SIGNAL(toggled(bool)), this, SLOT(onRestrictSelection(bool)));

}

Painter::~Painter()

{

}

//---------------------------------------------

void Painter::onNewFileClicked()

{

NewImageDialog dialog(this);

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

{

currentFileName.clear();

canvas->newImage(dialog.getColor(), dialog.getSizeImage());

colors->setBG(dialog.getColor());

}

}

void Painter::onOpenFileClicked()

{

QFileDialog openDialog(this, "Открыть изображение");

openDialog.setAcceptMode(QFileDialog::AcceptOpen);

openDialog.setFileMode(QFileDialog::AnyFile);

//установка фильтра

QStringList filters;

QList<QByteArray> formats = QImageReader::supportedImageFormats();

QString filterAllImage("Все изображения ");

filterAllImage.append("(");

foreach(QByteArray format, formats) filterAllImage.append("\*." + format + " ");

filterAllImage.append(")");

filters.append(filterAllImage);

foreach(QByteArray format, formats) filters << "\*." + format;

openDialog.setNameFilters(filters);

//Вызов диалога

if (openDialog.exec() == QFileDialog::Accepted)

{

currentFileName = openDialog.selectedFiles().at(0);

bool result = canvas->loadImage(currentFileName);

if (result == false) QMessageBox::critical(this, "Ошибка", "Фаил не может быть загружен");

}

}

//\*\*\* несделано \*\*\*//

void Painter::onSaveFileClicked()

{

//Проверка возможности пересохранить файл

bool supportedFormat = false;

/\*QString fileFormat = "";

QList<QByteArray> formats = QImageWriter::supportedImageFormats();

QList<QByteArray>::iterator i = formats.begin();

while(i != formats.end())

{

QString format(\*i);

if (fileFormat == format)

{

supportedFormat = true;

break;

}

i++;

}\*/

//выбор действия

if (!currentFileName.isEmpty() && supportedFormat)

{

bool result = canvas->saveImage(currentFileName);

if (result == false) QMessageBox::critical(this, "Ошибка", "Фаил не может быть сохранён");

}

else

{

onSaveAsFileClicked();

}

}

void Painter::onSaveAsFileClicked()

{

QFileDialog saveDialog(this, "Сохранить изображение");

saveDialog.setAcceptMode(QFileDialog::AcceptSave);

saveDialog.setFileMode(QFileDialog::AnyFile);

//установка фильтра

QStringList filters;

QList<QByteArray> formats = QImageWriter::supportedImageFormats();

foreach(QByteArray format, formats) filters << "\*." + format;

saveDialog.setNameFilters(filters);

//Вызов диалога

if (saveDialog.exec() == QFileDialog::Accepted)

{

currentFileName = saveDialog.selectedFiles().at(0);

currentFileName.append("." + saveDialog.selectedFilter().split(".").at(1));

bool result = canvas->saveImage(currentFileName);

if (result == false) QMessageBox::critical(this, "Ошибка", "Фаил не может быть сохранён");

}

}

//---------------------------------------------

void Painter::onAboutClicked()

{

AboutDialog dialog(this);

dialog.exec();

}

//---------------------------------------------

void Painter::selectFgColor()

{

QColorDialog dialog(this);

dialog.setOption(QColorDialog::ShowAlphaChannel);

dialog.setCurrentColor(colors->getFG());

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

{

colors->setFG(dialog.currentColor());

}

}

void Painter::selectBgColor()

{

QColorDialog dialog(this);

dialog.setOption(QColorDialog::ShowAlphaChannel);

dialog.setCurrentColor(colors->getBG());

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

{

colors->setBG(dialog.currentColor());

}

}

void Painter::changeColors(int type, const QColor& color)

{

QAction\* action = (type == ActionColors::fg) ? ui.actionFgColor : ui.actionBgColor;

action->setIcon(ActionColors::getIconOfColor(color));

}

//---------------------------------------------

void Painter::onCopyClicked()

{

if (canvas->isSA())

{

QClipboard \*clipboard = QApplication::clipboard();

clipboard->setImage(canvas->getSA());

}

}

void Painter::onCutClicked()

{

if (canvas->isSA())

{

QClipboard \*clipboard = QApplication::clipboard();

clipboard->setImage(canvas->getSA());

//закрасить цветом фона

QPainter painter(canvas->getImage());

painter.setCompositionMode(QPainter::CompositionMode\_Source);

painter.fillRect(canvas->getSARect(), colors->getBG());

canvas->update();

}

}

//\*\*\* несделано \*\*\*//

void Painter::onPasteClicked()

{

canvas->clearSA();

canvas->setCurrentTool(imageTool);

QAction\* action = groupActionTool->checkedAction();

if (action != NULL) action->setChecked(false);

}

//---------------------------------------------

void Painter::onSelectAllClicked()

{

canvas->setSAMaxSize();

}

void Painter::onDeselectClicked()

{

canvas->clearSA();

}

//---------------------------------------------

void Painter::selectedTool(QAction\* action)

{

QVariant data = action->data();

if (data.isValid() && data.canConvert<AbstractTool\*>())

{

AbstractTool\* tool = data.value<AbstractTool\*>();

canvas->setCurrentTool(tool);

