Правительство Российской Федерации

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

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №4

по дисциплине

«Языки программирования»

Работу выполнил
студент группы СКБ-201 _____ П.Е. Зильберштейн
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2021

Содержание

Постановка задачи

Разработать графическое приложение с использованием библиотеки Qt.

Общая часть

Приложение состоит из основного окна (наследовать QMainWindow), с панелью инструментов, на которой расположены:

1. Залипающие кнопки с выбором типа фигуры (количество кнопок соответствует количеству фигур в варианте).

2. Кнопка добавления фигуры. Все фигуры поворачиваются относительно центра описывающего их прямоугольника (по умолчанию против часовой стрелки). Параметры фигуры выбираются произвольно в допустимом диапазоне. Если ниодна фигура не выбрана, кнопка добавления неактивна.

3. .Кнопка удаления выделенной фигуры. Если фигура не выделена, кнопка неактивна. Основная часть окна предназначена для размещения фигур (запрещается использовать QGraphicsScene). Фон основной части окна – белый, цвет отрисовки фигур – черный.

При нажатии на фигуре левой кнопкой мыши – фигура выделяется (отрисовывается синим цветом). При нажатии на фигуре правой кнопкой мыши – открывается всплывающее меню. Меню должно содержать пункты «Удалить» и «Изменить». При выборе пункта «Изменить» – открывается модальное диалоговое окно, позволяющее изменить параметры фигуры, угол поворота, направление поворота, а также отображающее площадь и периметр фигуры. При перетаскивании выделенной фигуры она меняет свое положение в рамках окна. При достижении края окна перемещение прекращается. Пересечение с другими фигурами не учитывается.

Задачи

а) Задание на +1 балл. Добавить в выпадающее меню кнопки «Переместить», начинающую процесс перетаскивания фигуры, а также кнопку «Повернуть», открывающее диалоговое окно с ползунком -180, 0, 180 градусов (текущий поворот фигуры отображается в реальном времени).

б) Задание на +1 балл. Создать выпадающее меню для основной части окна и добавить в него кнопки «Удалить все» и «Удалить пересекающиеся».

в) Задание на +2 балла. При перемещении фигур учитывать пересечения с другими фигурами. Добавить в выпадающее меню основной части окна кнопку «Уместить», изменяющую поворот и положение всех фигур, а при большом их количестве также и размер (на минимальноезначение), так чтобы все фигуры разместились в окне.

1 Алгоритм решения задачи

1.1 Задача на 6 баллов

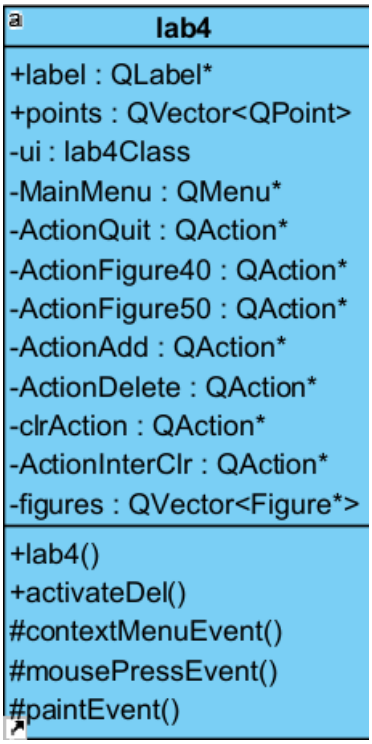

Рис. 1. UML-диаграмма класса lab4.

Для решения данной задачи был разработан класс lab4, UML диаграмма которого приведена на рис. 1, содержащий закрытые поля MainMenu типа указатель на QMenu, отвечающее за контекстное меню главного окна; ActionQuit, ActionFigure40, ActionFigure50, ActionAdd, ActionDelete, clrAction типа указатель на QAction, отвечающие за кнопки на панеле инструментов и в контестном меню; а также figures - вектор указателей на Figure, хранящий фигуры, необзодимые для отрисовки.
Также класс содержит:

- конструктор с параметром;
- функция activateDel, делающая кнопку "Удалить"в панели инструментов активной;
- Слоты Add, Delete, FullClr, отвечающие за добавление, удаление выбранного элемента и полное очищение главного окна соответсвенно;
- перегрузка функции contextMenuEvent, отвечающая за работу контекстного меню;
- перегрузка функции mousePressEvent, отвечающая за реакцию на нажатие кнопки мыши;
- перегрузка функции paintEvent, отвечающая за перерисовку содержимого окна;

Также для решения задачи был разработан класс Figure, UML диаграмма которого приведена на рис. 2, содержащий закрытые поля contMenu типа указатель на QMenu, отвечающее за контекстное меню главного окна; delAction, editAction, movAction, rotAction типа указатель на QAction, отвечающие за кнопки в контестном меню; W, H, X, R1, R2, R3, Q1, Q2, Angle типа int, отвечающие за параметры фигуры; Square, Perimeter типа qreal, отвечающие за площадь и

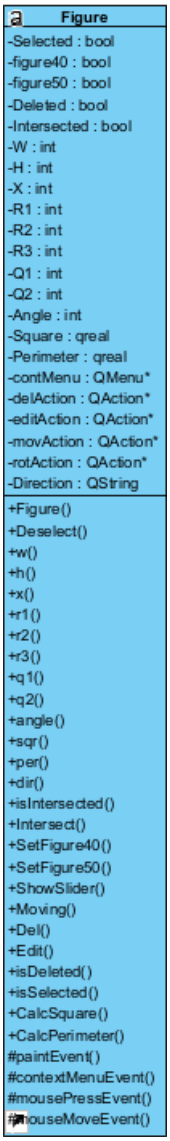

Рис. 2. UML-диаграмма класса Figure.

