Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра «Компьютерная безопасность»

ОТЧЕТ

К ЛАБОРАТОРНОЙ РАБОТЕ №11

По дисциплине

«Языки программирования»

Работу выполнил

Студент группы СКБ 193 _____ **С.О.Ташкинов**

подпись, дата

Работу проверил _____ **С.А. Булгаков**

Москва 2020

Постановка задачи

Разработать графическое приложение с использованием библиотеки Qt. Приложение состоитиз основного окна (наследовать QMainWindow), с панелью инструментов на которойрасположены:

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

2)Кнопка добавления фигуры. Все фигуры поворачиваются относительно центраописывающего их прямоугольника (по умолчению против часовой стрелки).Параметры фигуры выбираются произвольно в допустимом диапазоне. Если ни однафигура не выбрана, кнопка добавления не активна.

3)Кнопка удаления выделенной фигуры. Если фигура не выделена, кнопка не активна.

Основная часть окна предназначена для размещения фигур (запрещается использоватьQGraphicsScene). Фон основной части окна – белый, цвет отрисовки фигур – черный.При нажатии на фигуре левой кнопкой мыши – фигура выделяется (отрисовываетсясиним цветом). При нажатии на фигуре правой кнопкой мыши – фигура выделяется иоткрывается модальное диалоговое окно позволяющее изменить параметры фигуры, уголповорота, 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 неучитывается.

1 Общий алгоритм решения поставленной задачи

1.1 Классы

Для решения поставленной задачи были разработаны пользовательские классы Dialog1 и Dialog2, наследуемые от класса QDialog, используемые для вывода модального окна и считывания пользовательского ввода с клавиатуры; MainWindow, наследуемый от QMainWindow, используемый для вывода на экран основного окна. Figure1 и Figure2, наследуемые от класса QWidget, в методах которых происходит отрисовка фигур и считывание пользовательского ввода с мыши.

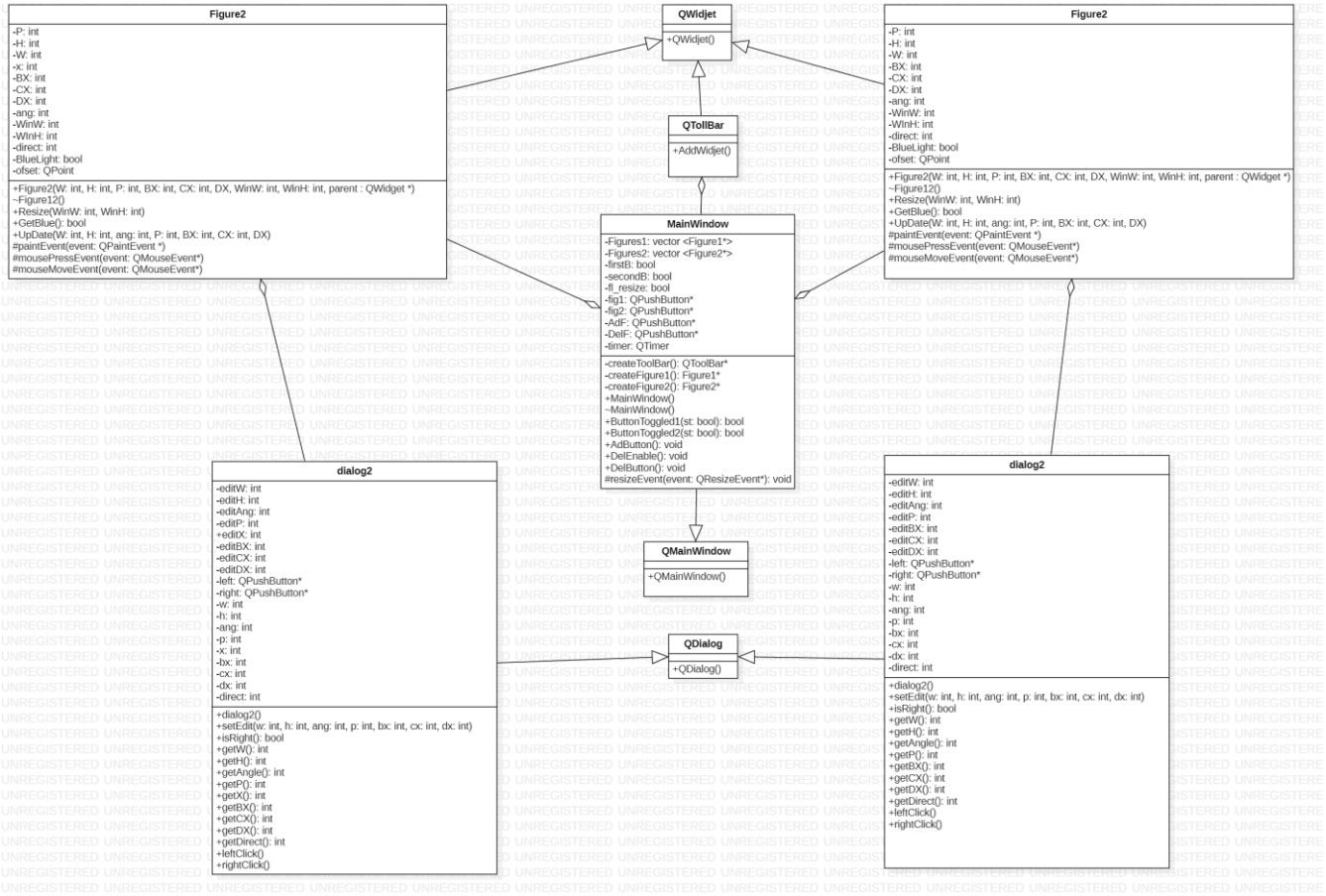


Рисунок 1 – UML 2.0 диаграмма классов

1.2 Поля классов

1.2.1 Класс MainWindow

vector <Figure1*> Figures1
Вектор содержащий все фигуры перового типа
vector <Figure2*> Figures2
Вектор содержащий все фигуры второго типа
bool firstB
Триггер нажатия кнопки, выбирающей первую фигуру
bool second
Триггер нажатия кнопки, выбирающей вторую фигуру
bool fl_resize
Триггер для перерисовки фигур
QPushButton*fig1
Кнопка для отрисовки фигуры 1 типа

