Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра “Компьютерная безопасность”

Отчёт к лабораторной работе №2 (2 год)

по дисциплине

«Языки программирования»

Работу выполнил

Студент группы СКБ221 _____ М. И. Нугманов

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*.

1. Алгоритм решения

Для решения поставленной задачи был создан наследник класса *QWidget*, в котором реализованы методы для Создания фигур, их Поворота, Изменения размера, а также вывода Контекстного меню, позволяющего Пользователю выполнять вышеперечисленные действия.

2. Реализация

В первую очередь, был создан класс Widget – наследник класса QWidget, содержащий в себе необходимые методы для работы с фигурами. Отдельна была создана структура Shape, содержащая информацию об фигуре. Также реализовано контекстное меню при помощи соединения customContextMenuRequested со слотом ShowContextMenu. Меню позволяет создавать и изменять фигуры. Подробнее с реализацией программы можно ознакомиться в *Листинге 1*.

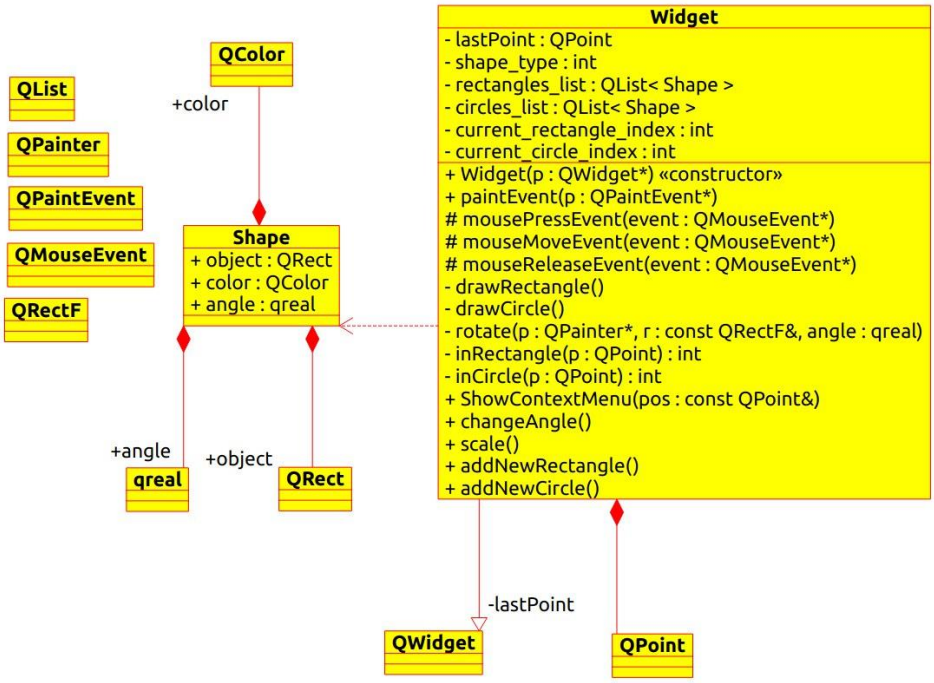

Рисунок 1 - UML-диаграмма widget.h

2.1 Методы класса

2.1.1 Widget(QWidget* parent)

Конструктор класса *Widget*. Устанавливает политику контекстного меню для виджета и соединяет сигнал *customContextMenuRequested* со слотом *ShowContextMenu*.

2.1.2 paintEvent()

Событие перерисовки виджета. Вызывает методы *drawRectangle()* и *drawCircle()* для отображения фигур на виджете.

2.1.3 drawRectangle()

Отрисовывает все прямоугольники из списка *rectangles_list*. Использует объект *QPainter* для рисования прямоугольников с учетом их цвета, угла и координат.

2.1.4 drawCircle()

Отрисовывает все круги из списка *circles_list*. Использует объект *QPainter* для рисования кругов с учетом их цвета, угла и координат.

2.1.5 InRectangle(QPoint clickPoint)

Проверяет, находится ли точка *clickPoint* внутри какого-либо прямоугольника из списка *rectangles_list*. Возвращает индекс прямоугольника, если точка находится внутри, или -1, если не находится.

2.1.6 InCircle(QPoint clickPoint)

Проверяет, находится ли точка *clickPoint* внутри какого-либо круга из списка *circles_list*. Возвращает индекс круга, если точка находится внутри, или -1, если не находится.

2.1.7 mousePressEvent(QMouseEvent *event)

Обрабатывает событие нажатия кнопки мыши. Определяет, на какой тип фигуры указывает точка *clickPoint* и сохраняет этот тип в *shape_type*. Запоминает индекс выбранной фигуры, если таковая имеется.