периметр фигуры соответсвенно; Direction типа QString, отвечающую за текущее направление поворота фигура; а также Selected, figure40, figure50, Deleted типа bool, являющиеся логическими флагами.
Также класс содержит:

- конструктор с параметром;
- функция Deselect, устанавливающая значение поля Selected на значение false:
- Функции, возвращающие параметры фигуры в качестве как rvalue, так и lvalue;
- Функции подсчета площади и периметра фигуры;
- перегрузка функции contextMenuEvent, отвечающая за работу контекстного меню;
- перегрузка функции mousePressEvent, отвечающая за реакцию на нажатие кнопки мыши;
- перегрузка функции paintEvent, отвечающая за перерисовку содержимого окна;
- перегрузка функции mouseMoveEvent, отвечающая за реакцию на перемещение мыши;

– сигнал signalBack, необходимый для связи с главным окном;

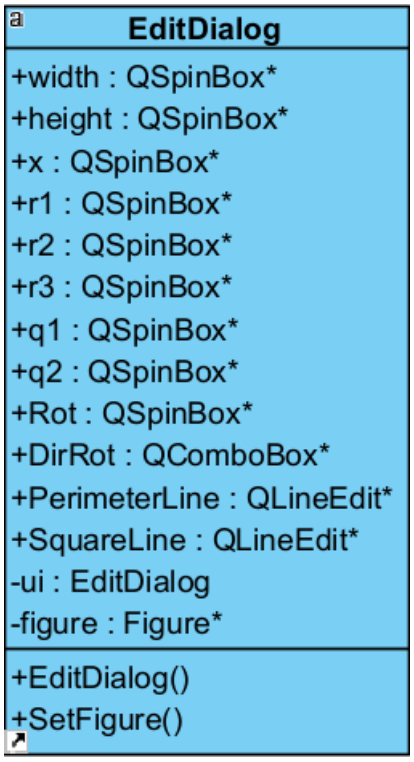

Рис. 3. UML-диаграмма класса EditDialog.

Также для решения задачи был разработан класс EditDialog, UML диаграмма которого приведена на рис. 3, содержащий закрытое поле figure типа указатель на Figure, отвечающее за изменяемую фигуру и открытые поля width, height, x, r1, r2, r3, q1, q2, Rot типа указатель на QSpinBox, отвечающие за изменение параметров фигуры в диалоговом окне; DirRot типа указатель на QComboBox, отвечающее за изменение направление поворота фигуры в диалоговом окне; PerimeterLine, SquareLine типа указатель на QLineEdit, отвечающие за вывод на экран периметра и площади фигуры.
Также класс содержит:

– конструктор с параметрами;

– функция SetFigure, устанавливающая исходные значения полей диалогового окна:

1.2 Задача на +1 балл

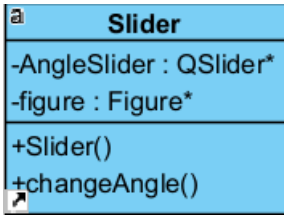

Рис. 4. UML-диаграмма класса Slider.

Для решения данной задачи дополнительно к уже имеющимся классам был разработан класс Slider, UML диаграмма которого приведена на рис. 4, содержащий закрытые поля figure типа

указатель на Figure, необходимый для связи с модифицируемой фигурой, а также AngleSldier типа указатель на QSlider.

Также класс содержит:

– конструктор с параметрами;

– функцию изменения угла фигуры;

1.3 Вторая задача на +1 балл

Для решения данной задачи в класс lab4 были добавлены функции FullClr, полностью очищающая главное окно, и InterClr, удаляющая пересекающиеся фигуры.

2 Выполнение задания

2.1 Задача на 6 баллов:

2.1.1 Функции класса lab4

Конструктор с параметром: поля, являющиеся указателями на QAction помещаются на необходимые меню и панель инструментов. ActionFigure 40 И ActionFigue50 помещаются в одну группу, чтобы не было возможности нажатия одновременно кнопки выбора обеих фигур, по умолчанию кнопка соответсвующая фигуре 40 зажата. Затем с помощью функции connect необходимые сигналы связываются с соответсвующими слотами.

Функция Add: создается новый виджет и добавляет его в вектор всех фигур.

Функция FullClr: сначала все виджеты с фигурами закрываются с помощью метода close, а затем вектор фигур полностью очищается и вызывается метод repaint для перерисовки содержимого главного окна.

Функция Intersection: данная функция является вспомогательной. С ее помощью определяется, пересекаются ли два прямоугольника.

Функция InterClr: данная функция отвечает за удаление пересекающихся фигур. сначала фигуры проходят проверку на то, какие из них пересекаются. Такие фигуры отмечаются и в дальнейшем происходит их закрытие соответсвующих виджетов и их удаление из вектора.

Функция Delete на панеле инструментов: сначала проверяется в цикле, какие фигуры выбраны, закрывает их и удаляет.

Перегрузка функции contextMenuEvent: запускает контекстное меню MainMenu при нажатии правой кнопкой мыши на главном окне.

Перегрузка функции mousePressEvent: при нажатии на кнопку мыши на главном окне со всех фигур снимается выделение.

Перегрузка функции paintEvent: сначала идет проверка количества выбранных фигур для того, чтобы поддерживать свойство того, что выбрана может быть только одна фигура. Затем если фигура из вектора фигур не удалена, то она выводится на галвное окно.

2.1.2 Функции класса Figure

Конструктор с параметром: поля, являющиеся указателями на QAction помещаются на необходимые меню и панель инструментов. Затем с помощью функции connect необходимые сигналы связываются с соответсвующими слотами. Далее задается размер виджета фигуры, случайным образом выбирается начальное местоположение, размеры и угол, высчитывается площадь и периметр.

Функция Moving: функция, отвечающая за перемещенике виджета из контекстного меню. Включается отслежввание движения мыши через setMouseTracking.

Функция Edit: сначала создается объект пользовательского класса EditDialog, затем ему передается текущая фигура, а если диалоговое окно завершилось с ключом Acccepted, то фигуре передаются новые параметры из диалогового окна.

Функция CalcSquare: суть подсчета площади заключается в уменьшении максимальной площади, т.е. площади прямоугольника на величину, равную сумме площадей кусков, способных дополнить фигуру до прямоугольника. Эти площади высчитываются в зависимости от формы фигуры и затем суммируется и вычитается из площади прямоугольника.

Функция CalcPerimeter: сначлаа считается длина каждого участка в отдельности, а для получения периметра эти значения суммируются.

Перегрузка функции paintEvent: сначала идет изменение размера на случай, если изменились значения фигуры перед перерисовкой. Затем если фигура выбрана, то цвет отрисовки меняется на синий. Затем координаты точки пересечения осей переносятся в центр виджета, чтобы фигура вращалась вокруг своего центра. Затем делаем поворот осей, учитывая текущее направление и затем начинаем поэлементную отрисовку в зависимости от типа фигуры.

Перегрузка функции contextMenuEvent: запускает контекстное меню contMenu при нажатии правой кнопкой мыши на главном окне.

Перегрузка функции mousePressEvent: при нажатии на кнопку мыши текущая фигура становится выбранной. Затем отключается отслеживание мыши через setMouseTracking, а также посылается сигнал SignalBack главному окну.

