Правительство Российской Федерации

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

Кафедра компьютерной безопасности

Отчёт к лабораторной работе №3

по дисциплине

«Языки программирования»

Работу выполнил

студент группы СКБ223 _____ А. Я. Сорочайкин
подпись, дата

Работу проверил _____ С. А. Булгаков
подпись, дата

Москва 2023

Содержание

Постановка задачи 3
1. Алгоритм решения задачи 4
2. Решение задачи 5
3. Тестирование программы 8
Приложение А 10

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.

1. Алгоритм решения задачи

Для решения задачи, был разработан основной класс Widget, а также класс треугольника, который можно вращать, масштабировать и перемещать. В основной функции создается объект класса Widget, на котором можно добавить экземпляры треугольника.

2. Решение задачи

Класс Widget разработан на основе класса QWidget. Класс треугольника также разработан на основе класса QWidget.

UML диаграмма зависимостей классов представлена на рисунке 1.

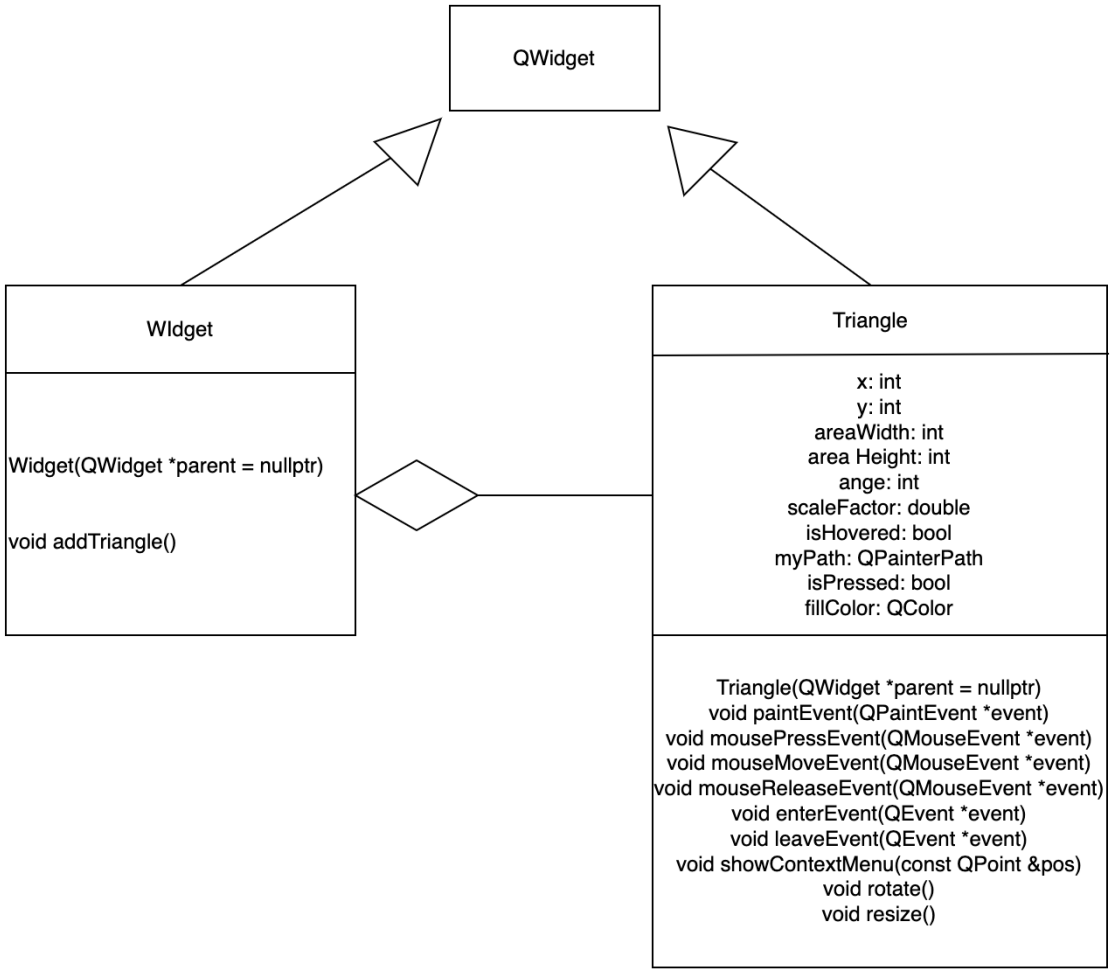

Рис. 1

2.1 Класс Triangle

Класс состоит из, полей int x, int y, отвечающих за координаты середины виджета, полей areaWidtgh и areaHeight, отвечающих за размер виджета фигуры, полей double scaleFactor и int angle – отвечающих за размер и наклон виджета соответственно, поля bool isHovered, отвечающего за нахождение курсора мыши над фигурой конструктора, поля QPainterPath myPath, отвечающего за путь, используемый для отрисовки фигуры, поля bool isPressed, отвечающего за то, нажата ли в данный момент левая кнопка мыши на виджете, поля QColor fillColor, отвечающего за цвет заливки фигуры, защищенных методов void paintEvent(QPaintEvent *event),

void mousePressEvent(QMouseEvent *event), void mouseMoveEvent(QMouseEvent *event), void mouseReleaseEvent(QMouseEvent *event), void enterEvent(QEvent *event), void leaveEvent(QEvent *event), а также приватных слотов void showContextMenu(const QPoint &pos), void rotate(), void resizeFigure().