QPushButton*fig2
Кнопка для отрисовки фигуры 2 типа
QPushButton* AdF
Кнопка для добавления фигуры выбранного типа
QPushButton*DelF
Кнопка для удаления выбранной фигуры
QTimer timer
Таймер проверки выделения фигур

1.2.2 Класс Figure1

int Q
Размер верхней середины нужного типа
int H
Высота фигуры
int W
Длина Фигуры
int x
Размер угла нужного типа
int AR
Размер угла нужного типа
int BR
Размер угла нужного типа
int CR
Размер угла нужного типа
int ang
Размер угла поворота фигуры
int WinW
Ширина главного экрана
int WinH

Высота главного экрана
int direct
Величина, определяющая направление поворота фигуры
bool BlueLight
триггер выделения фигуры
QPoint ofset
Используется для перемещения фигуры с зажатой клавишей мыши

1.2.3Класс Figure2

int P
Размер нижней середины нужного типа
int H
Высота фигуры
int W
Длина Фигуры
int BX
Размер угла нужного типа
int CX
Размер угла нужного типа
int DX
Размер угла нужного типа
int ang
Размер угла поворота фигуры
int WinW
Ширина главного экрана
int WinH
Высота главного экрана
int direct
Величина, определяющая направление поворота фигуры
bool BlueLight
триггер выделения фигуры
QPoint ofset
Используется для перемещения фигуры с зажатой клавишей мыши

1.2.4Dialog1

QLineEdit* editW
Строка ввода нового значения ширины фигуры
QLineEdit* editH
Строка ввода нового значения высоты фигуры
QLineEdit* editAng
Строка ввода нового значения угла поворота фигуры
QLineEdit* editQ
Строка ввода нового значения Q
QLineEdit* editX
Строка ввода нового значения X

QLineEdit* editAR
Строка ввода нового значения AR
QLineEdit* editBR
Строка ввода нового значения BR
QLineEdit* editCR
Строка ввода нового значения CR
QPushButton* left
Кнопка выбора направления поворота фигуры
QPushButton* right
Кнопка выбора направления поворота фигуры
int w
Содержит ширину фигуры
int h
Содержит высоту фигуры

int ang
Содержит угол поворота фигуры
int q
Содержит значение параметра q
int x
Содержит значение параметра x
int ar
Содержит значение параметра ar
int br
Содержит значение параметра br
int cr
Содержит значение параметра cr
int direct
Содержит направление поворота фигуры

1.2.5Dialog2

QLineEdit* editW
Строка ввода нового значения ширины фигуры
QLineEdit* editH
Строка ввода нового значения высоты фигуры
QLineEdit* editAng
Строка ввода нового значения угла поворота фигуры
QLineEdit* editP
Строка ввода нового значения Q
QLineEdit* editBX
Строка ввода нового значения BX
QLineEdit* editCX
Строка ввода нового значения CX
QLineEdit* editDX
Строка ввода нового значения DX

QPushButton* left
Кнопка выбора направления поворота фигуры
QPushButton* right
Кнопка выбора направления поворота фигуры
int w
Содержит ширину фигуры
int h
Содержит высоту фигуры
int ang
Содержит угол поворота фигуры
int p
Содержит значение параметра p
int bx
Содержит значение параметра bx
int cx
Содержит значение параметра cx

int cx
Содержит значение параметра cx
int direct
Содержит направление поворота фигуры

1.3 Методы классов
1.3.1 Класс MainWindow
createToolBar(): QToolBar*
Создает панель инструментов и добавляет кнопки
createFigure1(): Figure1*
Функция создает объект класса Figure1, который рисует на рабочем пространстве фигуру с произвольными характеристиками, и возвращает указатель на нее
createFigure2(): Figure2*
Функция создает объект класса Figure1, который рисует на рабочем пространстве фигуру с произвольными характеристиками, и возвращает указатель на нее
MainWindow()
Конструктор класса, создает главное окно
~MainWindow()
Деструктор Класса
ButtonToggled1(in st:bool): bool
Слот-обработчик зажатия кнопки выбора фигуры первого типа
ButtonToggled2(in st:bool): bool
Слот-обработчик зажатия кнопки выбора фигуры первого типа
AdButton(): void
Добавляет выбранную фигуру на экран

DelEnable(): void
Слот, который устанавливает доступность кнопки удаления фигуры если по истечению времени таймера какая-то фигура была выделена
DelButton(): void
Слот, который удаляет выбранную фигуру
resizeEvent(in event:QResizeEvent*): void
Переопределенный метод. Используется для изменения значений фигуры при изменении размеров окна