Перегрузка функции mouseMoveEvent: при зажатой левой кнопки мыши фигура перемещается в необходимую позицию,

2.1.3 Функции класса EditDialog

Конструктор с параметром: создаются указатели на QLabel под каждый параметр фигуры и помещаются на слой fl, являющийся указателем на QGridLayout, затем создаются необходимые панели ввода: 9 объектов QSpinBox, 2 QLineEdit и 1 QComboBox. Затем для всех полей ввода устанавливаются границы и необходимые свойства. Далее они помещаются на фрейм. Потом создается кнопка, означающая принятие введенных данных. В конце кнопка и слой добавляются на диалоговое окно.

Функция SetFigure: предназначена для заполнения полей ввода на диалоговом окне. Для этого сначала проверяется, не равен ли переданный указатель нулевому. Если данная проверка пройдена, то соответсвующие поля ввода заполняются полями фигуры

2.2 Задача на +1 балл

2.2.1 Дополнительные функции для класса lab4:

Функция FullClr: сначала все виджеты с фигурами закрываются с помощью метода close, а затем вектор фигур полностью очищается и вызывается метод repaint для перерисовки содержимого главного окна.

Функция Intersection: данная функция является вспомогательной. С ее помощью определяется, пересекаются ли два прямоугольника.

Функция InterClr: данная функция отвечает за удаление пересекающихся фигур. сначала фигуры проходят проверку на то, какие из них пересекаются. Такие фигуры отмечаются и в дальнейшем происходит их закрытие соответсвующих виджетов и их удаление из вектора.

2.3 Задача на +1 балл

2.3.1 Функции класса Slider

Конструктор с параметрами: полю figure присваивается значение переданного параметра. Выделяется память под поле AngleSlider, задается максимальное значение слайдера, минимальное, первоначальное положение и размер виджета. Для удобной отцентровки слайдер помещается на слой fl, являющийся указателем на QGridLayout. Затем происходит связывание с помощью функции connect сигнала, посылаемого слайдером при изменении значении со слотом changeAngle.

Функция changeAngle: фигуре передается новое значение угла, установленного на слайдере и обновляется.

3 Получение исполняемых модулей

Для получения исполняемого модуля lab4.exe была использована система сборки qmake.

Данный исполняемый модуль собирается из следующих файлов:

- lab4.h и lab4.cpp - файлы класса lab4;
- Figure.h и Figure.cpp - файлы класса Figure;
- EditDialog.h и EditDialog.cpp - файлы класса EditDialog;
- Slider.h и Slider.cpp - файлы класса Slider;
- main.cpp;

4 Тестирование

Тестирование производилось вручную. Проверялся общий функционал, а также кртические значения, например, выход фигуры за границы окна при перемещении, установка слишком большого значения в диалоговом окне при изменении фигуры, отсутсвие пересечения при очень сильном приближении двух фигур и другие.

Приложение А

А.1 Файл lab4.h

```
//VARIANT 40

#pragma once

#include <QtWidgets/QMainWindow>
#include "ui_lab4.h"
#include "Figure.h"

#include <QVector>
#include <QPushButton>
#include <QLabel>

class lab4 : public QMainWindow
{
        Q_OBJECT

public:
        lab4(QWidget* parent = Q_NULLPTR);
        void activateDel();

        QLabel* label;
        QVector<QPoint> points;

public slots:
        void Add();
        void Delete();
        void FullClr();
        void InterClr();

protected:
        void contextMenuEvent(QContextMenuEvent* event);
        void mousePressEvent(QMouseEvent* event);
        void paintEvent(QPaintEvent* event);

private:
        Ui::lab4Class ui;
        QMenu* MainMenu;
        QAction* ActionQuit, * ActionFigure40, * ActionFigure50, * ActionAdd,
                * ActionDelete, * clrAction, * ActionInterClr;
        //QToolBar* toolbar;
        QVector<Figure*> figures;
        //QVector<unsigned short> FigType;
};
```