2.1.1 Конструктор

Инициализирует поля класса, устанавливает политику контекстного меню, позволяющую пользователю определить собственное контекстное меню для данного виджета. Также, соединяем сигнал customContextMenuRequested и слотом showContextMenu

2.1.2 Метод paintEvent

Отвечает за отрисовку треугольника на виджете.

2.1.3 Метод mousePressEvent

При нажатии на виджет, устанавливает координаты нажатия. А также меняет значение поля isPressed на true.

2.1.4 Метод mouseMoveEvent

При перемещении меняет координаты фигуры на новые.

2.1.5 Метод mouseReleaseEvent(QMouseEvent *event)

Меняет значение поля isPressed на false.

2.1.6 Метод enterEvent(QEvent *event)

Меняет значение поля isHovered на true и запускает процесс обновления виджета.

2.1.7 Метод leaveEvent(QEvent *event)

Меняет значение поля isHovered на false и запускает процесс обновления виджета.

2.1.8 Слот showContextMenu

Создаем контекстное меню фигуры, с возможными действиями: rotate и resize, также, соединяет поля контекстного меню с соответствующими функциями.

2.1.9 Слот rotate

Отвечает за поворот фигуры, на указанную пользователем в диалоговом окне, величину.

2.1.10 Слот resizeFigure

Отвечает за изменение размеров фигуры, до указанных пользователем в диалоговом окне, размеров.

2.2. Класс Widget

Класс состоит из конструктора, а также приватного слота addTriangle.

2.2.1 Конструктор

Устанавливает политику контекстного меню, позволяющую пользователю определить собственное контекстное меню для данного виджета.

2.2.2 Слот addTriangle

Создает и отображает экземпляр соответствующего класса геометрической фигуры Triangle.

3. Тестирование программы

Общий вид окна программы с отображенным контекстным меню выбора геометрической фигуры показан на рисунке 2.

