МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 10

по дисциплине: Объектно-ориентированное программирование

тема: «Закрепление навыков программирования в объектно-ориентированном стиле. Визуальные компоненты. Знакомство с QT.»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
асс. Черников Сергей Викторович

Белгород 2024г.

Лабораторная работа №10

«Закрепление навыков программирования в объектно-ориентированном стиле. Визуальные компоненты. Знакомство с QT.»

Цель работы: приобретение практических навыков создания приложений на языке C++.

Вариант 3

Создать графический редактор типа Painter, отрисовка стандартных графических примитивов, выбор цвета, толщины нажима. Сохранение в файл, загрузка из файла.

Задание 1

2024_PV-223_Artur_3 Owner

Задание 2

Код программы:

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QList>
#include <QMainWindow>
class ScribbleArea;
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow();
protected:
    void closeEvent(QCloseEvent *event) override;
private slots:
            void open();
    void save();
    void penColor();
    void penWidth();
    void about();
private:
    void createActions();
    void createMenus();
    bool maybeSave();
    bool saveFile(const QByteArray &fileFormat);
    ScribbleArea *scribbleArea;
    QMenu *saveAsMenu;
    QMenu *fileMenu;
    QMenu *optionMenu;
    QMenu *primitivesMenu;
    QMenu *helpMenu;
    QAction *openAct;
    QList<QAction *> saveAsActs;
```

```
    QAction *exitAct;
    QAction *penColorAct;
    QAction *penWidthAct;
    QAction *printAct;
    QAction *paintCircleAct;
    QAction *paintTriangleAct;
    QAction *clearScreenAct;
    QAction *aboutAct;
    QAction *aboutQtAct;
};
#endif
```

paint.h:

```
#ifndef PAINT_H
#define PAINT_H
#include <QColor>
#include <QImage>
#include <QPoint>
#include <QWidget>
class ScribbleArea : public QWidget
{
    Q_OBJECT
public:
    ScribbleArea(QWidget *parent = 0);
    bool openImage(const QString &fileName);
    bool saveImage(const QString &fileName, const char *fileFormat);
    void setPenColor(const QColor &newColor);
    void setPenWidth(int newWidth);
    bool isModified() const { return modified; }
    QColor penColor() const { return myPenColor; }
    int penWidth() const { return myPenWidth; }
public slots:
            void clearImage();
    void print();
protected:
    void mousePressEvent(QMouseEvent *event) override;
    void mouseMoveEvent(QMouseEvent *event) override;
    void mouseReleaseEvent(QMouseEvent *event) override;
    void paintEvent(QPaintEvent *event) override;
    void paintCircle(const QPoint &center, int radius);
    void paintTriangle(const QPoint &p1, const QPoint &p2, const QPoint &p);
    void resizeEvent(QResizeEvent *event) override;
private:
    void drawLineTo(const QPoint &endPoint);
    void resizeImage(QImage *image, const QSize &newSize);
    bool modified;
    bool scribbling;
    int myPenWidth;
    QColor myPenColor;
    QImage image;
    QPoint lastPoint;
};
#endif // PAINT_H
```

PtrSmart:

```
#ifndef PTRSMART_H
#define PTRSMART_H
template <typename T>
class smart_ptr {
```

```
    T * obj;
public:
    smart_ptr(T *obj) : obj(obj) {};
    ~smart_ptr() {
        delete obj;
    }
    operator T*() {return obj;}
    T* operator->() { return obj; }
    T& operator* () { return *obj; }
};
#endif // PTRSMART_H
```

main.cpp:

```
#include "mainwindow.h"
#include "PtrSmart.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    smart_ptr<MainWindow> window(new MainWindow());
    window->show();
    return app.exec();
}
```

Mainwindow.cpp:

```
#include <QtWidgets>
#include <mainwindow.h>
#include <paint.h>
MainWindow::MainWindow()
{
    scribbleArea = new ScribbleArea;
    setCentralWidget(scribbleArea);
    createActions();
    createMenus();
    setWindowTitle(tr("Paint"));
    resize(1000, 1000);
}
void MainWindow::closeEvent(QCloseEvent *event)
{
    if (maybeSave()) {
        event->accept();
    } else {
        event->ignore();
    }
}
void MainWindow::open()
{
    if (maybeSave()) {
        QString fileName = QFileDialog::getOpenFileName(this,
                                                         tr("Open File"),
                                                         QDir::currentPath());
        if (!fileName.isEmpty())
            scribbleArea->openImage(fileName);
    }
}
void MainWindow::save()
{
    QAction *action = qobject_cast<QAction *>(sender());
    QByteArray fileFormat = action->data().toByteArray();
```