}

else

{

canvas->setCurrentTool(NULL);

}

}

void Painter::onRestrictSelection(bool state)

{

canvas->restrictSelection(state);

}

void Painter::changeSA()

{

//Обновление информации о выделенной области

QRect SARect = canvas->getSARect();

if (canvas->isSA())

{

ui.labelSelectX->setText(QString::number(SARect.x()));

ui.labelSelectY->setText(QString::number(SARect.y()));

ui.labelSelectW->setText(QString::number(SARect.width()));

ui.labelSelectH->setText(QString::number(SARect.height()));

}

else

{

ui.labelSelectX->setText("-");

ui.labelSelectY->setText("-");

ui.labelSelectW->setText("-");

ui.labelSelectH->setText("-");

}

}

//-----------------------------------------------------------------------

void Painter::onPrintClicked()

{

QPrinter printer(QPrinter::ScreenResolution);

QPrintPreviewDialog dialog(&printer, this);

dialog.setWindowFlags( Qt::Window );

connect(&dialog, SIGNAL(paintRequested(QPrinter \*)), SLOT(printDocument(QPrinter\*)));

dialog.exec();

}

void Painter::printDocument(QPrinter\* printer)

{

QPainter painter(printer);

printer->setPageMargins(20, 5, 5, 5, QPrinter::Millimeter);

painter.drawImage(0, 0, \*canvas->getImage());

}

***canvas.h***

#ifndef CANVAS\_H

#define CANVAS\_H

#include <QWidget>

#include "ui\_painter.h"

class AbstractTool;

class Painter;

class Canvas : public QWidget

{

Q\_OBJECT

public:

Canvas(QWidget \*parent);

~Canvas();

void newImage(QColor color, QSize size);

bool loadImage(const QString& fileName);

bool saveImage(const QString& fileName);

QImage\* getImage();

//методы выделенной области

void setSARect(QRect rect);

void setSAMaxSize();

void clearSA();

QImage getSA();

QRect getSARect();

bool isSA();

void restrictSelection(bool restrict);

bool isRestrictSelection();

//текущий инструмент

void setCurrentTool(AbstractTool\* \_tool);

AbstractTool\* getCurrentTool();

private:

void paintEvent(QPaintEvent \*event);

bool event(QEvent\* event);

QImage image;

AbstractTool\* tool;

QRect SARect;

bool RS;

QPixmap fon;

signals:

void changeSA();

};

#endif // CANVAS\_H

***canvas.cpp***

#include "StdAfx.h"

#include "canvas.h"

#include "abstracttool.h"

#include "painter.h"

Canvas::Canvas(QWidget \*parent)

: QWidget(parent), fon(":/Painter/Resources/img/fon.bmp")

{

QSizePolicy sizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

sizePolicy.setHorizontalStretch(0);

sizePolicy.setVerticalStretch(0);

setSizePolicy(sizePolicy);

clearSA();

tool = NULL;

}

Canvas::~Canvas()

{

}

void Canvas::newImage(QColor color, QSize size)

{

clearSA();

image = QImage(size, QImage::Format\_ARGB32);

//закрасить цветом фона

QPainter painter(getImage());

painter.setCompositionMode(QPainter::CompositionMode\_Source);

painter.fillRect(QRect(0, 0, image.width(), image.height()), color);

setMinimumSize(size);

}

bool Canvas::saveImage(const QString& fileName)

{

return image.save(fileName);

}

bool Canvas::loadImage(const QString& fileName)

{

clearSA();

bool result = image.load(fileName);

if (result == true) setMinimumSize(image.size());

return result;

}

QImage\* Canvas::getImage()

{

return &image;

}

void Canvas::paintEvent(QPaintEvent \*event)

{

//рисование pixmap

QPainter painter(this);

painter.setBackground(QBrush(fon));

painter.eraseRect(QRect(0, 0, width(), height()));

painter.drawImage(0, 0, image);

painter.end();

//предпросмотр

if (tool != NULL) tool->preview(this);

//рисование Select Area

if (isSA())

{

painter.begin(this);

painter.setCompositionMode(QPainter::CompositionMode\_Exclusion);

painter.setPen(QPen(Qt::white, 1, Qt::DashLine));

painter.drawRect(SARect.x(), SARect.y(), SARect.width()-1, SARect.height()-1);

if (RS)

{

painter.setCompositionMode(QPainter::CompositionMode\_SourceOver);

QColor rColor = QColor(3, 70, 132, 150);

QRect rectTop(0, 0, width() , SARect.y());

painter.fillRect(rectTop, rColor); //сверх

QRect rectBottom(QPoint(0, SARect.bottom()+1), QPoint(width(), height()));

painter.fillRect(rectBottom, rColor);//сниз

QRect rectLeft(QPoint(0, SARect.y()), QPoint(SARect.x()-1, SARect.bottom()));

painter.fillRect(rectLeft, rColor); //слева

QRect rectRight(QPoint(SARect.right()+1, SARect.y()), QPoint(width(), SARect.bottom()));

painter.fillRect(rectRight, rColor); //справа

}

painter.end();

}

}

bool Canvas::event(QEvent\* event)