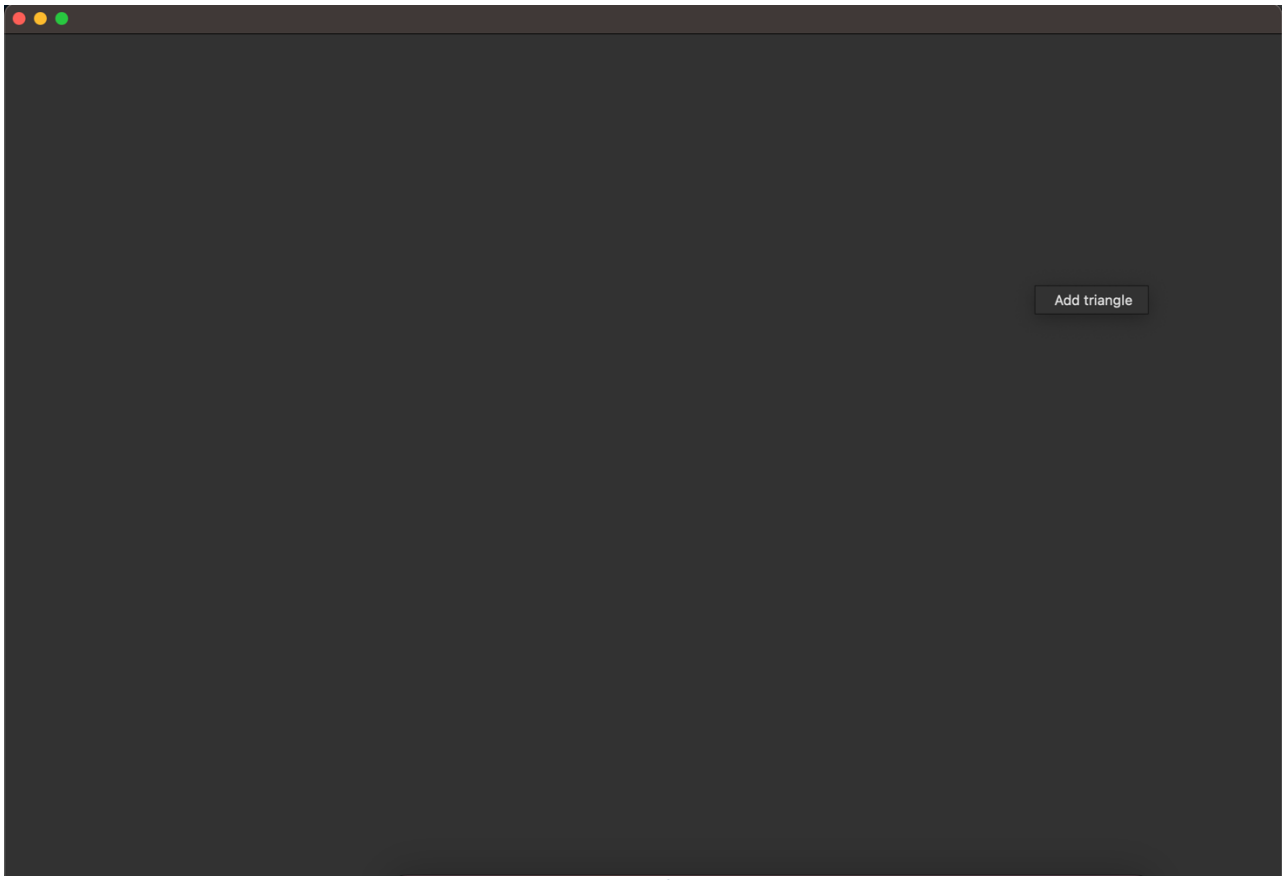

Рис. 2

Примеры отображения фигур и подсветки одной из них при наведении курсора показаны на рисунке 3.