2.1.8 ShowContextMenu(const QPoint &pos)

Показывает контекстное меню, связанное с выбранной фигурой (*rectangles_list* или *circles_list*) или меню для создания новой фигуры, если фигура не выбрана. Добавляет действия для вращения и изменения размера фигур в меню.

2.1.9 mouseMoveEvent(QMouseEvent *event)

Обрабатывает событие перемещения мыши с зажатой кнопкой. Перемещает выбранную фигуру (прямоугольник или круг) на величину изменения координат указателя мыши.

2.1.10 mouseReleaseEvent(QMouseEvent *event)

Обрабатывает событие отпускания кнопки мыши. Сбрасывает *shape_type* после того, как кнопка мыши была отпущена.

2.1.11 addNewRectangle()

Добавляет новый прямоугольник в список *rectangles_list* с начальными параметрами, такими как координаты, размер и случайный цвет. Вызывает *update()*, чтобы обновить отображение.

2.1.12 addNewCircle():

Добавляет новый круг в список *circles_list* с начальными параметрами, такими как координаты, размер и случайный цвет. Вызывает *update()*, чтобы обновить отображение.

2.1.13 changeAngle():

Изменяет угол поворота выбранной фигуры (прямоугольника или круга) на 30 градусов. Вызывает *update(),* чтобы обновить отображение.

2.1.14rotate(QPainter* painter, const QRectF& r, qreal angle)

Производит вращение фигуры, находящейся внутри *painter*, вокруг ее центра на заданный угол. Вызывает *update(),* чтобы обновить отображение.

2.1.15 scale()

Увеличивает размер выбранной фигуры (прямоугольника или круга), добавляя 5 пикселей к ее ширине и высоте. Вызывает *update(),* чтобы обновить отображение.

2.2 Контекстное меню

2.2.1 Нажатие ПКМ вне границ фигуры

При нажатии ПКМ вне границ фигуры появляется контекстное меню с возможность добавить новый Прямоугольник или новый Круг.

2.2.2 Нажатие ПКМ внутри границ фигуры

При нажатии ПКМ внутри границ фигуры появляется контекстное меню с возможностью повернуть выбраннную фигуру на 30 градусов по часовой стрелке или увеличить Высоту и Ширину выбранной фигуры на 5 пикселей.

3. Тестирование программы

На рисунках 2-6 приведены результаты тестирования программы:

Рисунок 2 - Начальный вид программы при ее запуске

Рисунок 3 - Отображение контекстного меню для создания новой фигуры

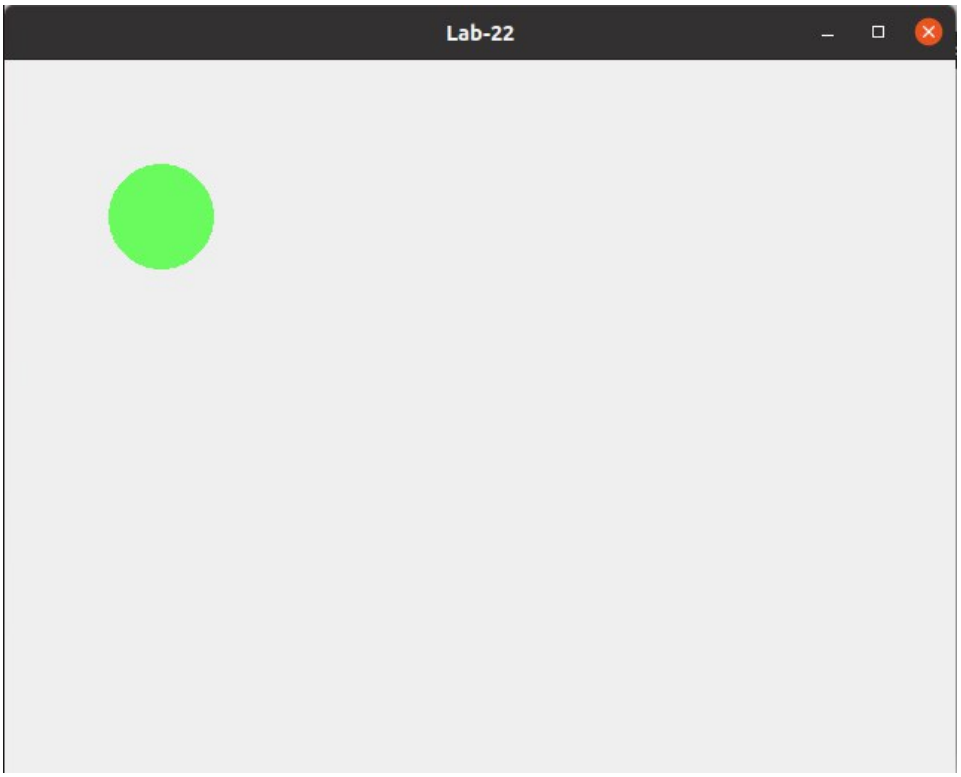

Рисунок 5 - Результат нажатия на поле "Добавить круг"