{

bool result = QWidget::event(event);

if (tool != NULL) tool->canvasEvent(event);

return result;

}

//---------------------------------------------

void Canvas::setCurrentTool(AbstractTool\* \_tool)

{

if (tool != NULL) tool->disable(true);

tool = \_tool;

if (tool != NULL) tool->enable();

}

AbstractTool\* Canvas::getCurrentTool()

{

return tool;

}

//---------------------------------------------

void Canvas::clearSA()

{

SARect.setSize(QSize(0, 0));

changeSA();

update();

}

void Canvas::setSAMaxSize()

{

setSARect(QRect(0, 0, width(), height()));

update();

}

QImage Canvas::getSA()

{

return image.copy(SARect);

}

void Canvas::setSARect(QRect rect)

{

QRect newSARect = rect.intersect(image.rect());

if (SARect != newSARect)

{

SARect = newSARect;

changeSA();

}

update();

}

QRect Canvas::getSARect()

{

return SARect;

}

bool Canvas::isSA()

{

return SARect.height() > 2 && SARect.width() > 2;

}

void Canvas::restrictSelection(bool restrict)

{

RS = restrict;

update();

}

bool Canvas::isRestrictSelection()

{

return RS;

}

//---------------------------------------------

***actioncolors.h***

#ifndef ACTIONCOLORS\_H

#define ACTIONCOLORS\_H

#include <QObject>

class ActionColors : public QObject

{

Q\_OBJECT

public:

ActionColors(QObject \*parent);

~ActionColors();

enum TypeColor {bg = 0, fg = 1};

void setColor(TypeColor type, const QColor& color);

void setBG(const QColor& color);

void setFG(const QColor& color);

QColor getColor(TypeColor type);

QColor getBG();

QColor getFG();

static QIcon getIconOfColor(const QColor& color);

static QPixmap getPixmapOfColor(const QColor& color, const QSize& size);

private:

QColor colors[2];

signals:

void changeColors(int, const QColor&);

};

#endif // ACTIONCOLORS\_H

***actioncolors.cpp***

#include "StdAfx.h"

#include "actioncolors.h"

ActionColors::ActionColors(QObject \*parent)

: QObject(parent)

{

}

ActionColors::~ActionColors()

{

}

void ActionColors::setColor(TypeColor type, const QColor& color)

{

if (color != colors[type])

{

colors[type] = color;

emit changeColors(type, color);

}

}

void ActionColors::setBG(const QColor& color)

{

setColor(bg, color);

}

void ActionColors::setFG(const QColor& color)

{

setColor(fg, color);

}

QColor ActionColors::getColor(TypeColor type)

{

return colors[type];

}

QColor ActionColors::getBG()

{

return colors[bg];

}

QColor ActionColors::getFG()

{

return colors[fg];

}

QIcon ActionColors::getIconOfColor(const QColor& color)

{

QPixmap pixmap = getPixmapOfColor(color, QSize(16, 16));

QPainter painter(&pixmap);

painter.setPen(QPen(Qt::black));

painter.drawRect(0, 0, 15, 15);

return QIcon(pixmap);

}

QPixmap ActionColors::getPixmapOfColor(const QColor& color, const QSize& size)

{

QPixmap pixmap(size);

QPainter painter(&pixmap);

QPixmap fon(":/Painter/Resources/img/fon.bmp");

painter.setBackground(QBrush(fon));

painter.eraseRect(pixmap.rect());

painter.fillRect(pixmap.rect(), QBrush(color));

return pixmap;

}

***newimagedialog.h***

#ifndef NEWIMAGEDIALOG\_H

#define NEWIMAGEDIALOG\_H

#include <QDialog>

#include <QPixmap>

#include "ui\_newimagedialog.h"

#include "actioncolors.h"

class NewImageDialog : public QDialog

{

Q\_OBJECT

public:

NewImageDialog(QWidget \*parent = 0);

~NewImageDialog();

QColor getColor();

QSize getSizeImage();

private:

Ui::NewImageDialog ui;

QSize sizeImage;

QColor color;

private slots:

void onTakeColor();

void accept();

};

#endif // NEWIMAGEDIALOG\_H

***newimagedialog.cpp***

#include "StdAfx.h"

#include "newimagedialog.h"

NewImageDialog::NewImageDialog(QWidget \*parent)

: QDialog(parent)

{

ui.setupUi(this);

color = QColor(Qt::white);

ui.buttonColor->setIcon(ActionColors::getIconOfColor(color));

connect(ui.buttonCreate, SIGNAL(clicked()), this, SLOT(accept()));

connect(ui.buttonCancel, SIGNAL(clicked()), this, SLOT(reject()));

connect(ui.buttonColor, SIGNAL(clicked()), this, SLOT(onTakeColor()));

}

NewImageDialog::~NewImageDialog(){}

void NewImageDialog::accept()

{

sizeImage.setHeight(ui.editH->text().toInt());

sizeImage.setWidth(ui.editW->text().toInt());

QDialog::accept();

}

void NewImageDialog::onTakeColor()

{

QColorDialog colorDlg(this);

colorDlg.setOption(QColorDialog::ShowAlphaChannel);

int code = colorDlg.exec();

if (code != 0)