```
    saveFile(fileFormat);
}
void MainWindow::penColor()
{
    QColor newColor = QColorDialog::getColor(scribbleArea->penColor());
    if (newColor.isValid())
        scribbleArea->setPenColor(newColor);
}
void MainWindow::penWidth()
{
    bool ok;
    int newWidth = QInputDialog::getInt(this, tr("Scribble"),
                                        tr("Select pen width:"),
                                        scribbleArea->penWidth(),
                                        1, 50, 1, &ok);
    if (ok)
        scribbleArea->setPenWidth(newWidth);
}
void MainWindow::about()
{
    QMessageBox::about(this, tr("About Scribble"),
                       tr("<p>The <b>Scribble</b> example is awesome</p>"));
}
void MainWindow::createActions()
{
    openAct = new QAction(tr("&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);
    }
    printAct = new QAction(tr("&Print..."), this);
    connect(printAct, SIGNAL(triggered()), scribbleArea, SLOT(print()));
    exitAct = new QAction(tr("E&xit"), this);
    exitAct->setShortcuts(QKeySequence::Quit);
    connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));
    penColorAct = new QAction(tr("&Pen Color..."), this);
    connect(penColorAct, SIGNAL(triggered()), this, SLOT(penColor()));
    penWidthAct = new QAction(tr("Pen &Width..."), this);
    connect(penWidthAct, SIGNAL(triggered()), this, SLOT(penWidth()));
    clearScreenAct = new QAction(tr("&Clear Screen"), this);
    clearScreenAct->setShortcut(tr("Ctrl+L"));
    connect(clearScreenAct, SIGNAL(triggered()),
            scribbleArea, SLOT(clearImage()));
    paintCircleAct = new QAction(tr("Circle..."), this);
    connect(paintCircleAct, SIGNAL(triggered()), this, SLOT(paintCircleAct));
    paintTriangleAct = new QAction(tr("Triangle..."), this);
    connect(paintTriangleAct, SIGNAL(triggered()), this,
            SLOT(paintTriangleAct));
    aboutAct = new QAction(tr("&About"), this);
    connect(aboutAct, SIGNAL(triggered()), this, SLOT(about()));
    aboutQtAct = new QAction(tr("About &Qt"), this);
    connect(aboutQtAct, SIGNAL(triggered()), qApp, SLOT(aboutQt()));
}
void MainWindow::createMenus()
{
    saveAsMenu = new QMenu(tr("&Save As"), this);
    foreach (QAction *action, saveAsActs)
    saveAsMenu->addAction(action);
    fileMenu = new QMenu(tr("&File"), this);
```