1.3.2 Класс Figure2

Figure2(W:int, H:int, P:int, BX:int, CX:int, DX, WinW:int, WinH:int, parent :QWidget *)
Конструктор класса, задает параметры фигуры
GetBlue(): bool
Функция передающая наличие выделения фигуры
Resize(in WinW:int, in WinH:int)
Метод изменяет значения ширины и высоты экрана при перерисовке
UpDate(in W:int, in H:int, in ang:int, in P:int, in BX:int, in CX:int, in DX)
Метод для обновления характеристик фигуры
paintEvent(in event:QPaintEvent *)
Переопределенная функция. Отрисовывает фигуры с заданными параметрами
mousePressEvent(in event:QMouseEvent*)
Переопределенная функция. Используется для выделения фигуры и вывода модального меню на экран
mouseMoveEvent(in event:QMouseEvent*)
Переопределенная функция. Используется для перетаскивания фигур с помощьюмыши.
~Figure2()
Деструктор класса

1.3.3 Класс Figure1

Figure1(W:int, H:int , Q:int, x:int, AR:int, BR:int, CR:int, WinW:int WinH:int parent :QWidget *)
Конструктор класса, задает параметры фигуры
GetBlue(): bool
Функция передающая наличие выделения фигуры
Resize(in WinW:int, in WinH:int)
Метод изменяет значения ширины и высоты экрана при перерисовке
UpDate(in W:int, in H:int, in ang:int, , Q:int, x:int, AR:int, BR:int, CR:int)
Метод для обновления характеристик фигуры
paintEvent(in event:QPaintEvent *)
Переопределенная функция. Отрисовывает фигуры с заданными параметрами
mousePressEvent(in event:QMouseEvent*)

Переопределенная функция. Используется для выделения фигуры и вывода модального меню на экран

mouseMoveEvent(in event:QMouseEvent*)

Переопределенная функция. Используется для перетаскивания фигур с помощьюмыши.

~Figure1()

Деструктор класса

1.3.4 Класс Dialog1

dialog1()

Конструктор класса, создающий модальное меню

setEdit(int w, int h, int ang, int q, int x, int ar, int br, int cr)

Метод считывающий параметры фигуры

isRight():bool

Метод проверяющий правильность введённых данных

getW():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getH():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getAngle():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getQ():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getX():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getAR():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getBR():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getCR():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getDirect():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

void leftClick()

Слот меняющий направление поворота фигуры

void rightClick()

Слот меняющий направление поворота фигуры

1.3.5 Класс Dialog2

dialog1()

Конструктор класса, создающий модальное меню

setEdit(int w, int h, int ang, int p, int bx, int cx, int dx)

Метод считывающий параметры фигуры

isRight():bool

Метод проверяющий правильность введённых данных

getW():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getH():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getAngle():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getP():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getBX():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getCX():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getDX():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

getDirect():int

Метод, возвращающий значение параметра фигуры, введенное пользователем

void leftClick()

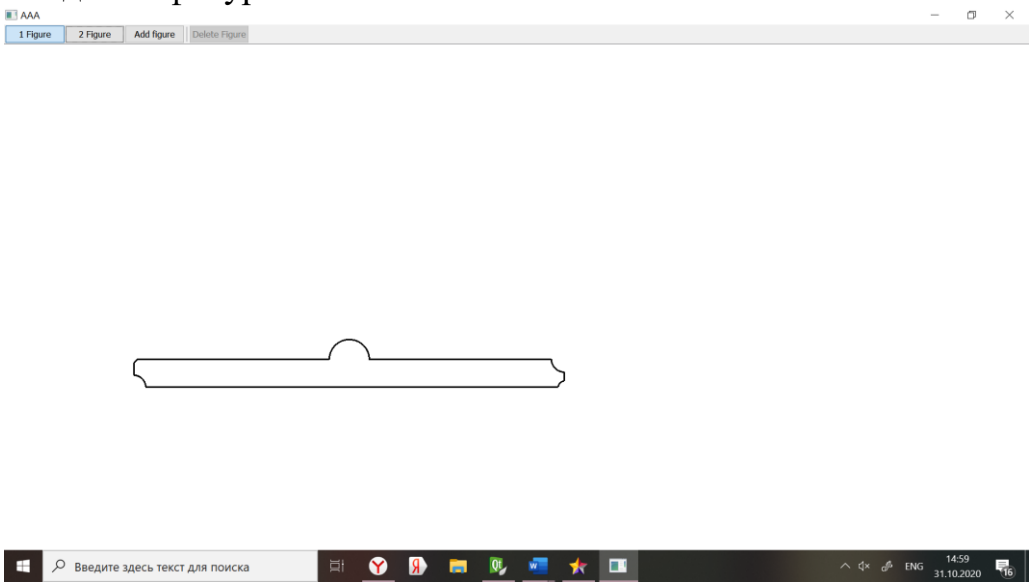
Слот меняющий направление поворота фигуры

void rightClick()