{

color = colorDlg.currentColor();

ui.buttonColor->setIcon(ActionColors::getIconOfColor(color));

}

}

QColor NewImageDialog::getColor()

{

return color;

}

QSize NewImageDialog::getSizeImage()

{

return sizeImage;

}

***abstracttool.h***

#ifndef ABSTRACTTOOL\_H

#define ABSTRACTTOOL\_H

#include <QObject>

#include "painter.h"

#include "canvas.h"

#include "actioncolors.h"

class AbstractTool : public QObject

{

Q\_OBJECT

public:

AbstractTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~AbstractTool();

virtual void enable();

virtual void disable(bool apply);

virtual void preview(QPaintDevice\* device);

//События canvas

virtual void canvasEvent(QEvent \*);

virtual void canvasMouseMove(QMouseEvent\*);

virtual void canvasMouseButtonRelease(QMouseEvent\*);

virtual void canvasMouseButtonPress(QMouseEvent\*);

virtual void canvasKeyPress(QKeyEvent\*);

protected:

Canvas\* canvas;

ActionColors\* colors;

Ui::PainterClass\* ui;

private:

QWidget\* page;

};

#endif // ABSTRACTTOOL\_H

***abstracttool.cpp***

#include "StdAfx.h"

#include "abstracttool.h"

AbstractTool::AbstractTool(Canvas\* \_canvas, QWidget\* \_page, Ui::PainterClass\* \_ui, ActionColors\* \_colors)

{

colors = \_colors;

canvas = \_canvas;

page = \_page;

ui = \_ui;

}

AbstractTool::~AbstractTool()

{

}

void AbstractTool::canvasEvent(QEvent \*event)

{

switch(event->type())

{

case QEvent::MouseMove:

{

QMouseEvent\* mouseEvent = static\_cast<QMouseEvent \*>(event);

canvasMouseMove(mouseEvent);

}

break;

case QEvent::MouseButtonPress:

{

QMouseEvent\* mouseEvent = static\_cast<QMouseEvent \*>(event);

canvasMouseButtonPress(mouseEvent);

}

break;

case QEvent::MouseButtonRelease:

{

QMouseEvent\* mouseEvent = static\_cast<QMouseEvent \*>(event);

canvasMouseButtonRelease(mouseEvent);

}

break;

case QEvent::KeyPress:

{

QKeyEvent\* keyEvent = static\_cast<QKeyEvent \*>(event);

canvasKeyPress(keyEvent);

}

}

}

void AbstractTool::enable()

{

ui->stackedParamTools->setCurrentWidget(page);

}

void AbstractTool::disable(bool apply)

{

ui->stackedParamTools->setCurrentWidget(ui->pageNoTool);

}

void AbstractTool::canvasMouseMove(QMouseEvent \*event) {}

void AbstractTool::canvasMouseButtonRelease(QMouseEvent \*event) {}

void AbstractTool::canvasMouseButtonPress(QMouseEvent \*event) {}

void AbstractTool::canvasKeyPress(QKeyEvent \*event) {}

void AbstractTool::preview(QPaintDevice\* device) {}

***brushtool.h***

#ifndef BRUSHTOOL\_H

#define BRUSHTOOL\_H

#include <QPointF>

#include <QPixmap>

#include <QColor>

#include <QLineF>

#include "abstracttool.h"

class BrushTool : public AbstractTool

{

Q\_OBJECT

public:

BrushTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~BrushTool();

private:

QColor color;

qreal size;

QPointF prPos;

QPixmap pixmap;

bool isPreview;

void drawLine(QPaintDevice\* device, QPointF p1, QPointF p2);

void drawCircle(QPaintDevice\* device, QPointF point);

protected:

void preview(QPaintDevice\* device);

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonRelease(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

};

#endif // BRUSHTOOL\_H

***brushtool.cpp***

#include "StdAfx.h"

#include "brushtool.h"

BrushTool::BrushTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

BrushTool::~BrushTool()

{

}

void BrushTool::canvasMouseMove(QMouseEvent \*event)

{

drawLine(&pixmap, prPos, event->posF());

prPos = event->posF();

canvas->update();

}

void BrushTool::canvasMouseButtonRelease(QMouseEvent \*event)

{

QPainter painter(canvas->getImage());

painter.drawPixmap(0, 0, pixmap);

isPreview = false;

}

void BrushTool::canvasMouseButtonPress(QMouseEvent \*event)

{

//Выбор цвета

color = (event->button() == Qt::LeftButton) ? colors->getFG() : colors->getBG();

//Задание размера

size = ui->sliderSizeBrush->value();

//Временный pixmap

pixmap = QPixmap(canvas->getImage()->size());

pixmap.fill(QColor(0,0,0,0));

drawCircle(&pixmap, event->posF());

prPos = event->posF();

isPreview = true;

}

void BrushTool::preview(QPaintDevice\* device)

{

if (isPreview)

{

QPainter painter(device);

painter.drawPixmap(0, 0, pixmap);

}

}

void BrushTool::drawLine(QPaintDevice\* device, QPointF p1, QPointF p2)