А.2 Файл lab4.cpp

```
#include "lab4.h"
#include "Figure.h"
#include <QMenuBar>
#include <QCoreApplication>
#include <QContextMenuEvent>
#include <QToolBar>
#include <QActionGroup>
#include <QPainter>

lab4::lab4(QWidget* parent)
```

```
        : QMainWindow(parent)
{
        ui.setupUi(this);
        MainMenu = menuBar()->addMenu(tr("&File"));
        //setGeometry(100, 100, 1280, 720);
        setFixedSize(1280, 720);
        //ui.toolbar = new QToolBar(this);
        clrAction = MainMenu->addAction("&Clear");
        ActionInterClr = MainMenu->addAction("&Delete intersected");
        ActionAdd = ui.toolbar->addAction("&Add");
        ActionDelete = ui.toolbar->addAction("&Delete");
        ActionDelete->setEnabled(false);
        ui.toolbar->addSeparator();
        ActionFigure40 = ui.toolbar->addAction("&Figure 40");
        ActionFigure50 = ui.toolbar->addAction("&Figure 50");
        QActionGroup* group = new QActionGroup(this);
        group->addAction(ActionFigure40);
        group->addAction(ActionFigure50);
        ActionFigure40->setCheckable(true);
        ActionFigure40->setChecked(true);
        ActionFigure50->setCheckable(true);
        //ActionAdd->setEnabled(false);

        label = new QLabel(this);

        setMouseTracking(true);

        connect(ActionFigure40, SIGNAL(triggered()), this,
                SLOT(Figure::SetFigure40()));
        connect(ActionFigure50, SIGNAL(trigerred()), this,
                SLOT(Figure::SetFigure50()));
        connect(ActionAdd, SIGNAL(triggered()), this, SLOT(Add()));
        connect(clrAction, &QAction::triggered, this, &lab4::FullClr);
        connect(ActionInterClr, &QAction::triggered, this, &lab4::InterClr);
        connect(ActionDelete, SIGNAL(triggered()), this, SLOT(Delete()));
}

void lab4::Add()
{
        Figure* f = new Figure(this);
        if (ActionFigure40->isChecked())
        {
                f->SetFigure40(true);
                f->SetFigure50(false);
        }
        else if (ActionFigure50->isChecked())
        {
                f->SetFigure40(false);
                f->SetFigure50(true);
        }
        figures.push_back(f);
        connect(figures[figures.size() - 1], &Figure::signalBack, this,
                &lab4::activateDel);
        repaint();
}

void lab4::activateDel()
{
        ActionDelete->setEnabled(true);
}
```

```
71
72  void lab4::FullClr()
73  {
74          for (auto elem : figures)
75                  elem->close();
76
77          figures.clear();
78          repaint();
79  }
80
81  bool Intersection(Figure* a, Figure* b) {
82          QVector<QPoint> vec1, vec2;
83
84          QPixmap tmp1 = a->grab(QRect(1, 1, a->width(), a->height()));
85          QImage image1 = tmp1.toImage();
86
87          //obj->label->setPixmap(QPixmap::fromImage(image1));
88          //obj->label->setGeometry(100, 100, 500, 500);
89
90          /*int X1 = a->x(),
91          X2 = b->x(),
92          Y1 = a->y(),
93          Y2 = b->y(),
94          X1glob = a->geometry().x();*/
95
96          for (std::size_t x{ 1 }; x < a->width(); ++x)
97          {
98                  for (std::size_t y{ 1 }; y < a->height(); ++y)
99                  {
100                         QPoint point(x, y);
101                         QPoint globPos(x + a->geometry().x(), y + a->geometry().y());
102                         QColor col = image1.pixelColor(point);
103
104                         if (col == QColor(Qt::black))
105                                 vec1.push_back(globPos);
106                 }
107         }
108
109         QPixmap tmp2 = b->grab(QRect(1, 1, b->width(), b->height()));
110         QImage image2 = tmp2.toImage();
111
112         for (std::size_t x{ 1 }; x < b->width(); ++x)
113         {
114                 for (std::size_t y{ 1 }; y < b->height(); ++y)
115                 {
116                         QPoint point(x, y);
117                         QPoint globPos(x + b->geometry().x(), y + b->geometry().y());
118                         QColor col = image2.pixelColor(point);
119
120                         if (col == QColor(Qt::black))
121                                 vec2.push_back(globPos);
122                 }
123         }
124
125         QVector<QPoint> inter;
126
127         for (auto elem1 : vec1)
128                 if (std::find(vec2.begin(), vec2.end(), elem1) != vec2.end())
129                         return true;
130
```

```
131         /*for (auto elem1 : vec1)
132         {
133                 for (auto elem2 : vec2)
134                 {
135                         if (elem1 == elem2)
136                                 inter.push_back(elem1);
137                 }
138         }*/
139
140         //obj->points = inter;
141
142         return false;
143 }
144
145 void lab4::InterClr()
146 {
147         if (figures.size() > 1)
148         {
149                 for (std::size_t i{ 0 }; i < figures.size() - 1; ++i)
150                         for (std::size_t j{ i + 1 }; j < figures.size(); ++j)
151                         {
152                                 if (!figures[i]->isDeleted()
153                                         && !figures[j]->isDeleted())
154                                 {
155                                         Figure* r1 = figures[i];
156                                         Figure* r2 = figures[j];
157                                         if (Intersection(r1, r2))
158                                         {
159                                                 figures[i]->Intersect() = true;
160                                                 figures[j]->Intersect() = true;
161                                         }
162                                 }
163                         }
164
165                 std::size_t k = 0;
166                 while (k < figures.size())
167                         if (figures[k]->isIntersected())
168                         {
169                                 figures[k]->close();
170                                 figures.erase(&figures[k]);
171                                 k = 0;
172                         }
173                         else ++k;
174
175                 repaint();
176         }
177 }
178
179 void lab4::Delete()
180 {
181         for (std::size_t i{ 0 }; i < figures.size(); ++i)
182         {
183                 if (figures[i]->isSelected())
184                 {
185                         figures[i]->close();
186                         figures.erase(&figures[i]);
187                 }
188         }
189         ActionDelete->setEnabled(false);
190 }
```

```
191
192 void lab4::contextMenuEvent(QContextMenuEvent* event)
193 {
194         MainMenu->exec(event->globalPos());
195 }
196
197 void lab4::mousePressEvent(QMouseEvent* event)
198 {
199         for (auto elem : figures)
200                 elem->Deselect();
201         setMouseTracking(false);
202         ActionDelete->setEnabled(false);
203         repaint();
204 }
205
206 void lab4::paintEvent(QPaintEvent* event)
207 {
208         QPainter p(this);
209         p.setPen(Qt::red);
210         p.drawPoints(points);
211         std::size_t cnt{ 0 };
212         for (auto elem : figures)
213         {
214                 if (elem->isSelected())
215                         ++cnt;
216         }
217
218         for (auto elem : figures)
219         {
220                 if (cnt >= 2)
221                         elem->Deselect();
222
223                 if (!elem->isDeleted())
224                 {
225                         elem->update();
226                         elem->show();
227                 }
228         }
229 }
```

Приложение Б

Б.1 Файл Figure.h

```
#pragma once
#include <QWidget>
#include <QMenu>
#include <QVariantAnimation>

class Figure : public QWidget
{
        Q_OBJECT;

public:
        explicit Figure(QWidget* parent = nullptr);

        void Deselect();

        int  w() const;
        int& w();

        int  h() const;
        int& h();

        int  x() const;
        int& x();

        int  r1() const;
        int& r1();

        int  r2() const;
        int& r2();

        int  r3() const;
        int& r3();

        int  q1() const;
        int& q1();

        int  q2() const;
        int& q2();

        int  angle() const;
        int& angle();

        qreal sqr() const;
        qreal per() const;

        QString dir() const;
        QString& dir();

        bool isIntersected() const;
        bool& Intersect();

        void SetFigure40(bool b);
        void SetFigure50(bool b);

        void ShowSlider();
        void Moving();
        void Del();
```

```
57          void Edit();
58          bool isDeleted();
59          bool isSelected();
60
61          void CalcSquare();
62          void CalcPerimeter();
63
64  protected:
65          void paintEvent(QPaintEvent* event);
66          void contextMenuEvent(QContextMenuEvent* event);
67          void mousePressEvent(QMouseEvent* event);
68          void mouseMoveEvent(QMouseEvent* event);
69
70  signals:
71          void signalBack();
72
73  private:
74          bool Selected, figure40 = true, figure50 = false, Deleted = false,
75                  Intersected = false;
76
77          int W, H, X, R1, R2, R3, Q1, Q2, Angle;
78          qreal Square, Perimeter;
79          QMenu* contMenu;
80          QAction* delAction, * editAction, * movAction, * rotAction;
81          QString Direction = "Counterclockwise";
82  };
```