Слот меняющий направление поворота фигуры

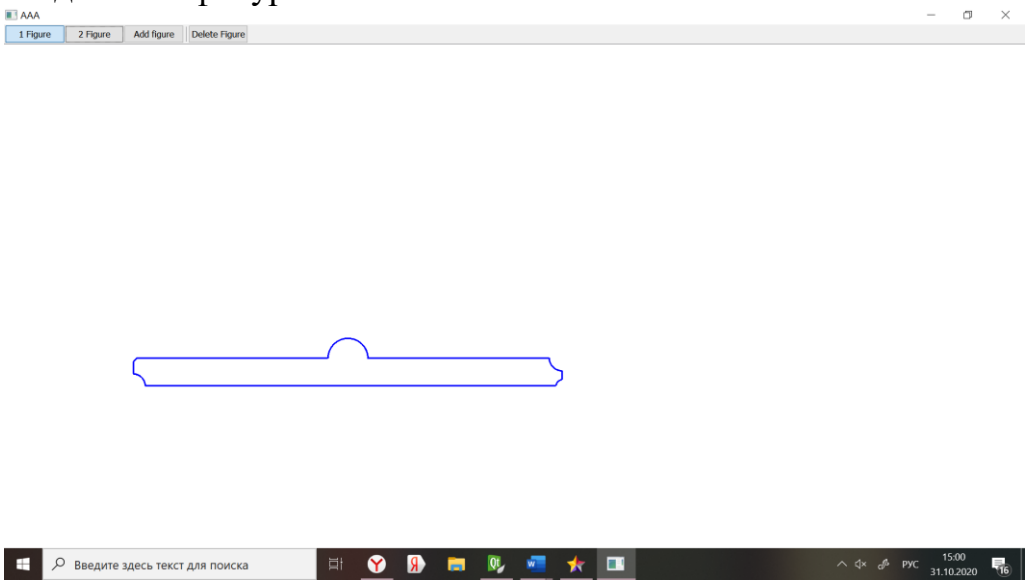
2.Тестирование программы

Создание фигуры



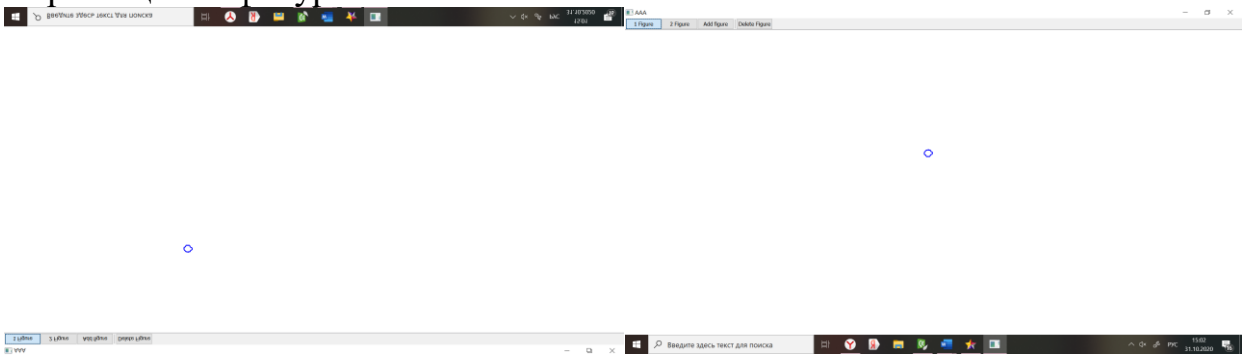
Листинг 1

Выделение фигуры



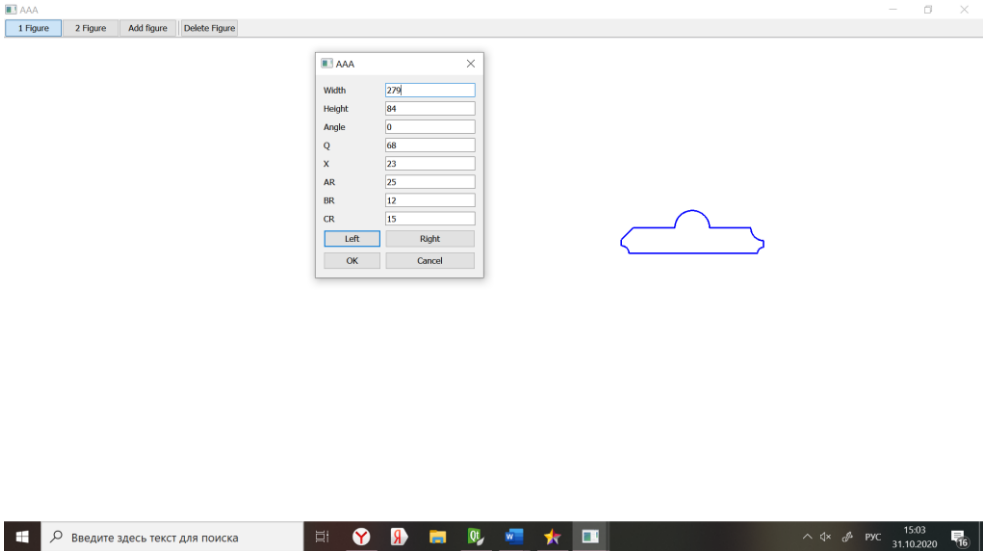
Листинг 2

Перемещение фигуры



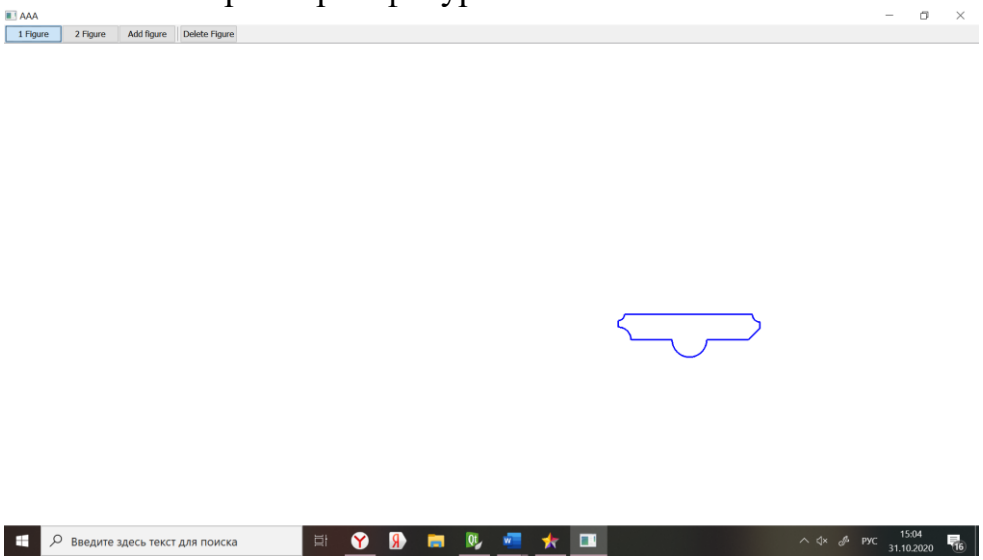
Листинг 3,4

Модальное меню



Листинг 5

Изменение параметров фигуры



3.Приложение
3.1.main.cpp

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.resize(800,500);
    w.show();