{

QPainter painter(device);

if (canvas->isRestrictSelection() && canvas->isSA())

{

painter.setClipRect(canvas->getSARect());

}

painter.setCompositionMode(QPainter::CompositionMode\_Source); //без учёта прозрачности цвета

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

painter.setPen(QPen(color, size, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter.drawLine(p1, p2);

canvas->update();

}

void BrushTool::drawCircle(QPaintDevice\* device, QPointF point)

{

QPainter painter(device);

if (canvas->isRestrictSelection() && canvas->isSA())

{

painter.setClipRect(canvas->getSARect());

}

painter.setCompositionMode(QPainter::CompositionMode\_Source); //без учёта прозрачности цвета

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

painter.setPen(Qt::NoPen);

painter.setBrush(color);

painter.drawEllipse(point, size/2, size/2);

canvas->update();

}

***erasertool.h***

#ifndef ERASERTOOL\_H

#define ERASERTOOL\_H

#include "abstracttool.h"

class EraserTool : public AbstractTool

{

Q\_OBJECT

public:

EraserTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~EraserTool();

private:

void draw();

QPoint p1, p2;

QColor color;

int size;

QPolygon getEraserShape(const QPoint& p1, const QPoint& p2, int size);

protected:

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

};

#endif // ERASERTOOL\_H

***erasertool.cpp***

#include "StdAfx.h"

#include "erasertool.h"

EraserTool::EraserTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

EraserTool::~EraserTool()

{

}

void EraserTool::canvasMouseMove(QMouseEvent\* event)

{

p1 = event->pos();

draw();

p2 = p1;

}

void EraserTool::canvasMouseButtonPress(QMouseEvent\* event)

{

p1 = event->pos();

p2 = event->pos();

color = (event->button() == Qt::LeftButton)? colors->getBG() : colors->getFG();

size = ui->sliderSizeEraser->value();

draw();

}

QPolygon EraserTool::getEraserShape(const QPoint& p1, const QPoint& p2, int size)

{

QPoint p = p1 - p2;

if (p.isNull())

{

QPolygon poly(4);

poly.setPoint(0, p1.x()+size/2, p1.y()+size/2);

poly.setPoint(1, p1.x()+size/2, p1.y()-size/2);

poly.setPoint(2, p1.x()-size/2, p1.y()-size/2);

poly.setPoint(3, p1.x()-size/2, p1.y()+size/2);

return poly;

}

int mx = p.x() > 0? 1 : -1;

int my = p.y() > 0? 1 : -1;

QPolygon poly(6);

//p1 rect

poly.setPoint(0, p1.x() + (size/2)\*mx, p1.y() - (size/2)\*my);

poly.setPoint(1, p1.x() + (size/2)\*mx, p1.y() + (size/2)\*my);

poly.setPoint(2, p1.x() - (size/2)\*mx, p1.y() + (size/2)\*my);

//p2 rect

poly.setPoint(3, p2.x() - (size/2)\*mx, p2.y() + (size/2)\*my);

poly.setPoint(4, p2.x() - (size/2)\*mx, p2.y() - (size/2)\*my);

poly.setPoint(5, p2.x() + (size/2)\*mx, p2.y() - (size/2)\*my);

return poly;

}

void EraserTool::draw()

{

QPainter painter(canvas->getImage());

if (canvas->isRestrictSelection() && canvas->isSA())

{

painter.setClipRect(canvas->getSARect());

}

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

painter.setCompositionMode(QPainter::CompositionMode\_Source);

painter.setPen(Qt::NoPen);

painter.setBrush(color);

painter.drawPolygon(getEraserShape(p1, p2, size));

canvas->update();

}

***filltool.h***

#ifndef FILLTOOL\_H

#define FILLTOOL\_H

#include <QRgb>

#include "abstracttool.h"

class FillTool : public AbstractTool

{

Q\_OBJECT

public:

FillTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~FillTool();

private:

QStack<QPoint> stack;

void floodFillScanlineStack(QImage\* image, int x, int y, QRgb newColor);

protected:

void canvasMouseButtonPress(QMouseEvent\*);

};

#endif // FILLTOOL\_H

***filltool.cpp***

#include "StdAfx.h"

#include "filltool.h"

FillTool::FillTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

FillTool::~FillTool()

{

}

void FillTool::canvasMouseButtonPress(QMouseEvent\* event)

{

QColor color = (event->button() == Qt::LeftButton)? colors->getFG() : colors->getBG();

floodFillScanlineStack(canvas->getImage(), event->x(), event->y(), color.rgba());

canvas->update();

}

//The scanline floodfill algorithm using our own stack routines, faster(http://lodev.org/cgtutor/floodfill.html)

void FillTool::floodFillScanlineStack(QImage\* image, int x, int y, QRgb newColor)

{

QRgb oldColor = image->pixel(x, y);

if (newColor == oldColor) return;

stack.clear();

int y1;

bool spanLeft, spanRight;

int w = image->width();

int h = image->height();

stack.push(QPoint(x, y));

while(!stack.isEmpty())

{

QPoint point = stack.pop();

x = point.x();

y = point.y();

y1 = y;

while(y1 >= 0 && image->pixel(x, y1) == oldColor) y1--;

y1++;

spanLeft = spanRight = 0;

while(y1 < h && image->pixel(x, y1) == oldColor )

