Правительство Российской Федерации

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

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №23

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 **3**

Основная часть **4**

1 Переименованы методы drawSquare() и drawCircle() 4

Переименованы методы drawSquare() и drawCircle() 4

2 Новые методы isPointInsideFigure() и isPointOnTheCorner() . . . 4

3 Метод paintEvent() 4

4 Новый аттрибут класса 4

Приложение A **5**

A.1 UML-диаграмма класса MainWindow 5

Приложение B **6**

B.1 Исходный код MainWindow.h 6

B.2 Исходный код MainWindow.cpp 8

B.3 Исходный код main.cpp 12

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 за границу курсором мыши. При нажатии на границу фигуры правой кнопкой мыши выводить контекстное меню позволяющее повернуть ее или изменить размер. При наведении курсора на внутреннюю часть фигуры подсвечивать ее полупрозрачным цветом.

Для реализации фигуры использовать класс QWidget и возможности классов QPainter и QPainterPath.

По выполненной работе составить отчет по ГОСТ 7.32, в котором отразить изменения относительно лабораторной работы №2. Отчет в обязательном порядке должен содержать UML 2.0 диаграмму классов. Отчет включить в состав исходных кодов программы в виде файла формата PDF.

Основная часть

Нижу будут описаны изменения в классе MainWindow.

1 Переименованы методы drawSquare() и drawCircle()

Методы drawSquare() и drawCircle() были переименованы в addSquare() и addCircle(), соответственно. Это было сделано для того, чтобы улучшить читаемость кода (фоктически, эти методы отвечают не за отризовку фигур, а за их добавление на экран).

2 Новые методы isPointInsideFigure() и isPointOnTheCorner()

Был убран метод isPointInside(). За место него были добавлены 2 метода:

- isPointInsideFigure() - этот метод проверяет, находиться курсор внутри фигуры или нет;
- sPointOnTheCorner() - этот метод проверяет, находиться курсор на границе нарисованной фигуры или нет. Также используется гиперпараметр EPS, который позволяет задать область, в пределах которой будет считаться попадание на границу фигуры;

3 Метод paintEvent()

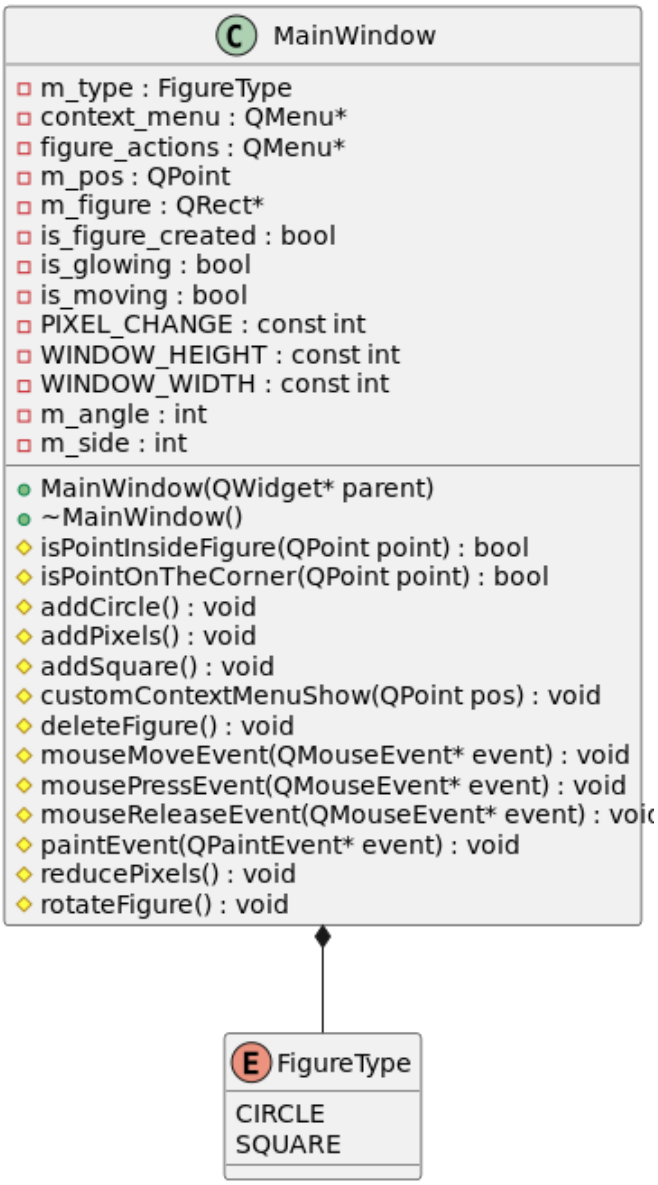
Была изменена логика метода paintEvent(). Теперь он подсвечивает фигуру при наведении на нее курсором.

