Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №22

по дисциплине

«Языки программирования»

Работу выполнил
студент группы СКБ-222 _____ Д.А. Спиридонов
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2023

Содержание

Постановка задачи

Разработать программу с использованием библиотеки Qt. В окне программы реализовать возможность добавления геометрической фигуры посредством контекстного меню. Реализовать перемещение фигуры в рамках окна при перетаскивании ее курсором мыши. При нажатии на фигуру правой кнопкой мыши выводить контекстное меню позволяющее повернуть или изменить размер фигуры.

Для реализации фигуры использовать класс QWidget и возможности класса QPainter.

Примечание: Не использовать QGraphicsScene и QPainterPath.

По выполненной работе составить отчет по ГОСТ 7.32 включающий описание разработанных классов. Отчет в обязательном порядке должен содержать UML 2.0 диаграмму классов. Отчет включить в состав исходных кодов программы в виде файла формата PDF.1

Основная часть

1 Методы класса MainWindow

1.1 Коструктор класса MainWindow

В конструкторе класса инициаизируются поля класса и создается окно приложения.

1.2 void customContextMenuShow(QPoint pos)

Слот, который вызывается при открытии контекстного меню в окне приложения. Принимает позицию курсора в качестве аргумента.

1.3 void drawSquare()

Метод, который создает квадратную фигуру и отображает ее на экране.

1.4 void drawCircle()

Метод, который создает круглую фигуру и отображает ее на экране.

1.5 void reducePixels()

Метод, который уменьшает размер фигуры на определенное количество пикселей.

1.6 addPixels()

Метод, который увеличивает размер фигуры на определенное количество пикселей.

1.7 void rotateFigure()

Метод, который поворачивает фигуру на определенный угол.

1.8 void deleteFigure()

Метод, который удаляет текущую фигуру.

1.9 bool isPointInside(QPoint point)

Метод, который проверяет, находится ли заданная точка внутри фигуры.

1.10 void paintEvent(QPaintEvent* event)

Метод, который вызывается для перерисовки окна приложения. В этом методе происходит отрисовка фигуры на экране.

1.11 void mousePressEvent(QMouseEvent* event)

Метод, который вызывается при нажатии кнопки мыши. В этом методе обрабатывается событие нажатия кнопки мыши.

1.12 void mouseMoveEvent(QMouseEvent* event)

Метод, который вызывается при перемещении мыши по окну. В этом методе обрабатывается событие перемещения мыши.

1.13 void mouseReleaseEvent(QMouseEvent* event)

Метод, который вызывается при отпускании кнопки мыши. В этом методе обрабатывается событие отпускания кнопки мыши.

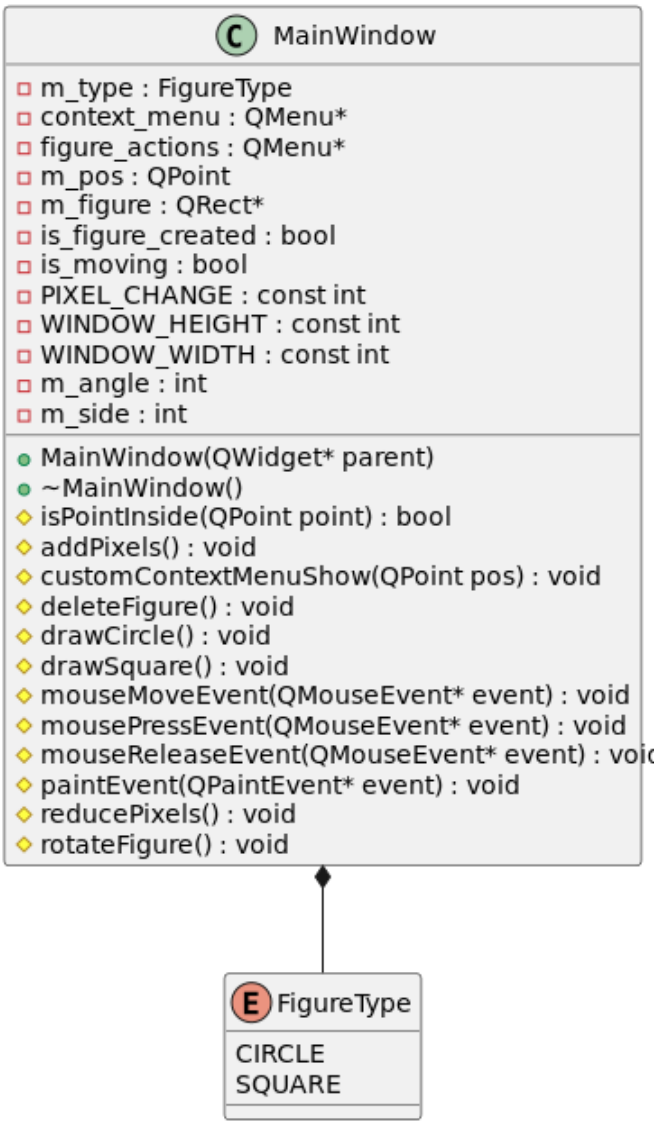
2 Поля класса

Опишем методы класса *MainWindow*.