{

image->setPixel(x, y1, newColor);

if(!spanLeft && x > 0 && image->pixel(x - 1, y1) == oldColor)

{

stack.push(QPoint(x - 1, y1));

spanLeft = 1;

}

else if(spanLeft && x > 0 && image->pixel(x - 1, y1) != oldColor)

{

spanLeft = 0;

}

if(!spanRight && x < w - 1 && image->pixel(x + 1, y1) == oldColor)

{

stack.push(QPoint(x + 1, y1));

spanRight = 1;

}

else if(spanRight && x < w - 1 && image->pixel(x + 1, y1) != oldColor)

{

spanRight = 0;

}

y1++;

}

}

}

***imagetool.h***

#ifndef IMAGETOOL\_H

#define IMAGETOOL\_H

#include "abstracttool.h"

class ImageTool : public AbstractTool

{

Q\_OBJECT

public:

ImageTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~ImageTool();

private:

QPoint pos;

QPoint dragPoint;

bool isImage, isDrag;

QImage image;

void draw(QPaintDevice\* device, bool preview);

protected:

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonRelease(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

void preview(QPaintDevice\* device);

void enable();

void disable(bool apply);

};

#endif // IMAGETOOL\_H

***imagetool.cpp***

#include "StdAfx.h"

#include "imagetool.h"

ImageTool::ImageTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

ImageTool::~ImageTool()

{

}

void ImageTool::enable()

{

AbstractTool::enable();

//начальные значения

pos = QPoint(0,0);

dragPoint = QPoint(0,0);

isImage = false;

isDrag = false;

//Взять изображение из буфера обмена

QClipboard \*clipboard = QApplication::clipboard();

image = clipboard->image();

isImage = true;

canvas->update();

}

void ImageTool::disable(bool apply)

{

AbstractTool::disable(apply);

//нарисовать

if (apply)

draw(canvas->getImage(), false);

canvas->update();

}

void ImageTool::preview(QPaintDevice\* device)

{

draw(device, true);

}

void ImageTool::draw(QPaintDevice\* device, bool preview)

{

if (isImage)

{

QPainter painter(device);

painter.drawImage(pos, image);

if (preview)

{

QRect rectP = image.rect();

rectP.moveTo(pos);

painter.setPen(QPen(Qt::black, 1, Qt::DashLine));

painter.drawRect(rectP.x(), rectP.y(), rectP.width()-1, rectP.height()-1);

}

}

}

void ImageTool::canvasMouseMove(QMouseEvent\* event)

{

if (isDrag)

{

QPoint mp = event->pos();

pos = mp + dragPoint;

canvas->update();

}

}

void ImageTool::canvasMouseButtonPress(QMouseEvent\* event)

{

if (isImage)

{

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

{

QPoint mp = event->pos();

QRect rectP = image.rect();

rectP.moveTo(pos);

if (rectP.contains(mp))

{

dragPoint = rectP.topLeft() - mp;

isDrag = true;

}

else

{

//нарисовать

draw(canvas->getImage(), false);

isImage = false;

}

}

else

{

//нарисовать

draw(canvas->getImage(), false);

isImage = false;

}

}

canvas->update();

}

void ImageTool::canvasMouseButtonRelease(QMouseEvent\* event)

{

isDrag = false;

}

***linetool.h***

#ifndef LINETOOL\_H

#define LINETOOL\_H

#include "abstracttool.h"

class LineTool : public AbstractTool

{

Q\_OBJECT

public:

LineTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~LineTool();

enum StateLineTool {SetPoints = 0, FindMovePoint = 1, MovePoint = 3};

private:

QVector<QPoint> points;

QPoint mp;

StateLineTool state;

int width;

int radius;

int indexMoveP;

void draw(QPaintDevice\* device, bool preview);

int findIndexPoint(const QVector<QPointF>& points, const QPointF& pos, qreal radius);

protected:

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

void canvasMouseButtonRelease(QMouseEvent\*);

void preview(QPaintDevice\* device);

void enable();

void disable(bool apply);

public slots:

void sizeChanged(int i);

};

#endif // LINETOOL\_H

***linetool.cpp***

#include "StdAfx.h"

#include "linetool.h"

LineTool::LineTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

connect(ui->sliderSizeLine, SIGNAL(valueChanged(int)), this, SLOT(sizeChanged(int)));

}

LineTool::~LineTool()

{

}

void LineTool::enable()

{

AbstractTool::enable();

sizeChanged(ui->sliderSizeLine->value());

state = SetPoints;

canvas->setMouseTracking(true);

}

void LineTool::disable(bool apply)

{

AbstractTool::disable(apply);

if (apply) draw(canvas->getImage(), false);

points.clear();

canvas->update();

canvas->setMouseTracking(false);

}

void LineTool::canvasMouseMove(QMouseEvent\* event)

{

mp = event->pos();

if (state == MovePoint)

points[indexMoveP] = mp;

canvas->update();

}

void LineTool::canvasMouseButtonRelease(QMouseEvent\* event)

{

if (state == MovePoint && event->button() == Qt::LeftButton)