4 Новый аттрибут класса

Был добавлен атрибут класса is_glowing.

Приложение А

А.1 UML-диаграмма класса MainWindow



Приложение B

B.1 Исходный код MainWindow.h

```
#ifndef __MAIN_WINDOW_H__
#define __MAIN_WINDOW_H__

#include <cmath>
#include <random>

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

#define EPS 10

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 addSquare();
    void addCircle();

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

protected:
    bool isPointInsideFigure(QPoint point);
    bool isPointOnTheCorner(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 = 0;
    int m_side  = 100;

    QPoint m_pos;
    QRect* m_figure;

    bool is_glowing = false;
```

```
    bool is_moving = false;
    bool is_figure_created = false;

    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)
{
    this->setFixedSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    this->setMouseTracking(true);

    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* addSquareAction = new QAction(tr("Нарисовать квадрат"), this);
    QAction* addCircleAction = new QAction(tr("Нарисовать круг"), this);

    connect(addSquareAction, &QAction::triggered, this, &MainWindow::addSquare);
    connect(addCircleAction, &QAction::triggered, this, &MainWindow::addCircle);

    context_menu->addAction(addSquareAction);
    context_menu->addAction(addCircleAction);

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

void MainWindow::addCircle()
{
    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);
        painter.setPen(QColor(0, 128, 128));
        if (is_glowing)
        {
            painter.setBrush(QColor(02, 205, 170, 30));
        }
        else
        {
            painter.setBrush(Qt::NoBrush);
        }

        if(m_type == FigureType::CIRCLE)
        {
            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.drawRect(*m_figure);
        }
    }
}


bool MainWindow::isPointOnTheCorner(QPoint point)
{
    if(m_type == FigureType::CIRCLE)
    {
        return ((4 * std::pow(point.x() - m_figure->center().x(), 2) + 4 * std::pow(point.y() - m_figure->ce
               ((4 * std::pow(point.x() - m_figure->center().x(), 2) + 4 * std::pow(point.y() - m_figure->ce

    }
    else if(m_type == FigureType::SQUARE)
    {
        return  (2 * abs(m_figure->center().x() - point.x()) <=  m_side + EPS) &&
                (2 * abs(m_figure->center().x() - point.x()) >=  m_side - EPS) &&
                (2 * abs(m_figure->center().y() - point.y()) <=  m_side + EPS) &&
                (2 * abs(m_figure->center().y() - point.y()) >=  m_side - EPS);
    }
    else return false;
}


bool MainWindow::isPointInsideFigure(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 if(m_type == FigureType::SQUARE)
    {
        return  (2 * abs(m_figure->center().x() - point.x()) <  m_side) &&
                (2 * abs(m_figure->center().y() - point.y()) <  m_side);
```

```
    }
    else return false;
}


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


void MainWindow::mouseMoveEvent(QMouseEvent* event)
{
    is_glowing = isPointInsideFigure(event->pos());

    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)
{
    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 && isPointInsideFigure(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()");
    if(m_type == FigureType::SQUARE)
    {
        m_angle = (m_angle + 30) % 360;
        this->update();
    }
    else if(m_type == FigureType::CIRCLE)
    {
        QString text[2] = {"Круг нельзя вращать!", "*Вращайте барабан*"};
        QMessageBox* warning = new QMessageBox(QMessageBox::Warning,
                                               "Предупреждение!",
                                               text[rand() % 2],
                                               QMessageBox::NoButton);
        warning->exec();
    }
}


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();
}
```