```
    fileMenu->addAction(openAct);
    fileMenu->addMenu(saveAsMenu);
    fileMenu->addAction(printAct);
    fileMenu->addSeparator();
    fileMenu->addAction(exitAct);
    optionMenu = new QMenu(tr("&Options"), this);
    optionMenu->addAction(penColorAct);
    optionMenu->addAction(penWidthAct);
    optionMenu->addSeparator();
    optionMenu->addAction(clearScreenAct);
    primitivesMenu = new QMenu(tr("&Primitives"), this);
    primitivesMenu ->addAction(paintCircleAct);
    primitivesMenu ->addAction(paintTriangleAct);
    helpMenu = new QMenu(tr("&Help"), this);
    helpMenu->addAction(aboutAct);
    helpMenu->addAction(aboutQtAct);
    menuBar()->addMenu(fileMenu);
    menuBar()->addMenu(optionMenu);
    menuBar()->addMenu(primitivesMenu);
    menuBar()->addMenu(helpMenu);
}
bool MainWindow::maybeSave()
{
    if (scribbleArea->isModified()) {
        QMessageBox::StandardButton ret;
        ret = QMessageBox::warning(this, tr("Scribble"),
                                   tr("The image has been modified.\n"
                                      "Do you want to save your changes?"),
                                   QMessageBox::Save | QMessageBox::Discard
                                   | QMessageBox::Cancel);
        if (ret == QMessageBox::Save) {
            return saveFile("png");
// If cancel do nothing
        } else if (ret == QMessageBox::Cancel) {
            return false;
        }
    }
    return true;
}
bool MainWindow::saveFile(const QByteArray &fileFormat)
{
    QString initialPath = QDir::currentPath() + "/untitled." + fileFormat;
    QString fileName = QFileDialog::getSaveFileName(this, tr("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 scribbleArea->saveImage(fileName, fileFormat.constData());
    }
}
```

Paint.cpp:

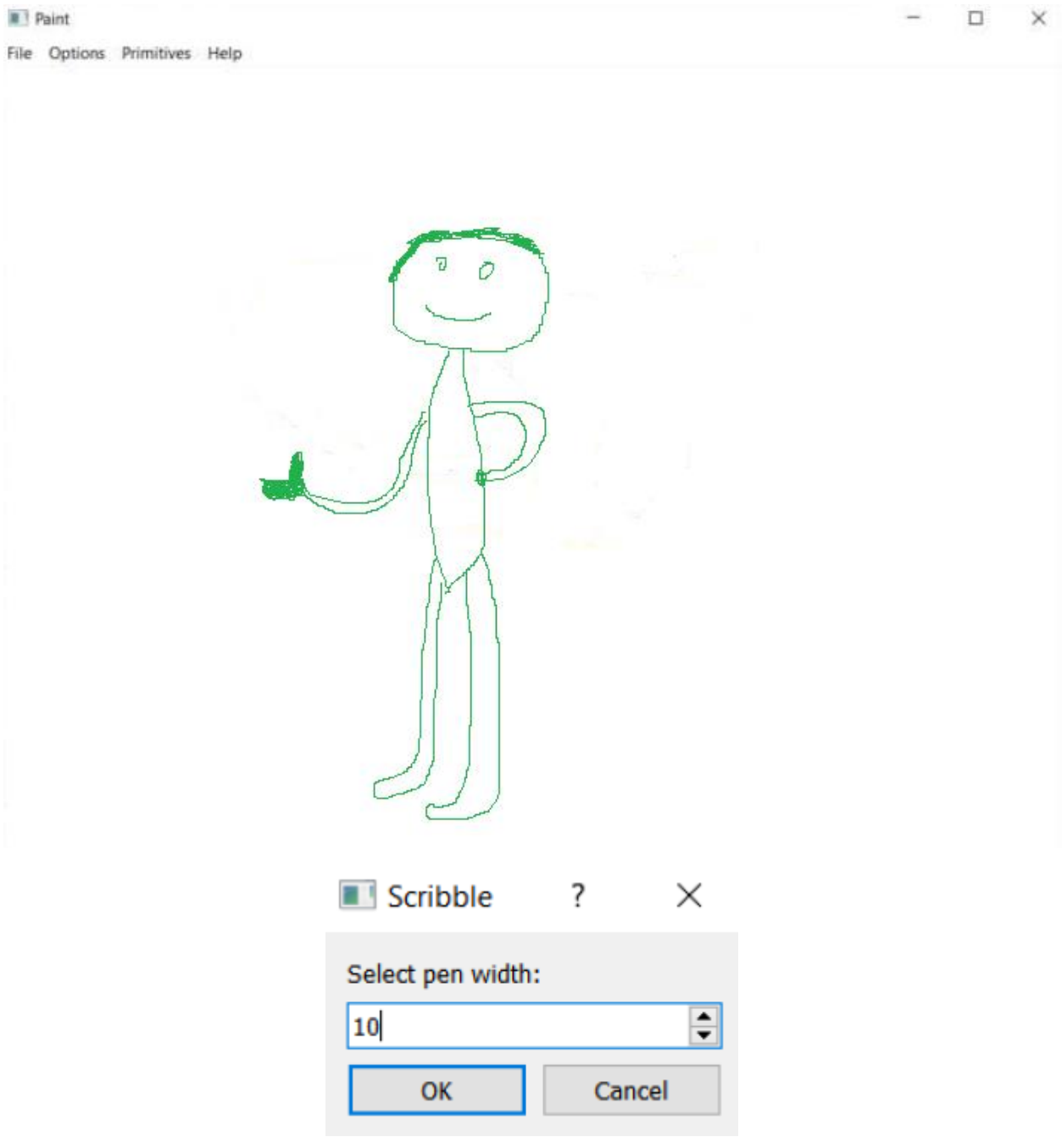
```
#include <QtWidgets>
#if defined(QT_PRINTSUPPORT_LIB)
#include <QtPrintSupport/qtprintsupportglobal.h>
#if QT_CONFIG(printdialog)
#include <QPrinter>
#include <QPrintDialog>
```

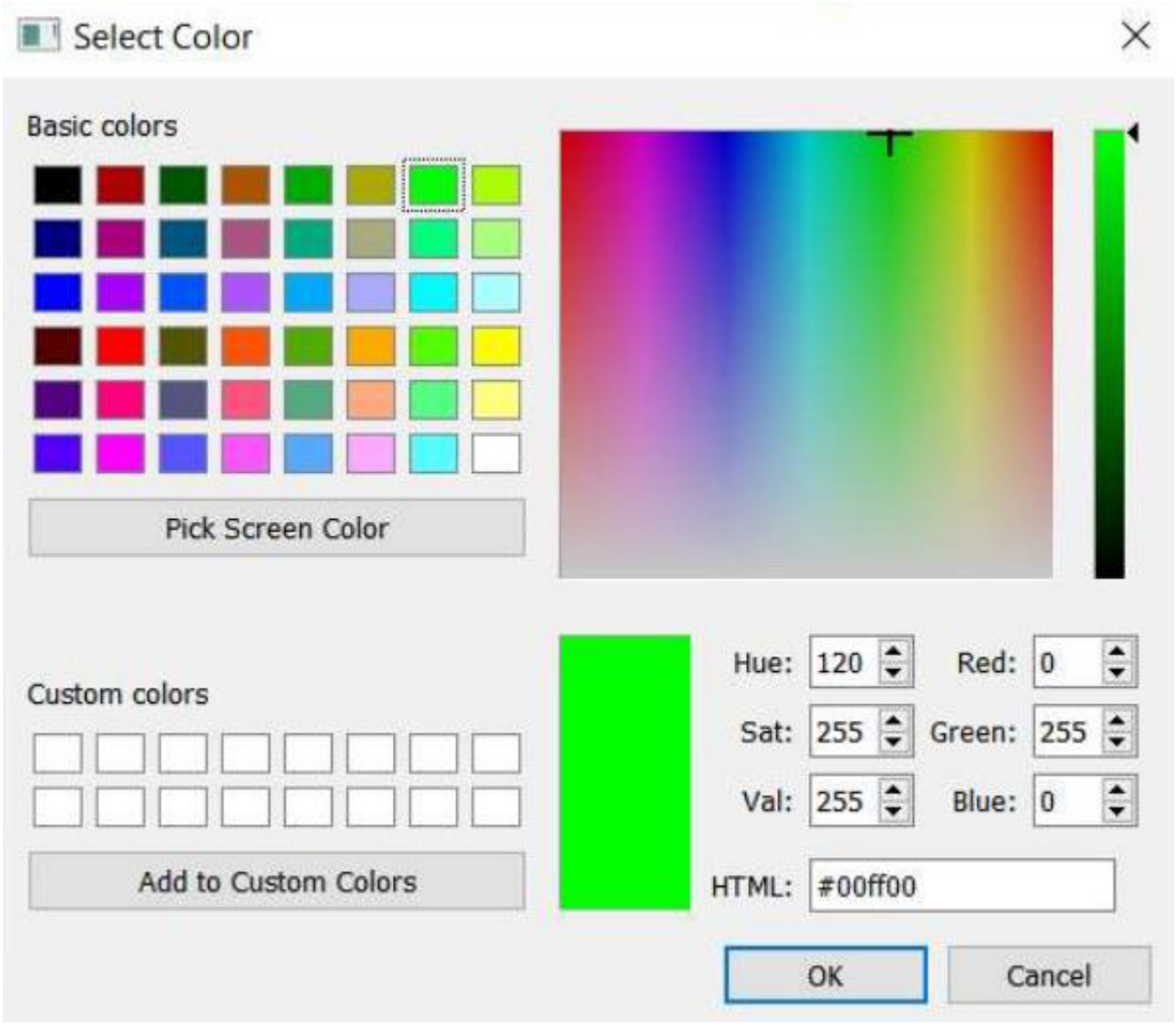
```
#endif
#endif
#include <paint.h>
ScribbleArea::ScribbleArea(QWidget *parent)
        : QWidget(parent)
{
    setAttribute(Qt::WA_StaticContents);
    modified = false;
    scribbling = false;
    myPenWidth = 1;
    myPenColor = Qt::blue;
}
bool ScribbleArea::openImage(const QString &fileName)
{
    QImage loadedImage;
    if (!loadedImage.load(fileName))
        return false;
    QSize newSize = loadedImage.size().expandedTo(size());
    resizeImage(&loadedImage, newSize);
    image = loadedImage;
    modified = false;
    update();
    return true;
}
bool ScribbleArea::saveImage(const QString &fileName, const char *fileFormat)
{
    QImage visibleImage = image;
    resizeImage(&visibleImage, size());
    if (visibleImage.save(fileName, fileFormat)) {
        modified = false;
        return true;
    } else {
        return false;
    }
}
void ScribbleArea::setPenColor(const QColor &newColor)
{
    myPenColor = newColor;
}
void ScribbleArea::setPenWidth(int newWidth)
{
    myPenWidth = newWidth;
}
void ScribbleArea::clearImage()
{
    image.fill(qRgb(255, 255, 255));
    modified = true;
    update();
}
void ScribbleArea::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton) {
        lastPoint = event->pos();
        scribbling = true;
    }
}
void ScribbleArea::mouseMoveEvent(QMouseEvent *event)
{
    if ((event->buttons() & Qt::LeftButton) && scribbling)
        drawLineTo(event->pos());
}
void ScribbleArea::mouseReleaseEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton && scribbling) {
```