{

state = FindMovePoint;

}

}

void LineTool::canvasMouseButtonPress(QMouseEvent\* event)

{

int btn = event->button();

switch(state)

{

case SetPoints:

{

if (btn == Qt::LeftButton)

{

points.append(event->pos());

}

else

{

if (points.size() == 1)

{

points.clear();

}

else if (points.size() > 1)

{

state = FindMovePoint;

}

}

}

break;

case FindMovePoint:

{

if (btn == Qt::LeftButton)

{

indexMoveP = -1;

qreal r = radius;

for (int i = 0; i < points.size(); i++)

{

qreal x = points[i].x() - event->x();

qreal y = points[i].y() - event->y();

qreal len = qSqrt(x\*x + y\*y);

if (len < r)

{

r = len;

indexMoveP = i;

}

}

if (indexMoveP != -1)

{

state = MovePoint;

}

else

{

//Нарисовать на pixmap

draw(canvas->getImage(), false);

state = SetPoints;

points.clear();

//...

if (btn == Qt::LeftButton) points.append(event->pos());

}

}

else if (btn == Qt::RightButton)

{

//Нарисовать на pixmap

draw(canvas->getImage(), false);

state = SetPoints;

points.clear();

}

}

}

ui->labelCountPoint->setText(QString::number(points.size()));

canvas->update();

}

int LineTool::findIndexPoint(const QVector<QPointF>& points, const QPointF& pos, qreal radius)

{

int j = -1;

qreal r = radius;

for (int i = 0; i < points.size(); i++)

{

qreal x = points[i].x() - pos.x();

qreal y = points[i].y() - pos.y();

qreal len = qSqrt(x\*x + y\*y);

if (len < r)

{

r = len;

j = i;

}

}

return j;

}

void LineTool::preview(QPaintDevice\* device)

{

draw(device, true);

}

void LineTool::draw(QPaintDevice\* device, bool preview)

{

if (points.size() > 0)

{

QPainter painter(device);

if (canvas->isRestrictSelection() && canvas->isSA())

{

painter.setClipRect(canvas->getSARect());

}

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

painter.setPen(QPen(colors->getFG(), width, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

//рисование линий

QPainterPath path;

path.moveTo(points[0]);

foreach(QPoint point, points)

path.lineTo(point);

if (preview && state == SetPoints) path.lineTo(mp);

painter.drawPath(path);

//рисование кружков на изломах

if (preview)

{

painter.setClipping(false);

painter.setPen(QPen(Qt::black, 2));

foreach(QPoint point, points)

painter.drawEllipse(point, radius/2, radius/2);

//кружок под курсором

if (state == SetPoints)

{

painter.drawEllipse(mp, radius/2, radius/2);

}

}

}

}

void LineTool::sizeChanged(int i)

{

width = i;

radius = ( qMax(width, 8) + 4 ) \* 1.5;

canvas->update();

}

***selecttool.h***

#ifndef SELECTTOOL\_H

#define SELECTTOOL\_H

#include <QPoint>

#include "abstracttool.h"

class SelectTool : public AbstractTool

{

Q\_OBJECT

public:

SelectTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~SelectTool();

private:

QPoint point;

bool selectActive;

protected:

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonRelease(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

};

#endif // SELECTTOOL\_H

***selecttool.cpp***

#include "StdAfx.h"

#include "selecttool.h"

SelectTool::SelectTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

SelectTool::~SelectTool()

{

}

void SelectTool::canvasMouseMove(QMouseEvent\* event)

{

if (selectActive)

{

QRect rect(point, event->pos());

canvas->setSARect(rect.normalized());

}

}

void SelectTool::canvasMouseButtonRelease(QMouseEvent\* event)

{

if (selectActive)

{

QRect rect(point, event->pos());

canvas->setSARect(rect.normalized());

selectActive = false;

}

}

void SelectTool::canvasMouseButtonPress(QMouseEvent\* event)

{

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

{

point = event->pos();

selectActive = true;

}

else if (event->button() == Qt::RightButton)

{

canvas->clearSA();

selectActive = false;

}

}

***takecolortool.h***

#ifndef TAKECOLORTOOL\_H

#define TAKECOLORTOOL\_H

#include "abstracttool.h"

class TakeColorTool : public AbstractTool

{

Q\_OBJECT

public:

TakeColorTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~TakeColorTool();

protected:

void canvasMouseButtonPress(QMouseEvent\*);

void disable(bool apply);

void enable();

};

#endif // TAKECOLORTOOL\_H

***takecolortool.cpp***

#include "StdAfx.h"

#include "takecolortool.h"

TakeColorTool::TakeColorTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

}

TakeColorTool::~TakeColorTool()

{

}

void TakeColorTool::enable()

{

AbstractTool::enable();

canvas->setMouseTracking(true);

}

void TakeColorTool::disable(bool apply)

{

AbstractTool::disable(apply);

canvas->setMouseTracking(false);

}

void TakeColorTool::canvasMouseButtonPress(QMouseEvent\* event)

{

QRgb rgb = canvas->getImage()->pixel(event->pos());

QColor color = QColor::fromRgba(rgb);