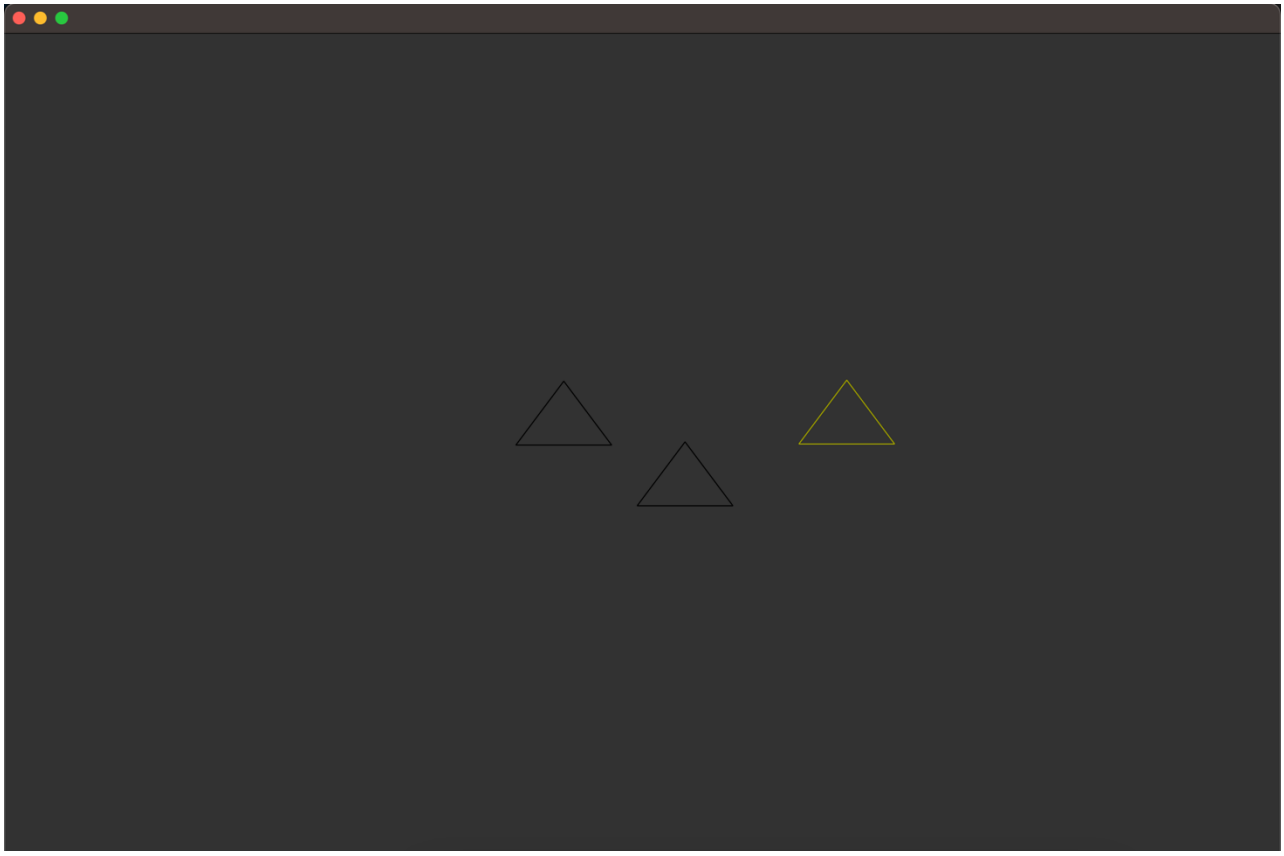

Рис. 3

На рисунке 4 представлено контекстное меню взаимодействия с фигурами, а также результаты поворота и изменения размера фигур.