Б.2 Файл Figure.cpp

```
1   #include "Figure.h"
2   #include "Slider.h"
3   #include "EditDialog.h"
4   #include <QPainter>
5   #include <QMouseEvent>
6   #include <QContextMenuEvent>
7   #include <random>
8   #include <QMenu>
9   #include <cmath>
10  #include <numeric>
11
12  Figure::Figure(QWidget* parent)
13          :QWidget(parent), Selected(false)
14  {
15          std::random_device rd;
16          std::mt19937 mersenne(rd());
17
18          /*W = 200;        H = 150; X = 20;
19          Q1 = 40; Q2 = 30;
20          R1 = 20; R2 = 30; R3 = 40;*/
21
22          Angle = mersenne() % 360;
23
24          W = mersenne() % 200 + 75;
25          H = (mersenne() % W + 100) % W + 30;
26          X = (mersenne() % H / 3 + 50) % H / 3 + 1;
27          Q1 = (mersenne() % W / 4 + 100) % W / 4 + 1;
28          Q2 = (mersenne() % W / 4 + 100) % W / 4 + 1;
29          R1 = (mersenne() % H / 3 + 50) % H / 3 + 1;
30          R2 = (mersenne() % H / 3 + 50) % H / 3 + 1;
31          R3 = (mersenne() % H / 3 + 50) % H / 3 + 1;
32
```

```
33          contMenu = new QMenu(this);
34          delAction = new QAction(tr("&Delete"));
35          editAction = new QAction(tr("&Edit"));
36          movAction = new QAction(tr("&Move"));
37          rotAction = new QAction(tr("&Rotate"));
38          contMenu->addAction(editAction);
39          contMenu->addAction(movAction);
40          contMenu->addAction(rotAction);
41          contMenu->addAction(delAction);
42
43          connect(delAction, &QAction::triggered, this, &Figure::Del);
44          connect(editAction, &QAction::triggered, this, &Figure::Edit);
45          connect(rotAction, &QAction::triggered, this, &Figure::ShowSlider);
46          connect(movAction, &QAction::triggered, this, &Figure::Moving);
47
48          resize(sqrt(W * W + H * H), sqrt(W * W + H * H));
49
50          std::size_t curx = mersenne() % (1280 - width());
51          std::size_t cury = mersenne() % (720 - height());
52
53          cury = (cury < 50) ? 50 : cury;
54
55          move(curx, cury);
56
57          //QPalette Pal(palette());
58          //Pal.setColor(QPalette::Background, Qt::white);
59          //setAutoFillBackground(true);
60          //setPalette(Pal);
61
62          CalcSquare();
63          CalcPerimeter();
64
65          repaint();
66  }
67
68  void Figure::Deselect()
69  {
70          Selected = false;
71  }
72
73  int Figure::h() const
74  {
75          return H;
76  }
77
78  int& Figure::h()
79  {
80          return H;
81  }
82
83  int Figure::x() const
84  {
85          return X;
86  }
87
88  int Figure::w() const
89  {
90          return W;
91  }
92
```

```
93  int& Figure::w()
94  {
95          return W;
96  }
97
98  void Figure::SetFigure40(bool b)
99  {
100         figure40 = b;
101 }
102
103 void Figure::SetFigure50(bool b)
104 {
105         figure50 = b;
106 }
107
108 void Figure::ShowSlider()
109 {
110         Slider* obj = new Slider(this, this);
111         obj->show();
112 }
113
114 void Figure::Moving()
115 {
116         setMouseTracking(true);
117 }
118
119 void Figure::Del()
120 {
121         Deleted = true;
122         close();
123         repaint();
124 }
125
126 void Figure::Edit()
127 {
128         EditDialog edit(this, this);
129         edit.SetFigure(this);
130         if (edit.exec() == edit.Accepted)
131         {
132                 w() = edit.width->value();
133                 h() = edit.height->value();
134                 x() = edit.x->value();
135                 r1() = edit.r1->value();
136                 r2() = edit.r2->value();
137                 r3() = edit.r3->value();
138                 q1() = edit.q1->value();
139                 q2() = edit.q2->value();
140                 angle() = edit.Rot->value();
141                 dir() = edit.DirRot->currentText();
142                 CalcSquare();
143                 CalcPerimeter();
144                 qreal newSide = std::sqrt(std::pow(static_cast<qreal>(W), 2)
145                         + std::pow(static_cast<qreal>(H), 2));
146                 QSize newSize(newSide, newSide);
147                 resize(newSize);
148         }
149
150         update();
151 }
152
```

```
153 bool Figure::isDeleted()
154 {
155         return Deleted;
156 }
157
158 bool Figure::isSelected()
159 {
160         return Selected;
161 }
162
163 void Figure::CalcSquare()
164 {
165         qreal SqRect = W * H;
166
167         if (figure40)
168         {
169                 qreal SqA = 0.5 * R1 * R1;
170                 qreal SqB = 0.5 * R2 * R2;
171                 qreal SqC = std::_Pi * R3 * R3 / 4;
172                 qreal SqD = 0.5 * X * X;
173                 qreal SqE = std::_Pi * Q1 * Q1 / 8;
174                 qreal SqF = std::_Pi * Q2 * Q2 / 8;
175                 Square = SqRect - (SqA + SqB + SqC + SqD + SqE);
176         }
177         else if (figure50)
178         {
179                 qreal SqA = std::_Pi * R2 * R2;
180                 qreal SqB = std::_Pi * R3 * R3;
181                 qreal SqC = 0.5 * X * X;
182                 qreal SqD = std::_Pi * R1 * R1;
183                 qreal SqE = std::_Pi * Q1 * Q1 / 8;
184                 qreal SqF = std::_Pi * Q2 * Q2 / 8;
185                 Square = SqRect - (SqA + SqB + SqC + SqD + SqE + SqF);
186         }
187 }
188
189 void Figure::CalcPerimeter()
190 {
191         if (figure40)
192         {
193                 QVector<qreal> parts(9);
194                 parts[0] = 2 * std::_Pi * R3 / 4;
195                 parts[1] = W - R3 - R2;
196                 parts[2] = 2 * std::_Pi * R2 / 4;
197                 parts[3] = H - R1 - R2;