    return a.exec();
}
```

3.2.mainwinow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtGui>
#include <QToolBar>
#include <QActionGroup>
#include <QPalette>
#include <QPushButton>
#include <Figure1.h>
#include <Figure2.h>
#include <vector>
#include <QTimer>
#include <iterator>
using namespace std;
class MainWindow : public QMainWindow
{
    Q_OBJECT

private:
    vector <Figure1*> Figures1;
    vector <Figure2*> Figures2;
    bool firstB;
    bool secondB;
    bool fl_resize;

    QPushButton*fig1;
    QPushButton*fig2;
    QPushButton* AdF;
    QPushButton*DelF;

    QToolBar* createToolBar();
    Figure1* createFigure1();
```

```
    Figure2* createFigure2();

    QTimer timer;
public:
    MainWindow();
    ~MainWindow();
 public slots:
 void ButtonToggled1(bool st);
 void ButtonToggled2(bool st);
 void AdButton();
 void DelEnable();
 void DelButton();
protected:
    void resizeEvent(QResizeEvent* event);
};

#endif // MAINWINDOW_H
```

3.3.mainwinow.cpp

```
#include "mainwindow.h"

QToolBar* MainWindow::createToolBar()
{
    QToolBar *toolbar = addToolBar("main toolbar");
    toolbar->setMovable(false);
    QPalette toolBarPal(palette());
    toolBarPal.setColor(QPalette::Background, Qt::gray);
    toolbar->setAutoFillBackground(true);
    toolbar->setPalette(toolBarPal);

    this->fig1 = new QPushButton("1 Figure",this);
    this->fig2 = new QPushButton("2 Figure",this);
    this->AdF  = new QPushButton("Add figure", this);
    toolbar->addWidget(fig1);
    toolbar->addWidget(fig2);
    toolbar->addWidget(AdF);
    toolbar->addSeparator();

    this->DelF = new QPushButton("Delete Figure",this);
    toolbar->addWidget(DelF);
    this->DelF->setEnabled(false);
    connect(DelF,SIGNAL(clicked()),this,SLOT(DelButton()));

    this->fig1->setCheckable(true);
    connect(fig1, SIGNAL(toggled(bool)), this,
SLOT(ButtonToggled1(bool)));

    this->fig2->setCheckable(true);
    connect(fig2, SIGNAL(toggled(bool)), this,
SLOT(ButtonToggled2(bool)));
```

```
    this->AdF->setEnabled(false);
    connect(AdF,SIGNAL(clicked()),this,SLOT(AdButton()));
    return toolbar;

}
MainWindow::MainWindow()
    : QMainWindow(),fl_resize(false)
{
    this->move(400,0);
    this->resize(1000,800);

    QPalette WindowPal(palette());
    WindowPal.setColor(QPalette::Background, Qt::white);
    this->setPalette(WindowPal);

    addToolBar(Qt::TopToolBarArea, createToolBar());

    connect(&timer, SIGNAL(timeout()), this, SLOT(DelEnable()));
    timer.start(5);
}

void MainWindow::ButtonToggled1(bool state)
{
    this->firstB=state;
    if(state)
    {
        if(this->fig2->isChecked()){this->fig2-
>setChecked(false);}
        this->AdF->setEnabled(true);
    }
else
    {
        if((this->firstB)==false &&(this->secondB)==false){this-
>AdF->setEnabled(false);}
   }

}
void MainWindow::ButtonToggled2(bool state)
{
    this->secondB=state;
    if(state)
    {
     if(this->fig1->isChecked()){this->fig1->setChecked(false);}
     this->AdF->setEnabled(true);
    }
else
    {
     if((this->secondB)==false &&(this->firstB)==false){this-
>AdF->setEnabled(false);}
   }

}
Figure1* MainWindow::createFigure1()
```

```
{
    int StX=35+(rand() % (width() - 200)) ;
    int StY=35+(rand() % (height() - 200)) ;
    int W=4+(rand() % (width() - StX)) ;
    int H=3+(rand() % (std::min(W-3, height() - StY)));
    int Q=1+(rand() % (W/4));
    int x=1+(rand() % (H/3)) ;
    int AR=1+(rand() % (H/3)) ;
    int BR=1+(rand() % (H/3)) ;
    int CR=1+(rand() % (H/3)) ;
    Figure1* figure = new Figure1( W, H, Q, x, AR,
BR,CR,width(),height(), this);
   figure->show();
   figure->move(StX,StY);
   figure->resize(width(),height());
   return figure;
}
Figure2* MainWindow::createFigure2()
{
    int StX=35+rand() % (width() - 200) ;
    int StY=35+rand() % (height() - 200) ;
    int W=4+rand() % (width() - StX) ;
    int H=3+rand() % (std::min(W-3, height() - StY));
    int P=1+rand() % (W/4);
    int BX=1+rand() % (H/3) ;
    int CX=1+rand() % (H/3) ;
    int DX=1+rand() % (H/3) ;
    Figure2* figure = new Figure2(W, H, P,BX,
CX,DX,width(),height(), this);
   figure->show();
   figure->move(StX,StY);
   figure->resize(width(),height());
   return figure;
}

void MainWindow::AdButton()
{
    if((firstB==true)&&(secondB==false))
    {
    Figures1.push_back(createFigure1());
    }
    else if((secondB==true)&&(firstB==false))
    {
    Figures2.push_back(createFigure2());
    }
}

void MainWindow::resizeEvent(QResizeEvent* event)
{
    Q_UNUSED(event);
    if (!fl_resize)
    {
        fl_resize = true;
```

```
    }
    else
    {
        for (size_t i=0;i<Figures1.size();i++)
        {
            Figures1[i]->Resize(width(), height());
            Figures1[i]->repaint();
        }
        for (size_t i=0;i<Figures2.size();i++)
        {
            Figures2[i]->Resize(width(), height());
            Figures2[i]->repaint();
        }

    }
}

void MainWindow::DelButton()
{
    vector <Figure1*> :: iterator it1 ;
    vector <Figure2*> :: iterator it2 ;
    if (DelF->isEnabled())
    {
    for (int i = Figures1.size() - 1; i >= 0; --i)
    {
        if (Figures1[i]->GetBlue())
        {
        delete Figures1[i];
        it1 = (Figures1.begin()+i);
        Figures1.erase(it1);
        }
    }

    for (int i = Figures2.size() - 1; i >= 0; --i)
    {
        if (Figures2[i]->GetBlue())
        {
        delete Figures2[i];
        it2 =( Figures2.begin() + i);
        Figures2.erase(it2);
        }
    }
    }
    }
void MainWindow::DelEnable()
{
    bool trigger = false;
    for (size_t i=0;i<Figures1.size();i++)
    {
    if (Figures1[i]->GetBlue())
    {
        trigger = true;
        break;
```

```
    }
    }
    for (size_t i=0;i<Figures2.size();i++)
    {
    if (Figures2[i]->GetBlue())
    {
        trigger = true;
        break;
    }
    }

    if (trigger) {DelF->setEnabled(true);}
    else {DelF->setEnabled(false);}
}
MainWindow::~MainWindow(){}
```