- Константы WINDOW_HEIGHT, WINDOW_WIDTH и PIXEL_CHANGE определяют высоту и ширину окна приложения, а также количество пикселей, на которое изменяется размер фигуры.
- Переменные m_type, m_angle, m_side, m_pos хранят информацию о текущем типе фигуры, угле поворота, размере стороны фигуры и позиции курсора соответственно.
- Переменная m_figure является указателем на объект класса QRect и используется для хранения информации о текущей фигуре.
- Переменные is_moving и is_figure_created используются для отслеживания состояния перемещения фигуры и создания фигуры соответственно.
- Переменные context_menu и figure_actions являются объектами класса QMenu и используются для создания контекстного меню и действий, связанных с фигурой.

Приложение А

А.1 UML-диаграмма: class MainWindow



Приложение B

B.1 Исходный код *MainWindow.h*

```
#ifndef __MAIN_WINDOW_H__
#define __MAIN_WINDOW_H__

#include <QMessageBox>
#include <QMouseEvent>
#include <QWidget>
#include <QPainter>
#include <QMenu>
#include <QAction>

#include <QDebug>

enum class FigureType { CIRCLE, SQUARE };

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget* parent=nullptr);
    ~MainWindow() = default;

protected slots:
    void customContextMenuShow(QPoint pos);

    void drawSquare();
    void drawCircle();

    void reducePixels();
    void addPixels();
    void rotateFigure();
    void deleteFigure();

protected:

    bool isPointInside(QPoint point);

    void paintEvent(QPaintEvent* event);
    void mousePressEvent(QMouseEvent* event);
    void mouseMoveEvent(QMouseEvent* event);
    void mouseReleaseEvent(QMouseEvent* event);

private:
    const int WINDOW_HEIGHT = 600;
    const int WINDOW_WIDTH  = 600;
    const int PIXEL_CHANGE  = 100;

    FigureType m_type;
    int        m_angle;
    int        m_side;

    QPoint m_pos;
    QRect* m_figure;

    bool is_moving;
    bool is_figure_created;

    QMenu* context_menu;
    QMenu* figure_actions;
};
```

```
#endif // __MAIN_WINDOW_H__
```