```
        drawLineTo(event->pos());
        scribbling = false;
    }
}
void ScribbleArea::paintEvent(QPaintEvent *event)
{
    QPainter painter(this);
    QRect dirtyRect = event->rect();
    painter.drawImage(dirtyRect, image, dirtyRect);
}
void ScribbleArea::paintCircle(const QPoint &center, int radius)
{
    QPainter painter(&image);
    painter.setRenderHint(QPainter::Antialiasing);
    painter.setPen(Qt::black);
    painter.drawEllipse(center, radius, radius);
    update();
}
void ScribbleArea::paintTriangle(const QPoint &p1, const QPoint &p2, const
QPoint &p3)
{
    QPainter painter(&image);
    painter.setRenderHint(QPainter::Antialiasing);
    painter.setPen(Qt::black);
    painter.drawLine(p1, p2);
    painter.drawLine(p2, p3);
    painter.drawLine(p3, p1);
    update();
}
void ScribbleArea::resizeEvent(QResizeEvent *event)
{
    if (width() > image.width() || height() > image.height()) {
        int newWidth = qMax(width() + 128, image.width());
        int newHeight = qMax(height() + 128, image.height());
        resizeImage(&image, QSize(newWidth, newHeight));
        update();
    }
    QWidget::resizeEvent(event);
}
void ScribbleArea::drawLineTo(const QPoint &endPoint)
{
    QPainter painter(&image);
    painter.setPen(QPen(myPenColor, myPenWidth, Qt::SolidLine, Qt::RoundCap,
                        Qt::RoundJoin));
    painter.drawLine(lastPoint, endPoint);
    modified = true;
    int rad = (myPenWidth / 2) + 2;