3.4 Figure1.h

```
#ifndef PAINTFIGURE_H
#define PAINTFIGURE_H
#include <QPainter>
#include <QWidget>
#include <QPainter>
#include <QWidget>
#include <QMouseEvent>
#include <cmath>
#include<dialog1.h>
class Figure1 : public QWidget {
Q_OBJECT
  private:
    int Q;
    int H;
    int W;
    int x;
    int AR;
    int BR;
    int CR;
    int ang;
    int WinW;
    int WinH;
    int direct;
    bool BlueLight;
    QPoint ofset;

public:
    Figure1(int W,int H,int Q,int x,int AR,int BR,int CR,int
WinW,int WinH,QWidget *parent=0);
    ~Figure1();
    void Resize(int WinW,int WinH);
    bool GetBlue();
    void UpDate(int W,int H,int ang,int Q,int x,int AR,int
BR,int CR);
```

```
protected:
   void paintEvent(QPaintEvent *event);
   void mousePressEvent(QMouseEvent* event);
   void mouseMoveEvent(QMouseEvent* event);
};

#endif // PAINTFIGURE_H
```

3.5 Figure1.cpp

```
#include <Figure1.h>

void Figure1::paintEvent(QPaintEvent *e)
{
    Q_UNUSED(e);
    QPainter painter(this);
    resize(W * sqrt(2) + 1, W * sqrt(2) + 1);

    painter.translate(W * sqrt(2)/2, W * sqrt(2)/2);
    painter.rotate(ang*direct);
    painter.translate(-W/2,-W/2);
    if(BlueLight)
    {
     painter.setPen(QPen(Qt::blue, 3, Qt::SolidLine,
Qt::FlatCap));
    }
    else
    {
    painter.setPen(QPen(Qt::black, 3, Qt::SolidLine,
Qt::FlatCap));
    }
    painter.drawLine(x, Q/2, W/2 - Q/2, Q/2);
    painter.drawArc(W/2 - Q/2, 0, Q, Q, +180 * 16, -180 * 16);
    painter.drawLine(W/2 + Q/2, Q/2, W - AR, Q/2);
    painter.drawArc(W-AR, Q/2-AR, 2 * AR, 2 * AR, 180 * 16, 90 *
16);
    painter.drawLine(W, Q/2+AR, W, H - BR);
    painter.drawArc(W-BR, H-BR, 2 * BR, 2 * BR, 90 * 16, 90 *
16);
    painter.drawLine(W-BR, H, CR, H);
    painter.drawArc(-CR, H-CR, 2 * CR, 2 * CR, 0, 90 * 16);
    painter.drawLine(0, H -CR, 0, x+Q/2);
    painter.drawLine(0,x+Q/2,x,Q/2);
}
Figure1::Figure1(int W,int H,int Q,int x,int AR,int BR,int
CR,int WinW,int WinH,QWidget *parent):
    QWidget(parent),BlueLight(false),ang(0),direct(1)
{
    this->WinW=WinW;
    this->WinH=WinH;
    this->W=W;
    this->H=H;
```

```
    this->Q=Q;
    this->x=x;
    this->AR=AR;
    this->BR=BR ;
    this->CR=CR;
}
void Figure1::Resize(int WinW,int WinH)
{
  this->WinW=WinW;
  this->WinH=WinH;
  QWidget::update();
}

bool Figure1::GetBlue()
{
 return BlueLight;
}

void Figure1::mousePressEvent(QMouseEvent* event)
{
    if (event->buttons() & Qt::LeftButton)
    {
    if (BlueLight){ BlueLight= false;}
    else BlueLight = true;
    QWidget::update();

    ofset = event->globalPos() - pos();
    }
    if (event->buttons() & Qt::RightButton)
    {
    if (!BlueLight) BlueLight = true;
    QWidget::update();
    dialog1* dialog = new dialog1();
    dialog->move(600,100);
    dialog->setEdit(W, H, ang, Q, x, AR, BR, CR);
    dialog->show();

    if ((dialog->exec() == QDialog::Accepted))
    {
        if (!dialog->isRight())
        {
        QMessageBox::information(0,"Information", "Incorrect
data");
        }
        else
        {
        UpDate(dialog->getW(), dialog->getH(), dialog-
>getAngle(), dialog->getQ(), dialog->getX()
               , dialog->getAR(), dialog->getBR(), dialog-
>getCR());
        direct = dialog->getDirect();
        QWidget::update();
        }
```

```
        }

    delete dialog;
    }
  }
void Figure1::mouseMoveEvent(QMouseEvent* event)
{
    BlueLight = true;
    QWidget::update();

    if (event->buttons() & Qt::LeftButton)
    {
    if (((event->globalPos() - ofset).x() + width() > WinW) ||
        ((event->globalPos() - ofset).y() + height() > WinH) ||
        ((event->globalPos() - ofset).x() <= 0) ||
        ((event->globalPos() - ofset).y() <= 35))
        ofset = event->globalPos() - pos();
    else
        move(event->globalPos() - ofset);
    }
}

 void Figure1:: UpDate(int W,int H,int ang,int Q,int x,int
AR,int BR,int CR)
 {
     this->W=W;
     this->H=H;
     this->ang=ang;
     this->Q=Q;
     this->x=x;
     this->AR=AR;
     this->BR=BR;
     this->CR=CR;
 }
Figure1:: ~Figure1(){}
```