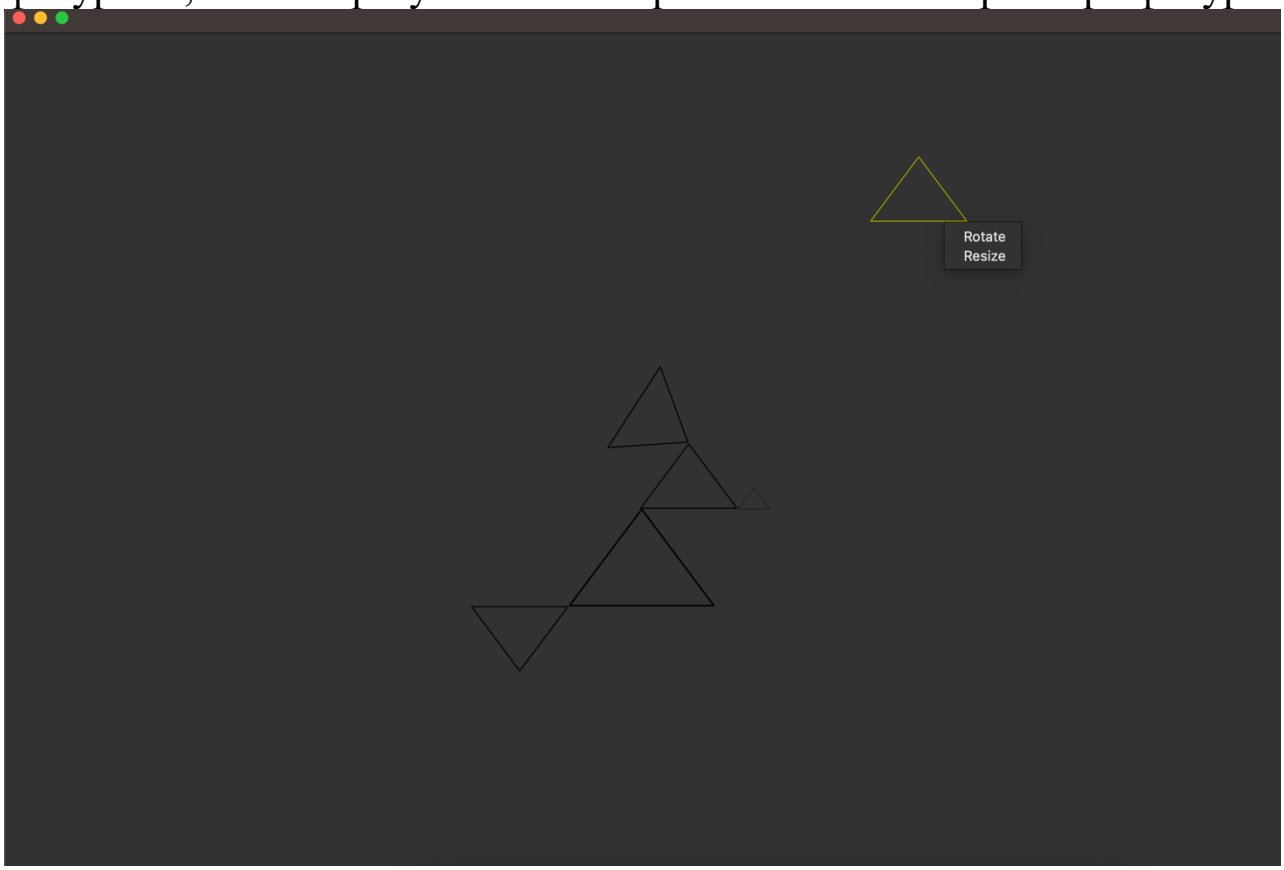

Рис. 4

Приложение А

А1. Исходный код main.cpp

```
#include <QApplication>

#include "widget.h"

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

А2. Исходный код widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget {
  Q_OBJECT

 public:
  Widget(QWidget *parent = nullptr);

 private slots:
  void addTriangle();
};

#endif
```

А3. Исходный код widget.cpp

```
#include "widget.h"

#include <QAction>
#include <QInputDialog>
#include <QMenu>

#include "figure.h"

Widget::Widget(QWidget *parent) : QWidget(parent) {
  setMinimumSize(1200, 800);

  QMenu *contextMenu = new QMenu(this);
  QAction *addTriangleAction = new QAction("Add triangle", this);
  connect(addTriangleAction, &QAction::triggered, this, &Widget::addTriangle);
  contextMenu->addAction(addTriangleAction);

  setContextMenuPolicy(Qt::CustomContextMenu);
  connect(this, &QWidget::customContextMenuRequested,
          [this, contextMenu](const QPoint &pos) {
            contextMenu->popup(mapToGlobal(pos));
          });
}

void Widget::addTriangle() {
  Triangle *shape = new Triangle(this);
  shape->show();
}
```

А4. Исходный код figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <QMouseEvent>
#include <QPainterPath>
#include <QWidget>
```