198                 parts[4] = 2 * std::_Pi * R1 / 4;
199                 parts[5] = W - R1 - Q1 - X;
200                 parts[6] = std::_Pi * Q1;
201                 parts[7] = sqrt(2) * X;
202                 parts[8] = H - R3 - X;
203
204                 Perimeter = std::accumulate(parts.begin(), parts.end(), 0);
205         }
206         else if (figure50)
207         {
208                 QVector<qreal> parts(10);
209                 parts[0] = 2 * std::_Pi * R1 / 4;
210                 parts[1] = H - R1 - X;
211                 parts[2] = sqrt(2) * X;
212                 parts[3] = W - X - Q2 - R3;
```

```
213                 parts[4] = std::_Pi * Q2;
214                 parts[5] = 2 * std::_Pi * R3 / 4;
215                 parts[6] = H - R2 - R3;
216                 parts[7] = 2 * std::_Pi * R2 / 4;
217                 parts[8] = W - Q1 - R2 - R1;
218                 parts[9] = std::_Pi * Q1;
219 
220                 Perimeter = std::accumulate(parts.begin(), parts.end(), 0);
221         }
222 }
223 
224 void Figure::paintEvent(QPaintEvent* event)
225 {
226         QPainter p(this);
227 
228         //p.setPen(Qt::red);
229         //p.drawPoint(a);
230         //p.setPen(Qt::green);
231         //p.drawPoint(b);
232 
233         resize(sqrt(W * W + H * H), sqrt(W * W + H * H));
234 
235         QPen pen;
236         pen.setColor(Qt::black);
237         pen.setWidth(3);
238 
239         if (Selected)
240         {
241                 pen.setColor(Qt::darkBlue);
242         }
243 
244         p.setPen(pen);
245 
246         p.translate(QPoint(sqrt(W * W + H * H) / 2, sqrt(W * W + H * H) / 2));
247         if (Direction == "Clockwise")
248                 p.rotate(Angle);
249         else if (Direction == "Counterclockwise")
250                 p.rotate(-Angle);
251         //       40
252         if (figure40)
253         {
254                 qreal num = W / 2;
255                 qreal num2 = H / 2;
256                 p.drawLine(X - num, -num2, (W - Q1) / 2 - num, -num2);
257                 p.drawArc((W - Q1) / 2 - num, -Q1 / 2 - num2,
258                         Q1, Q1, 180 * 16, 180 * 16);
259                 p.drawLine((W + Q1) / 2 - num, -num2, W - R1 - num, -num2);
260                 p.drawArc(W - 2 * R1 - num, -num2, 2 * R1, 2 * R1, 0, 90 * 16);
261                 p.drawLine(W - num, R1 - num2, W - num, H - R2 - num2);
262                 p.drawArc(W - 2 * R2 - num, H - 2 * R2 - num2,
263                         2 * R2, 2 * R2, 270 * 16, 90 * 16);
264                 p.drawLine(W - R2 - num, H - num2, R3 - num, H - num2);
265                 p.drawArc(-R3 - num, H - R3 - num2, 2 * R3, 2 * R3, 0, 90 * 16);
266                 p.drawLine(-num, H - R3 - num2, -num, X - num2);
267                 p.drawLine(-num, X - num2, X - num, -num2);
268         }
269         else if (figure50)
270         {
271                 qreal num = W / 2;
272                 qreal num2 = H / 2;
```

```
273                 p.drawLine(R1 - num, -num2, (W - Q1) / 2 - num, -num2);
274                 p.drawArc((W - Q1) / 2 - num, -Q1 / 2 - num2,
275                         Q1, Q1, 180 * 16, 180 * 16);
276                 p.drawLine((W + Q1) / 2 - num, -num2, W - R2 - num, -num2);
277                 p.drawArc(W - R2 - num, -R2 - num2,
278                         2 * R2, 2 * R2, 180 * 16, 90 * 16);
279                 p.drawLine(W - num, R2 - num2, W - num, H - R3 - num2);
280                 p.drawArc(W - R3 - num, H - R3 - num2,
281                         2 * R3, 2 * R3, 90 * 16, 90 * 16);
282                 p.drawLine(W - R3 - num, H - num2, (W + Q2) / 2 - num, H - num2);
283                 p.drawArc((W - Q2) / 2 - num, H - Q2 / 2 - num2,
284                         Q2, Q2, 0, 180 * 16);
285                 p.drawLine((W - Q2) / 2 - num, H - num2, X - num, H - num2);
286                 p.drawLine(X - num, H - num2, -num, H - X - num2);
287                 p.drawLine(-num, H - X - num2, -num, R1 - num2);
288                 p.drawArc(-R1 - num, -R1 - num2, 2 * R1, 2 * R1, 270 * 16, 90 * 16);
289         }
290 }
291
292 void Figure::contextMenuEvent(QContextMenuEvent* e)
293 {
294         QTransform matrix;
295
296         int ynew, xnew, ynew1, xnew1, ynew2, xnew2, x1, y1;
297         qreal ugol = Angle;
298
299         if (Direction == "Clockwise")
300                 ugol = -Angle;
301
302         matrix = matrix.translate(width() / 2, height() / 2).rotate(-ugol);
303
304         ugol *= std::_Pi / 180;
305
306         x1 = e->x();
307         y1 = e->y();
308         QPoint newpoint = QPoint(x1, y1) * matrix.inverted();
309
310         if (figure40)
311         {
312                 ynew1 = ((-H + Q1) / 2);
313                 ynew2 = (H / 2 - R3);
314                 xnew1 = (-W / 2 + std::max(X, R3));
315                 xnew2 = W / 2;
316         }
317         else if (figure50)
318         {
319                 ynew1 = -H / 2 + qMax(qMax(R1, R2), Q1 / 2);
320                 ynew2 = H / 2 - qMax(qMax(X, Q2 / 2), R3);
321                 xnew1 = -W / 2 + qMax(X, R1);
322                 xnew2 = W / 2 - qMax(R2, R3);
323         }
324
325         if (newpoint.y() > ynew1 && newpoint.y() < ynew2 &&
326                 newpoint.x() > xnew1 && newpoint.x() < xnew2)
327                 contMenu->exec(e->globalPos());
328 }
329
330 void Figure::mousePressEvent(QMouseEvent* e)
331 {
332         QTransform matrix;
```

```
333
334         int ynew, xnew, ynew1, xnew1, ynew2, xnew2, x1, y1;
335         qreal ugol = Angle;
336
337         if (Direction == "Clockwise")