    update(QRect(lastPoint, endPoint).normalized()
                   .adjusted(-rad, -rad, +rad, +rad));
    lastPoint = endPoint;
}
void ScribbleArea::resizeImage(QImage *image, const QSize &newSize)
{
    if (image->size() == newSize)
        return;
    QImage newImage(newSize, QImage::Format_RGB32);
    newImage.fill(qRgb(255, 255, 255));
    QPainter painter(&newImage);
    painter.drawImage(QPoint(0, 0), *image);
    *image = newImage;
}
void ScribbleArea::print()
{
#if QT_CONFIG(printdialog)
```

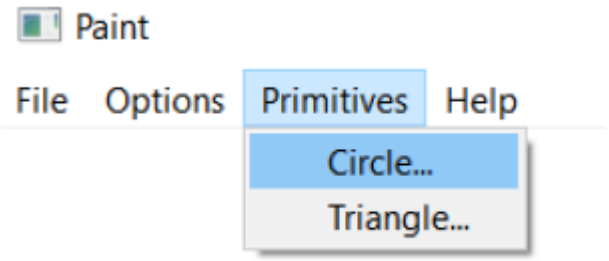
```
    // Can be used to print
QPrinter printer(QPrinter::HighResolution);
// Open printer dialog and print if asked
QPrintDialog printDialog(&printer, this);
if (printDialog.exec() == QDialog::Accepted) {
QPainter painter(&printer);
QRect rect = painter.viewport();
QSize size = image.size();
size.scale(rect.size(), Qt::KeepAspectRatio);
painter.setViewport(rect.x(), rect.y(), size.width(), size.height());
painter.setWindow(image.rect());
painter.drawImage(0, 0, image);
}
#endif // QT_CONFIG(printdialog)
}
```

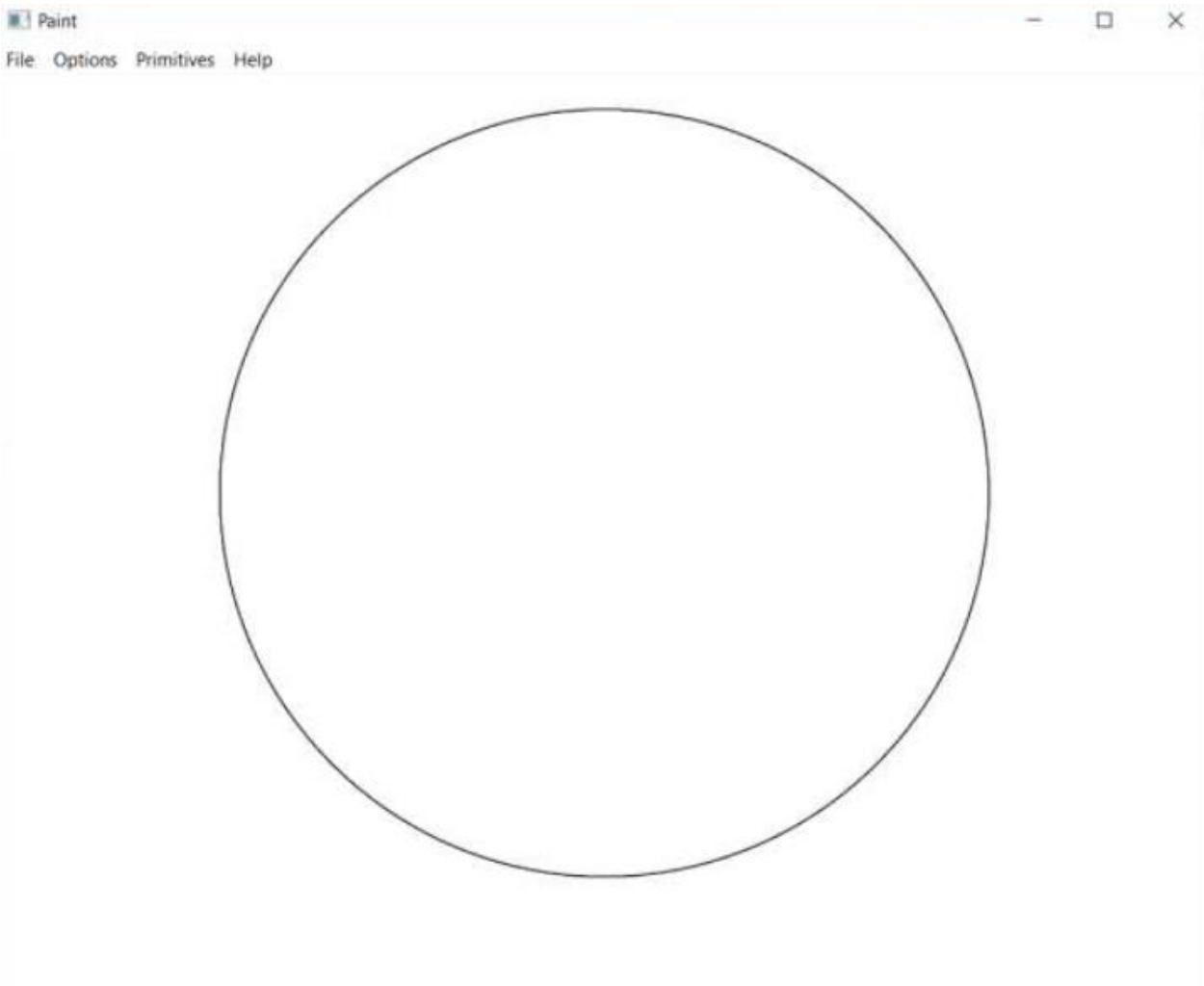
Результат работы:




Select Color
Basic colors
Pick Screen Color
Custom colors
Add to Custom Colors
Hue: 120
Sat: 255
Val: 255
Red: 0
Green: 255
Blue: 0
HTML: #00ff00
OK
Cancel


Paint
File Options Primitives Help


Paint
File Options Primitives Help
Circle...
Triangle...


Paint
File Options Primitives Help