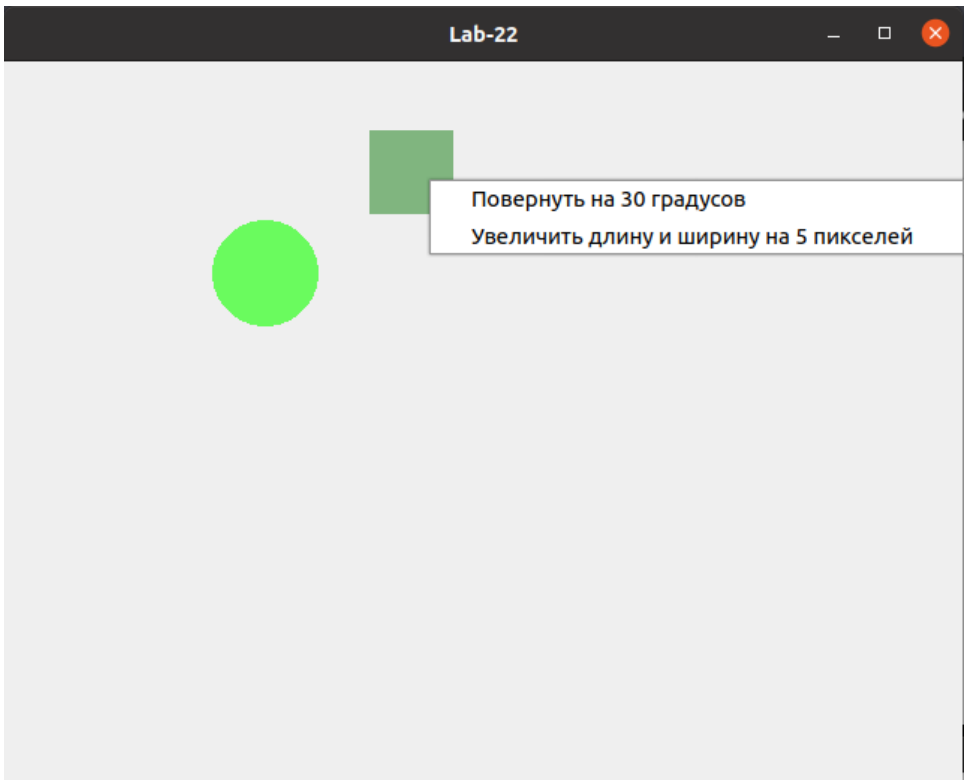

Рисунок 4 - Отображение контекстного меню для изменения выбранной фигуры

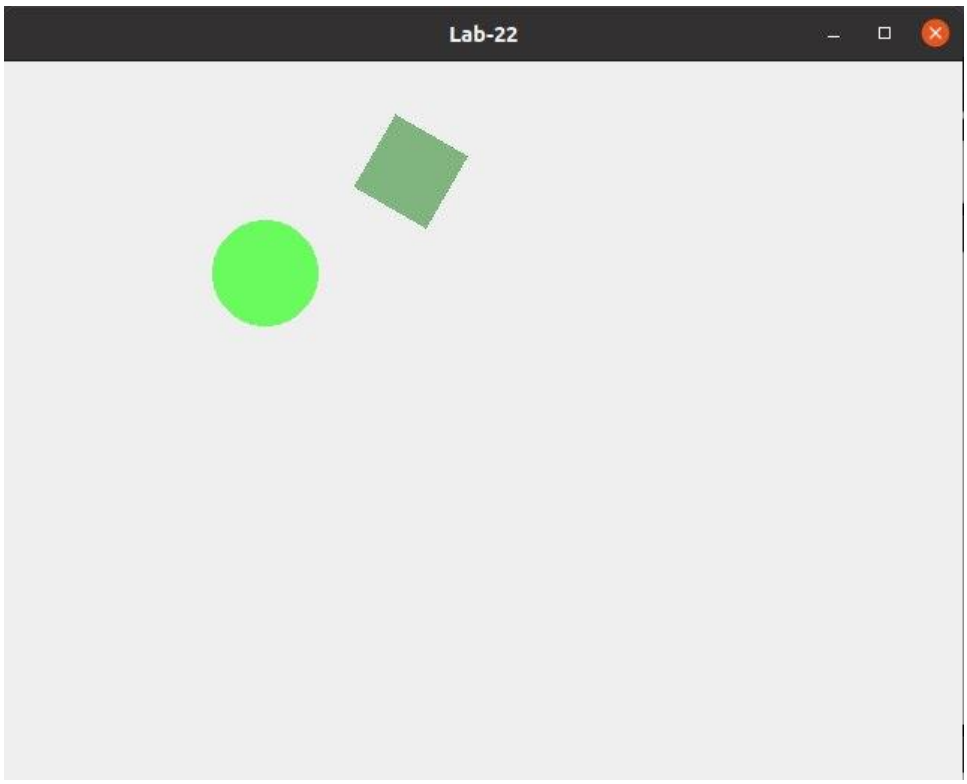

Рисунок 6 - Результат нажатия на поле "Повернуть на 30 градусов"

Листинг 1

1.1 Исходный код widgets.h

```
#include <QWidget>
#include <QPainter>
#include <QMouseEvent>

struct Shape {
	QRect  object;
	QColor color;
	qreal angle = 0;
};

class Widget: public QWidget {
	Q_OBJECT
public:
	Widget(QWidget* p = nullptr);
	void paintEvent(QPaintEvent *p = nullptr);
protected:
	void mousePressEvent(QMouseEvent *event) override;
	void mouseMoveEvent(QMouseEvent *event) override;
	void mouseReleaseEvent(QMouseEvent *event) override;
private:
	void drawRectangle();
	void drawCircle();
	void rotate(QPainter* p, const QRectF& r, qreal angle);
	int inRectangle(QPoint p);
	int inCircle(QPoint p);
	QPoint lastPoint;
	int shape_type;
	QList<Shape> rectangles_list = {};
	QList<Shape> circles_list    = {};
	int current_rectangle_index = 0;
	int current_circle_index    = 0;
public slots:
	void ShowContextMenu(const QPoint &pos);
	void changeAngle();
	void scale();
	void addNewRectangle();
	void addNewCircle();
};
```