338                 ugol = -Angle;
339
340         matrix = matrix.translate(width() / 2, height() / 2).rotate(-ugol);
341
342         ugol *= std::_Pi / 180;
343
344         x1 = e->localPos().x();
345         y1 = e->localPos().y();
346         QPoint newpoint = QPoint(x1, y1) * matrix.inverted();
347
348         if (figure40)
349         {
350                 ynew1 = ((-H + Q1) / 2);
351                 ynew2 = (H / 2 - R3);
352                 xnew1 = (-W / 2 + std::max(X, R3));
353                 xnew2 = W / 2;
354         }
355         else if (figure50)
356         {
357                 ynew1 = -H / 2 + qMax(qMax(R1, R2), Q1 / 2);
358                 ynew2 = H / 2 - qMax(qMax(X, Q2 / 2), R3);
359                 xnew1 = -W / 2 + qMax(X, R1);
360                 xnew2 = W / 2 - qMax(R2, R3);
361         }
362
363         if (newpoint.y() > ynew1 && newpoint.y() < ynew2 &&
364                 newpoint.x() > xnew1 && newpoint.x() < xnew2)
365         {
366                 Selected = true;
367                 setMouseTracking(false);
368
369                 signalBack();
370         }
371         else Selected = false;
372
373         repaint();
374 }
375
376 void Figure::mouseMoveEvent(QMouseEvent* e)
377 {
378         QTransform matrix;
379         //a = e->localPos();
380         int ynew, xnew, ynew1, xnew1, ynew2, xnew2, x1, y1;
381         qreal ugol = Angle;
382
383         if (Direction == "Clockwise")
384                 ugol = -Angle;
385
386         matrix = matrix.translate(width() / 2, height() / 2).rotate(-ugol);
387
388         ugol *= std::_Pi / 180;
389
390         x1 = e->localPos().x();
391         y1 = e->localPos().y();
392         QPoint newpoint = QPoint(x1, y1) * matrix.inverted();
```

```
393
394         if (figure40)
395         {
396                 ynew1 = ((-H + Q1) / 2);
397                 ynew2 = (H / 2 - R3);
398                 xnew1 = (-W / 2 + std::max(X, R3));
399                 xnew2 = W / 2;
400         }
401         else if (figure50)
402         {
403                 ynew1 = -H / 2 + qMax(qMax(R1, R2), Q1 / 2);
404                 ynew2 = H / 2 - qMax(qMax(X, Q2 / 2), R3);
405                 xnew1 = -W / 2 + qMax(X, R1);
406                 xnew2 = W / 2 - qMax(R2, R3);
407         }
408
409         if (newpoint.y() > ynew1 && newpoint.y() < ynew2 &&
410                 newpoint.x() > xnew1 && newpoint.x() < xnew2)
411         {
412                 move(e->windowPos().x() - width()  / 2,
413                          e->windowPos().y() - height() / 2);
414
415                 if (e->windowPos().x() - width() / 2 > 1280 - geometry().width())
416                         move(1280 - geometry().width(),
417                                 e->windowPos().y() - height() / 2);
418                 if (e->windowPos().y() - height() / 2 < 50)
419                         move(e->windowPos().x() - width() / 2, 50);
420                 if (e->windowPos().x() - width() / 2 < 0)
421                         move(0, e->windowPos().y() - height() / 2);
422                 if (e->windowPos().y() - width() / 2 > 720 - geometry().height())
423                         move(e->windowPos().x() - width() / 2,
424                                 720 - geometry().height());
425
426                 if ((e->windowPos().x() - width() / 2 >= 1280 - geometry().width())
427                         && (e->windowPos().y() - height() / 2 >=
428                                 720 - geometry().height()))
429                         move(1280 - geometry().width(), 720 - geometry().height());
430
431                 if ((e->windowPos().x() - width() / 2 >= 1280 - geometry().width())
432                         && (e->windowPos().y() - height() / 2 <= 50))
433                         move(1280 - geometry().width(), 50);
434
435                 if ((e->windowPos().x() - width() / 2 <= 0)
436                         && (e->windowPos().y() - height() / 2 <= 50))
437                         move(0, 50);
438
439                 if ((e->windowPos().x() - width() / 2 <= 0)
440                         && (e->windowPos().y() - height() / 2 >=
441                                 720 - geometry().height()))
442                         move(0, 720 - geometry().height());
443         }
444         update();
445 }
446
447 int& Figure::x()
448 {
449         return X;
450 }
451
452 int Figure::r1() const
```

```
453 {
454         return R1;
455 }
456
457 int& Figure::r1()
458 {
459         return R1;
460 }
461
462 int Figure::r2() const
463 {
464         return R2;
465 }
466
467 int& Figure::r2()
468 {
469         return R2;
470 }
471
472 int Figure::r3() const
473 {
474         return R3;
475 }
476
477 int& Figure::r3()
478 {
479         return R3;
480 }
481
482 int Figure::q1() const
483 {
484         return Q1;
485 }
486
487 int& Figure::q1()
488 {
489         return Q1;
490 }
491
492 int Figure::q2() const
493 {
494         return Q2;
495 }
496
497 int& Figure::q2()
498 {
499         return Q2;
500 }
501
502 int Figure::angle() const
503 {
504         return Angle;
505 }
506
507 int& Figure::angle()
508 {
509         return Angle;
510 }
511
512 qreal Figure::sqr() const
```

```
513 {
514         return Square;
515 }
516
517 qreal Figure::per() const
518 {
519         return Perimeter;
520 }
521
522 QString Figure::dir() const
523 {
524         return Direction;
525 }
526
527 QString& Figure::dir()
528 {
529         return Direction;
530 }
531
532 bool Figure::isIntersected() const
533 {
534         return Intersected;
535 }
536
537 bool& Figure::Intersect()
538 {
539         return Intersected;
540 }
```