3.6 Figure2.h

```
#ifndef FIGURE2_H
#define FIGURE2_H

#include <QPainter>
#include <QWidget>
#include <QPainter>
#include <QWidget>
#include <QMouseEvent>
#include <cmath>
#include <dialog2.h>
class Figure2 : public QWidget
{
Q_OBJECT
      private:
```

```
        int H;
        int W;
        int P;
        int BX;
        int CX;
        int DX;
        int ang;
        int WinW;
        int WinH;
        int direct;
        bool BlueLight;
        QPoint ofset;
     public:
        Figure2(int W,int H,int P,int BX,int CX,int DX,int
WinW,int WinH,QWidget *parent=0);
         ~Figure2();
        void Resize(int WinW,int WinH);
        bool GetBlue();
        void UpDate(int W,int H,int ang,int P,int BX,int CX,int
DX);

     protected:
        void paintEvent(QPaintEvent *event);
        void mousePressEvent(QMouseEvent* event);
        void mouseMoveEvent(QMouseEvent* event);
};

#endif // FIGURE2_H
```

3.7 Figure2.cpp

```
#include "Figure2.h"
void Figure2::paintEvent(QPaintEvent *e)
{
     Q_UNUSED(e);
    QPainter painter(this);
    resize(W * sqrt(2) + 1, W * sqrt(2) + 1);

    painter.translate(W * sqrt(2)/2, W * sqrt(2)/2);
    painter.rotate(ang*direct);
    painter.translate(-W/2,-W/2);
    if(BlueLight)
    {
     painter.setPen(QPen(Qt::blue, 3, Qt::SolidLine,
Qt::FlatCap));
    }
    else
    {
    painter.setPen(QPen(Qt::black, 3, Qt::SolidLine,
Qt::FlatCap));
    }
    painter.drawLine(DX, 0, W,0);
```

```
    painter.drawLine(W,0, W,H-BX);
    painter.drawLine(W, H-BX, W-BX,H-BX);
    painter.drawLine(W-BX, H-BX, W-BX, H);
    painter.drawLine(W-BX, H, W/2 + P/2, H);
    painter.drawLine(W/2 + P/2, H, W/2 + P/2, H + P/2);
    painter.drawLine(W/2 + P/2, H + P/2, W/2 - P/2, H + P/2);
    painter.drawLine(W/2 - P/2, H + P/2, W/2 - P/2, H);
    painter.drawLine(W/2 - P/2, H, CX, H);
    painter.drawLine(CX, H, 0, H - CX);
    painter.drawLine(0, H- CX, 0, DX);
    painter.drawLine(0, DX, DX, DX);
    painter.drawLine(DX,DX,DX,0);

}
Figure2::Figure2(int W,int H,int P,int BX,int CX,int DX,int
WinW,int WinH,QWidget *parent):
    QWidget(parent),BlueLight(false),ang(0),direct(1)
{
    this->WinW=WinW;
    this->WinH=WinH;
    this->W=W;
    this->H=H;
    this->P=P;
    this->BX=BX;
    this->CX=CX ;
    this->DX=DX;
}
 void Figure2::Resize(int WinW,int WinH)
{
  this->WinW=WinW;
  this->WinH=WinH;
}

 bool Figure2::GetBlue()
 {
  return BlueLight;
 }

 void Figure2::mousePressEvent(QMouseEvent* event)
 {
     if (event->buttons() & Qt::LeftButton)
     {
     if (BlueLight){ BlueLight= false;}
     else BlueLight = true;
     QWidget::update();

     ofset = event->globalPos() - pos();
     }
     if (event->buttons() & Qt::RightButton)
     {
     if (!BlueLight) BlueLight = true;
     QWidget::update();
     if (!BlueLight) BlueLight = true;
```

```
     QWidget::update();
     dialog2* dialog = new dialog2();
     dialog->move(600,100);
     dialog->setEdit(W, H, ang, P, BX, CX, DX);
     dialog->show();

     if ((dialog->exec() == QDialog::Accepted))
     {
         if (!dialog->isRight())
         {
         QMessageBox::information(0,"Information", "Incorrect
data");
         }
         else
         {
         UpDate(dialog->getW(), dialog->getH(), dialog-
>getAngle(), dialog->getP(), dialog->getBX()
                , dialog->getCX(), dialog->getDX());
         direct = dialog->getDirect();
         QWidget::update();
         }
         }

     delete dialog;
     }
 }
void Figure2:: mouseMoveEvent(QMouseEvent* event)
{
    BlueLight = true;
    QWidget::update();

    if (event->buttons() & Qt::LeftButton)
    {
    if (((event->globalPos() - ofset).x() + width() > WinW) ||
        ((event->globalPos() - ofset).y() + height() > WinH) ||
        ((event->globalPos() - ofset).x() <= 0) ||
        ((event->globalPos() - ofset).y() <= 35))
        ofset = event->globalPos() - pos();
    else
        move(event->globalPos() - ofset);
    }
}
void Figure2:: UpDate(int W,int H,int ang,int P,int BX,int
CX,int DX)
{
    this->W=W;
    this->H=H;
    this->ang=ang;
    this->P=P;
    this->BX=BX;
    this->CX=CX;
    this->DX=DX;
}
```

```
Figure2:: ~Figure2()
{
}
```