B.2 Исходный код *MainWindow.cpp*

```
#include "MainWindow.h"


MainWindow::MainWindow(QWidget* parent)
    : QWidget(parent)
{
    qDebug("MainWindow::MainWindow(QWidget* parent)");

    this->setWindowTitle(tr("QPainter Test"));
    this->setFixedSize(WINDOW_WIDTH, WINDOW_HEIGHT);

    m_angle = 0;
    m_side = 100;
    is_moving = false;
    is_figure_created = false;
    m_pos = QPoint(
        (WINDOW_WIDTH - m_side)/2,
        (WINDOW_HEIGHT - m_side)/2
    );
    m_figure = new QRect(m_pos.x(), m_pos.y(), m_side, m_side);

    context_menu = new QMenu(this);

    QAction* drawSquareAction = new QAction(tr("Нарисовать квадрат"), this);
    QAction* drawCircleAction = new QAction(tr("Нарисовать круг"), this);

    connect(drawSquareAction, &QAction::triggered, this, &MainWindow::drawSquare);
    connect(drawCircleAction, &QAction::triggered, this, &MainWindow::drawCircle);

    context_menu->addAction(drawSquareAction);
    context_menu->addAction(drawCircleAction);

    figure_actions = new QMenu(this);

    QAction* reducePixelsAction = new QAction(tr("Уменьшить на 100 пикселей"), this);
    QAction* addPixelsAction = new QAction(tr("Увеличить на 100 пикселей"), this);
    QAction* rotateFigureAction = new QAction(tr("Повернуть фигуру на 30 градусов"), this);
    QAction* deleteFigureAction = new QAction(tr("Удалить фигуру"), this);

    connect(reducePixelsAction, SIGNAL(triggered()), this, SLOT(reducePixels()));
    connect(addPixelsAction,    SIGNAL(triggered()), this, SLOT(addPixels()));
    connect(rotateFigureAction, SIGNAL(triggered()), this, SLOT(rotateFigure()));
    connect(deleteFigureAction, SIGNAL(triggered()), this, SLOT(deleteFigure()));

    figure_actions->addAction(reducePixelsAction);
    figure_actions->addAction(addPixelsAction);
    figure_actions->addAction(rotateFigureAction);
    figure_actions->addAction(deleteFigureAction);

    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, SIGNAL(customContextMenuRequested(QPoint)), this, SLOT(customContextMenuShow(QPoint)));
}


void MainWindow::drawSquare()
{
    qDebug("MainWindow::drawSquare(QPaintEvent* event)\n");
```

```
    QPainter painter(this);

    is_figure_created = true;
    m_type = FigureType::SQUARE;
    this->update();
}

void MainWindow::drawCircle()
{
    qDebug("MainWindow::drawCircle(QPaintEvent* event)\n");

    is_figure_created = true;
    m_type = FigureType::CIRCLE;
    this->update();
}

void MainWindow::paintEvent(QPaintEvent* event)
{
    qDebug("MainWindow::paintEvent(QPaintEvent* event)\n");
    if(is_figure_created)
    {
        QPainter painter(this);

        painter.setRenderHint(QPainter::Antialiasing);

        if(m_type == FigureType::CIRCLE)
        {
            painter.translate(m_figure->center());
            painter.rotate(m_angle);
            painter.translate(-m_figure->center());

            painter.setBrush(Qt::blue);
            painter.setPen(Qt::black);
            painter.drawEllipse(*m_figure);
        }
        else if(m_type == FigureType::SQUARE)
        {
            painter.translate(m_figure->center());
            painter.rotate(m_angle);
            painter.translate(-m_figure->center());

            painter.setBrush(Qt::green);
            painter.setPen(Qt::black);
            painter.drawRect(*m_figure);
        }
    }
}


bool MainWindow::isPointInside(QPoint point)
{
    qDebug("MainWindow::isPointInside(QPoint point)");

    if(m_type == FigureType::CIRCLE) {
        return ((point.x() >= m_figure->x()) &&
                (point.y() >= m_figure->y()) &&
                (point.x() <= m_figure->x() + m_side) &&
                (point.y() <= m_figure->y() + m_side));
    }
    else {
        return (abs(m_figure->center().x() - point.x()) <=  m_side/2 &&
                abs(m_figure->center().y() - point.y()) <=  m_side/2);
    }
}
```

```
void MainWindow::mousePressEvent(QMouseEvent *event)
{
    if(is_figure_created && isPointInside(event->pos()) &&  (event->button() & Qt::LeftButton))
    {
        is_moving = true;
        m_pos = event->pos();
    }
}

void MainWindow::mouseMoveEvent(QMouseEvent* event)
{
    qDebug("void MainWindow::mouseMoveEvent(QMouseEvent* event)");

    if(is_moving) {
        int x_shift = event->pos().x() - m_pos.x();
        int y_shift = event->pos().y() - m_pos.y();
        m_pos = event->pos();

        m_figure->moveTo(m_figure->x() + x_shift,
                         m_figure->y() + y_shift);
        this->update();
    }
}


void MainWindow::mouseReleaseEvent(QMouseEvent *event)
{
    if ((event->button() & Qt::LeftButton) && is_moving) {
        is_moving = false;
    }
}

void MainWindow::customContextMenuShow(QPoint pos)
{
    QPoint globalPos = mapToGlobal(pos);

    qDebug("void MainWindow::customMenuRequested(QPoint pos)");

    if(!is_figure_created) {
        context_menu->exec(globalPos);
    }
    else if(is_figure_created && isPointInside(pos)){
        figure_actions->exec(globalPos);
    }
}


void MainWindow::reducePixels()
{
    qDebug("void MainWindow::reducePixels()");

    if(m_side >= 2 * PIXEL_CHANGE) {
        m_side -= PIXEL_CHANGE;
        m_figure->setWidth(m_side);
        m_figure->setHeight(m_side);
    }
    else {
        QMessageBox* warning = new QMessageBox(QMessageBox::Warning,
                                               "Предупреждение!",
                                               "Фигуру нельзя сделать еще меньше!",
                                               QMessageBox::NoButton);
        warning->exec();
    }
    this->update();
```

```
}


void MainWindow::addPixels()
{
    qDebug("void MainWindow::addPixels()");

    if(m_side < WINDOW_HEIGHT / 2 - PIXEL_CHANGE) {
        m_side += PIXEL_CHANGE;
    }
    else {
        QMessageBox* warning = new QMessageBox(QMessageBox::Warning,
                                               "Предупреждение!",
                                               "Фигуру нельзя сделать еще больше!",
                                               QMessageBox::NoButton);
        warning->exec();
    }
    m_figure->setWidth(m_side);
    m_figure->setHeight(m_side);
    this->update();
}


void MainWindow::rotateFigure()
{
    qDebug("void MainWindow::rotate()");

    m_angle = (m_angle + 30) % 360;
    this->update();
}


void MainWindow::deleteFigure()
{
    qDebug("void MainWindow::deleteFigure()");

    is_figure_created = false;
    m_angle = 0;
    this->update();
}
```

B.3 Исходный код *main.cpp*

```
#include <QApplication>
#include "MainWindow.h"


int main(int argc, char** argv)
{
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
```