1.2 Исходный код widget.cpp

```
#include "widget.h"
#include <QPainterPath>
#include <QDebug>
#include <QMenu>
#include <QAction>
#include <QRandomGenerator>

Widget::Widget(QWidget *parent): QWidget(parent) {
    this->setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, SIGNAL(customContextMenuRequested(const QPoint &)),
        this, SLOT(ShowContextMenu(const QPoint &)));
}
void Widget::paintEvent(QPaintEvent *) {
    drawRectangle();
    drawCircle();
}
void Widget::drawRectangle() {
    QPainter painter(this);
    for(const Shape &shape : rectangles_list) {
            rotate(&painter, shape.object, shape.angle);
            painter.setBrush(shape.color);
            painter.setPen(shape.color);
            painter.drawRect(shape.object);
    }
}
void Widget::drawCircle() {
    QPainter painter(this);
    for(const Shape &shape : circles_list) {
            rotate(&painter, shape.object, shape.angle);
            painter.setBrush(shape.color);
            painter.setPen(shape.color);
            painter.drawEllipse(shape.object);
    }
}
int Widget::inRectangle(QPoint clickPoint) {
    for (int i = 0; i < rectangles_list.size(); ++i) {
        if (clickPoint.x() >= rectangles_list[i].object.x() &&
            clickPoint.x() <= (rectangles_list[i].object.x()
rectangles_list[i].object.width()) &&
            clickPoint.y() >= rectangles_list[i].object.y() &&
```

```
            clickPoint.y() <= (rectangles_list[i].object.y() +
rectangles_list[i].object.height())) {
                return i;
            }
        }
        return -1;
}
int Widget::inCircle(QPoint clickPoint)
{
        for (int i = 0; i < circles_list.size(); ++i) {
            if (clickPoint.x() >= circles_list[i].object.x() &&
                clickPoint.x() <= (circles_list[i].object.x() +
circles_list[i].object.width()) &&
                clickPoint.y() >= circles_list[i].object.y() &&
                clickPoint.y() <= (circles_list[i].object.y() +
circles_list[i].object.height())) {
                return i;
            }
        }
        return -1;
}

void Widget::mousePressEvent(QMouseEvent *event)
{
        QPoint clickPoint = event->pos();
        if(inRectangle(clickPoint) != -1) {
            shape_type = 1;
            current_rectangle_index = inRectangle(clickPoint);
        }
        else if(inCircle(clickPoint) != -1) {
            shape_type = 2;
            current_circle_index = inCircle(clickPoint);
        }
        else
            shape_type = 0;

        if(shape_type)
        {
            if (event->button() == Qt::LeftButton) {
                    lastPoint = clickPoint;
            }
            else if(event->button() == Qt::RightButton )
```

```
        {
            qDebug() << "Right button is clicked!";
        }
    }
}

void Widget::ShowContextMenu(const QPoint &pos)
{
    if(inRectangle(pos) != -1)
        shape_type = 1;
    else if(inCircle(pos) != -1)
        shape_type = 2;
    else
        shape_type = 0;

    if(shape_type)
    {
        QMenu contextMenu(tr("Контекстное меню"), this);
        QAction action1("Повернуть на 30 градусов", this);
        connect(&action1, SIGNAL(triggered()), this, SLOT(changeAngle()));

        QAction action2("Увеличить длину и ширину на 5 пикселей", this);
        connect(&action2, SIGNAL(triggered()), this, SLOT(scale()));