```
class Triangle : public QWidget {
  Q_OBJECT

  int x, y;
  int areaWidth = 180;
  int areaHeight = 180;
  int angle;
  double scaleFactor;
  bool isHovered = false;
  QPainterPath myPath;
  bool isPressed = false;
  QColor fillColor = QColor(0, 0, 0, 0);

 public:
  Triangle(QWidget *parent = nullptr);

 protected:
  void paintEvent(QPaintEvent *event);
  void mousePressEvent(QMouseEvent *event);
  void mouseMoveEvent(QMouseEvent *event);
  void mouseReleaseEvent(QMouseEvent *event);
  void enterEvent(QEvent *event);
  void leaveEvent(QEvent *event);

 private slots:
  void showContextMenu(const QPoint &pos);
  void rotate();
  void resize();
};

#endif
```

A5. Исходный код figure.cpp

```
#include "figure.h"

#include <QDebug>
#include <QInputDialog>
#include <QMenu>
#include <QPainter>
#include <QPainterPath>

Triangle::Triangle(QWidget *parent) : QWidget(parent) {
  setFixedSize(areaWidth, areaHeight);

  QPainterPath triangle;
  triangle.moveTo(-45, 30);
  triangle.lineTo(0, -30);
  triangle.lineTo(45, 30);
  triangle.closeSubpath();

  myPath.addPath(triangle);

  setMouseTracking(true);
  setContextMenuPolicy(Qt::CustomContextMenu);
  connect(this, &QWidget::customContextMenuRequested, this,
          &Triangle::showContextMenu);
  angle = 0;
  scaleFactor = 1.0;
}

void Triangle::paintEvent(QPaintEvent *event) {
  QPainter painter(this);
  QTransform originalTransform = painter.transform();
  painter.setRenderHint(QPainter::Antialiasing, true);
  painter.translate(areaWidth / 2, areaHeight / 2);
  painter.rotate(angle);
  painter.scale(scaleFactor, scaleFactor);
```

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

  if (isHovered) {
    painter.setBrush(Qt::NoBrush);
    painter.setPen(QPen(QColor(165, 165, 0), 1));
  } else {
    painter.setBrush(QBrush(fillColor));
  }

  myPath = originalTransform.map(myPath);
  painter.drawPath(myPath);
}

void Triangle::mousePressEvent(QMouseEvent *event) {
  x = event->pos().x();
  y = event->pos().y();
  isPressed = true;
}

void Triangle::mouseMoveEvent(QMouseEvent *event) {
  if (isPressed) {
    int deltaX = event->pos().x() - x;
    int deltaY = event->pos().y() - y;
    move(pos().x() + deltaX, pos().y() + deltaY);
  }
  event->ignore();
}

void Triangle::mouseReleaseEvent(QMouseEvent *event) { isPressed = false; }

void Triangle::showContextMenu(const QPoint &pos) {
  QMenu contextMenu(this);
  QAction rotateAction("Rotate", this);
  connect(&rotateAction, &QAction::triggered, this, &Triangle::rotate);
  contextMenu.addAction(&rotateAction);

  QAction resizeAction("Resize", this);
  connect(&resizeAction, &QAction::triggered, this, &Triangle::resize);
  contextMenu.addAction(&resizeAction);

  contextMenu.exec(mapToGlobal(pos));
  isPressed = false;
}

void Triangle::rotate() {
  bool flag;
  int newAngle = QInputDialog::getInt(
      this, "Rotate",
      "Enter the angle of rotation of the Triangle in degrees:", 0, -360, 360,
      1, &flag);
  if (flag) {
    angle = newAngle;
    update();
  }
}

void Triangle::resize() {
  bool flag;
  int newSize = QInputDialog::getInt(
      this, "Resize", "Enter a new Triangle size:", 100, 10, 150, 1, &flag);
  if (flag) {
    scaleFactor = static_cast<double>(newSize) / 100.0;
    update();
  }
}

void Triangle::enterEvent(QEvent *event) {
  isHovered = true;
```

```
  update();
}

void Triangle::leaveEvent(QEvent *event) {
  isHovered = false;
  update();
}
```