Приложение В

В.1 Файл EditDialog.h

```
#pragma once

#include "ui_EditDialog.h"
#include "Figure.h"

#include <QDialog>
#include <QSpinBox>
#include <QPushButton>
#include <QComboBox>

class EditDialog : public QDialog
{
        Q_OBJECT

public:
        EditDialog(Figure* f, QWidget* parent = Q_NULLPTR);

        void SetFigure(Figure* f);

        QSpinBox* width, * height, * x, * r1, * r2, * r3, * q1, * q2, * Rot;
        QComboBox* DirRot;
        QLineEdit* PerimeterLine, * SquareLine;

private:
        Ui::EditDialog ui;
        Figure* figure;
};
```

В.2 Файл EditDialog.cpp

```
#include "EditDialog.h"
#include <QLabel>
#include <QGridLayout>
#include <QFrame>
#include <QComboBox>
#include <QLineEdit>

EditDialog::EditDialog(Figure* f, QWidget* parent)
        : QDialog(parent), figure(f)
{
        ui.setupUi(this);

        QFrame* frame = new QFrame(this);
        frame->setFrameStyle(QFrame::Box | QFrame::Sunken);

        QLabel* lW = new QLabel(tr("Width"), frame);
        QLabel* lH = new QLabel(tr("Height"), frame);
        QLabel* lX = new QLabel(tr("X"), frame);
        QLabel* lR1 = new QLabel(tr("R1"), frame);
        QLabel* lR2 = new QLabel(tr("R2"), frame);
        QLabel* lR3 = new QLabel(tr("R3"), frame);
        QLabel* lQ1 = new QLabel(tr("Q1"), frame);
        QLabel* lQ2 = new QLabel(tr("Q2"), frame);
        QLabel* lDirRot = new QLabel(tr("Direction of rotation"), frame);
        QLabel* lRot = new QLabel(tr("Rotation"), frame);
        QLabel* lPer = new QLabel(tr("Perimeter"), frame);
```

```
27          QLabel* lS = new QLabel(tr("Square"), frame);
28
29          DirRot = new QComboBox(frame);
30          DirRot->addItem(tr("Counterclockwise"));
31          DirRot->addItem(tr("Clockwise"));
32
33          width = new QSpinBox(frame);
34          height = new QSpinBox(frame);
35          x = new QSpinBox(frame);
36          r1 = new QSpinBox(frame);
37          r2 = new QSpinBox(frame);
38          r3 = new QSpinBox(frame);
39          q1 = new QSpinBox(frame);
40          q2 = new QSpinBox(frame);
41          Rot = new QSpinBox(frame);
42
43          PerimeterLine = new QLineEdit(frame);
44          SquareLine = new QLineEdit(frame);
45          PerimeterLine->setReadOnly(true);
46          SquareLine->setReadOnly(true);
47
48          width->setRange(60, 300);
49          height->setRange(24, width->maximum() - 1);
50          x->setRange(1, height->maximum() / 3 - 1);
51          r1->setRange(1, height->maximum() / 3 - 1);
52          r2->setRange(1, height->maximum() / 3 - 1);
53          r3->setRange(1, height->maximum() / 3 - 1);
54          q1->setRange(1, width->maximum() / 4 - 1);
55          q2->setRange(1, width->maximum() / 4 - 1);
56          Rot->setRange(0, 359);
57
58          width->setValue(f->w());
59          height->setValue(f->h());
60          x->setValue(f->x());
61          r1->setValue(f->r1());
62          r2->setValue(f->r2());
63          r3->setValue(f->r3());
64          q1->setValue(f->q1());
65          q2->setValue(f->q2());
66          Rot->setValue(f->angle());
67          SquareLine->setText(QString::number(f->sqr()));
68          PerimeterLine->setText(QString::number(f->per()));
69
70          QGridLayout* fl = new QGridLayout(frame);
71          fl->addWidget(lW, 0, 0);
72          fl->addWidget(lH, 1, 0);
73          fl->addWidget(lX, 2, 0);
74          fl->addWidget(lR1, 3, 0);
75          fl->addWidget(lR2, 4, 0);
76          fl->addWidget(lR3, 5, 0);
77          fl->addWidget(lQ1, 6, 0);
78          fl->addWidget(lQ2, 7, 0);
79          fl->addWidget(lDirRot, 8, 0);
80          fl->addWidget(lRot, 9, 0);
81          fl->addWidget(lPer, 10, 0);
82          fl->addWidget(lS, 11, 0);
83          fl->addWidget(width, 0, 1);
84          fl->addWidget(height, 1, 1);
85          fl->addWidget(x, 2, 1);
86          fl->addWidget(r1, 3, 1);
```

```
87          fl->addWidget(r2, 4, 1);
88          fl->addWidget(r3, 5, 1);
89          fl->addWidget(q1, 6, 1);
90          fl->addWidget(q2, 7, 1);
91          fl->addWidget(DirRot, 8, 1);
92          fl->addWidget(Rot, 9, 1);
93          fl->addWidget(PerimeterLine, 10, 1);
94          fl->addWidget(SquareLine, 11, 1);
95          frame->setLayout(fl);
96
97          QPushButton* accbutt = new QPushButton(tr("Accept"), this);
98          connect(accbutt, SIGNAL(clicked()), this, SLOT(accept()));
99
100         QGridLayout* l = new QGridLayout(this);
101         l->addWidget(frame, 0, 0);
102         l->addWidget(accbutt, 1, 0);
103         setLayout(l);
104
105         setModal(true);
106 }
107
108 void EditDialog::SetFigure(Figure* f)
109 {
110         figure = f;
111         if (figure != nullptr)
112         {
113                 width->setValue(f->w());
114                 height->setValue(f->h());
115                 x->setValue(f->x());
116                 r1->setValue(f->r1());
117                 r2->setValue(f->r2());
118                 r3->setValue(f->r3());
119                 q1->setValue(f->q1());
120                 q2->setValue(f->q2());
121                 Rot->setValue(f->angle());
122                 if (DirRot->currentText() != f->dir())
123                         DirRot->setCurrentIndex((DirRot->currentIndex() + 1) % 2);
124                 SquareLine->setText(QString::number(f->sqr()));
125                 PerimeterLine->setText(QString::number(f->per()));
126
127                 height->setRange(24, width->value() - 1);
128                 x->setRange(1, height->value() / 3 - 1);
129                 r1->setRange(1, height->value() / 3 - 1);
130                 r2->setRange(1, height->value() / 3 - 1);
131                 r3->setRange(1, height->value() / 3 - 1);
132                 q1->setRange(1, width->value() / 4 - 1);
133                 q2->setRange(1, width->value() / 4 - 1);
134         }
135 }
```

Приложение Г

Г.1 Файл Slider.h

```
#pragma once

#include <QDialog>
#include <QSlider>
#include "Figure.h"

class Slider : public QDialog
{
        Q_OBJECT

public:
        Slider(Figure* f, QWidget* parent);
        void changeAngle();

private:
        QSlider* AngleSlider;
        Figure* figure;
};
```

Г.2 Файл Slider.cpp

```
#include "Slider.h"
#include "Figure.h"
#include <QSlider>
#include <QGridLayout>
#include <QDialog>

Slider::Slider(Figure* f, QWidget* parent)
        : QDialog(parent), figure(f)
{
        AngleSlider = new QSlider(Qt::Horizontal, this);
        AngleSlider->show();
        AngleSlider->setMaximum(180);
        AngleSlider->setMinimum(-180);
        AngleSlider->setValue(0);
        setFixedSize(100, 100);

        QGridLayout* fl = new QGridLayout(this);
        fl->addWidget(AngleSlider, 0, 0);
        setLayout(fl);
        connect(AngleSlider, &QSlider::valueChanged, this, &Slider::changeAngle);
}

void Slider::changeAngle()
{
        figure->angle() = AngleSlider->value();
        figure->update();
}
```

Приложение Д

Д.1 Файл main.cpp

```
#include "lab4.h"
#include <QtWidgets/QApplication>

int main(int argc, char* argv[])
{
        QApplication a(argc, argv);
        lab4 w;
        w.show();
        return a.exec();
}
```