        contextMenu.addAction(&action1);
        contextMenu.addAction(&action2);
        contextMenu.exec(mapToGlobal(pos));
    }
    else {
        QMenu contextMenu(tr("Контекстное меню"), this);
        QAction action3("Добавить прямоугольник", this);
        connect(&action3, SIGNAL(triggered()), this, SLOT(addNewRectangle()));
        QAction action4("Добавить круг", this);
        connect(&action4, SIGNAL(triggered()), this, SLOT(addNewCircle()));

        contextMenu.addAction(&action3);
        contextMenu.addAction(&action4);
        contextMenu.exec(mapToGlobal(pos));
    }

}
```

```
void Widget::mouseMoveEvent(QMouseEvent *event) {
    if (shape_type) {
        int delta_x = event->pos().x() - lastPoint.x();
        int delta_y = event->pos().y() - lastPoint.y();
        lastPoint = event->pos();
        switch (shape_type)
        {
        case 1:

rectangles_list[current_rectangle_index].object.moveTo(rectangles_list[current_rectan
gle_index].object.x() + delta_x, rectangles_list[current_rectangle_index].object.y()
+ delta_y);
            break;
        case 2:
circles_list[current_circle_index].object.moveTo(circles_list[current_circle_index].o
bject.x() + delta_x, circles_list[current_circle_index].object.y() + delta_y);
                break;

        default:
            break;
        }
        update();
    }
}

void Widget::mouseReleaseEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton && shape_type) {

        shape_type = 0;
    }
}

void Widget::addNewRectangle() {
    Shape newRectangle;
    newRectangle.object = QRect(10, 10, 50, 50);
    newRectangle.color = QColor(QRandomGenerator::global()->bounded(256),
QRandomGenerator::global()->bounded(256), QRandomGenerator::global()->bounded(256));
    rectangles_list.append(newRectangle);
    update();
}
```

```
void Widget::addNewCircle() {
    Shape newCircle;
    newCircle.object = QRect(70, 70, 70, 70);
    newCircle.color = QColor(QRandomGenerator::global()->bounded(256),
QRandomGenerator::global()->bounded(256), QRandomGenerator::global()->bounded(256));
    circles_list.append(newCircle);
    update();
}

void Widget::changeAngle() {
    switch (shape_type) {
    case 1:
        rectangles_list[current_rectangle_index].angle += 30;
        break;
    case 2:
        circles_list[current_circle_index].angle += 30;
        break;
    default:
        break;
    }
}

void Widget::rotate(QPainter* painter, const QRectF& r, qreal angle) {
    painter->translate(r.center());
    painter->rotate(angle);
    painter->translate(-r.center());
    update();
}
void Widget::scale() {
    switch (shape_type)
    {
    case 1:

rectangles_list[current_rectangle_index].object.setWidth(rectangles_list[current_rect
angle_index].object.width()+5);

rectangles_list[current_rectangle_index].object.setHeight(rectangles_list[current_rec
tangle_index].object.height()+5);
        break;
    case 2:
```

```
circles_list[current_circle_index].object.setWidth(circles_list[current_circle_index]
.object.width()+5);

circles_list[current_circle_index].object.setHeight(circles_list[current_circle_index
].object.height()+5);
        break;
    default:
        break;
    }
}
```

1.3 Исходный код main.cpp

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```