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

colors->setFG(color);

else

colors->setBG(color);

}

***texttool.h***

#ifndef TEXTTOOL\_H

#define TEXTTOOL\_H

#include "abstracttool.h"

class TextTool : public AbstractTool

{

Q\_OBJECT

public:

TextTool(Canvas\*, QWidget\*, Ui::PainterClass\*, ActionColors\*);

~TextTool();

enum StateTextTool {SetPositionText = 0, EditText = 1, MoveText = 2};

private:

QPoint pos;

QRect rect;

QString text;

QFont font;

bool visibleFon;

int cpos;

QPoint dragPoint;

StateTextTool state;

void draw(QPaintDevice\*);

protected:

void canvasMouseMove(QMouseEvent\*);

void canvasMouseButtonRelease(QMouseEvent\*);

void canvasMouseButtonPress(QMouseEvent\*);

void preview(QPaintDevice\* device);

void enable();

void disable(bool apply);

public slots:

void valueChanged();

void textCPhanged(int, int);

};

#endif // TEXTTOOL\_H

***texttool.cpp***

#include "StdAfx.h"

#include "texttool.h"

TextTool::TextTool(Canvas\* canvas, QWidget\* page, Ui::PainterClass\* ui, ActionColors\* colors)

: AbstractTool(canvas, page, ui, colors)

{

connect(ui->textTool\_edit, SIGNAL(textChanged(const QString&)), this, SLOT(valueChanged()));

connect(ui->textTool\_size, SIGNAL(valueChanged(const QString&)), this, SLOT(valueChanged()));

connect(ui->textTool\_font, SIGNAL(currentFontChanged(const QFont&)), this, SLOT(valueChanged()));

connect(ui->textTool\_fonV, SIGNAL(stateChanged(int)), this, SLOT(valueChanged()));

connect(ui->checkItalic, SIGNAL(stateChanged(int)), this, SLOT(valueChanged()));

connect(ui->checkBold, SIGNAL(stateChanged(int)), this, SLOT(valueChanged()));

connect(ui->textTool\_edit, SIGNAL(cursorPositionChanged(int, int)), this, SLOT(textCPhanged(int, int)));

valueChanged();

}

TextTool::~TextTool()

{

}

void TextTool::disable(bool apply)

{

AbstractTool::disable(apply);

if (apply) draw(canvas->getImage());

state = SetPositionText;

ui->textTool\_edit->clear();

canvas->update();

}

void TextTool::enable()

{

AbstractTool::enable();

state = SetPositionText;

}

void TextTool::canvasMouseMove(QMouseEvent\* event)

{

if (state == MoveText)

{

pos = event->pos() + dragPoint;

canvas->update();

}

}

void TextTool::canvasMouseButtonRelease(QMouseEvent\* event)

{

if (state == MoveText) state = EditText;

}

void TextTool::canvasMouseButtonPress(QMouseEvent\* event)

{

int btn = event->button();

QRect rectP = rect;

rectP.moveTo(pos);

if (state == EditText)

{

if (btn == Qt::RightButton)

{

draw(canvas->getImage());

ui->textTool\_edit->clear();

state = SetPositionText;

}

else

{

if (rectP.contains(event->pos()))

{

state = MoveText;

dragPoint = rectP.topLeft() - event->pos();

}

else

{

draw(canvas->getImage());

ui->textTool\_edit->clear();

state = SetPositionText;

}

}

}

if (state == SetPositionText && btn == Qt::LeftButton)

{

pos = event->pos();

ui->textTool\_edit->setFocus();

state = EditText;

}

canvas->update();

}

void TextTool::draw(QPaintDevice\* device)

{

QPainter painter(device);

if (canvas->isRestrictSelection() && canvas->isSA())

{

painter.setClipRect(canvas->getSARect());

}

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

painter.setPen(colors->getFG());

painter.setFont(font);

QRect rectP = rect;

rectP.moveTo(pos);

if (!visibleFon)

painter.fillRect(rectP, colors->getBG());

painter.drawText(rectP , 0, text);

}

void TextTool::preview(QPaintDevice\* device)

{

if (state == EditText || state == MoveText)

{

draw(device);

//Вывод курсора и выделения

QFontMetrics fm(font);

int shiftCW = fm.width(text, cpos); //смешение текстового курсора

int h = fm.height(); //высота текста

QRect rectP = rect;

rectP.moveTo(pos);

int xc = rectP.x() + shiftCW;

int y = rectP.y();

QPainter painter(device);

painter.fillRect(QRect(xc, y, 1, h), colors->getFG());

}

}

void TextTool::valueChanged()

{

text = ui->textTool\_edit->text();

visibleFon = ui->textTool\_fonV->isChecked();

int size = ui->textTool\_size->value();

bool bold = ui->checkBold->isChecked();

bool italic = ui->checkItalic->isChecked();

font = ui->textTool\_font->currentFont();

font.setPointSize(size);

font.setItalic(italic);

font.setBold(bold);

QFontMetrics fm(font);

rect = fm.boundingRect(text);

canvas->update();

}

void TextTool::textCPhanged(int oldP, int newP)

{

cpos = newP;

canvas->update();

}

**Диаграмма вариантов использования**