3.8 dialog1.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QLabel>
#include <QtGui>
class dialog1 : public QDialog
{
Q_OBJECT
private:
        QLineEdit* editW;
        QLineEdit* editH;
        QLineEdit* editAng;
        QLineEdit* editQ;
        QLineEdit* editX;
        QLineEdit* editAR;
        QLineEdit* editBR;
        QLineEdit* editCR;


        QPushButton* left;
        QPushButton* right;

        int w;
        int h;
        int ang;
        int q;
        int x;
        int ar;
        int br;
        int cr;
        int direct;

    public:
        dialog1();

        void setEdit(int w, int h, int ang, int q, int x, int
ar, int br, int cr);

        bool isRight();

        int getW();
        int getH();
        int getAngle();
        int getQ();
```

```
        int getX();
        int getAR();
        int getBR();
        int getCR();
        int getDirect();

public slots:
    void leftClick();
    void rightClick();

};

#endif // DIALOG_H
```

3.9 dialog1.cpp

```
#include "dialog1.h"

dialog1::dialog1() : QDialog(0, Qt::WindowTitleHint |
Qt::WindowSystemMenuHint)
{
    direct = 1;
    editW = new QLineEdit;
    editH = new QLineEdit;
    editAng = new QLineEdit;
    editQ = new QLineEdit;
    editX = new QLineEdit;
    editAR = new QLineEdit;
    editBR = new QLineEdit;
    editCR = new QLineEdit;

    QLabel* labelW = new QLabel("Width");
    QLabel* labelH = new QLabel("Height");
    QLabel* labelAngle = new QLabel("Angle");

    QLabel* labelQ= new QLabel("Q");
    QLabel* labelX= new QLabel("X");
    QLabel* labelAR= new QLabel("AR");
    QLabel* labelBR= new QLabel("BR");
    QLabel* labelCR= new QLabel("CR");

    left = new QPushButton("Left");
    left->setCheckable(true);
    connect(left, SIGNAL(clicked()), this, SLOT(leftClick()));

    right = new QPushButton("Right");
    right->setCheckable(true);
    connect(right, SIGNAL(clicked()), this, SLOT(rightClick()));

    QPushButton* btnOK = new QPushButton("OK");
    QPushButton* btnCancel = new QPushButton("Cancel");
```

```
    connect(btnOK, SIGNAL(clicked()), SLOT(accept()));
    connect(btnCancel, SIGNAL(clicked()), SLOT(reject()));

    QGridLayout* layout = new QGridLayout;
    layout->addWidget(labelW, 0, 0);
    layout->addWidget(editW, 0, 1);

    layout->addWidget(labelH, 1, 0);
    layout->addWidget(editH, 1, 1);

    layout->addWidget(labelAngle, 2, 0);
    layout->addWidget(editAng, 2, 1);

    layout->addWidget(labelQ, 3, 0);
    layout->addWidget(editQ, 3, 1);

    layout->addWidget(labelX, 4, 0);
    layout->addWidget(editX, 4, 1);

    layout->addWidget(labelAR, 5, 0);
    layout->addWidget(editAR, 5, 1);

    layout->addWidget(labelBR, 6, 0);
    layout->addWidget(editBR, 6, 1);

    layout->addWidget(labelCR, 7, 0);
    layout->addWidget(editCR, 7, 1);

    layout->addWidget(left, 8, 0);
    layout->addWidget(right, 8, 1);

    layout->addWidget(btnOK, 9, 0);
    layout->addWidget(btnCancel, 9, 1);

    setLayout(layout);
}

void dialog1::setEdit(int w, int h, int ang, int q, int x, int
ar, int br, int cr)
{
    editW->setText(QString::number(w));
    editH->setText(QString::number(h));
    editAng->setText(QString::number(ang));
    editQ->setText(QString::number(q));
    editX->setText(QString::number(x));
    editAR->setText(QString::number(ar));
    editBR->setText(QString::number(br));
    editCR->setText(QString::number(cr));
}

bool dialog1::isRight()
{
    bool check;
```

```
    QString temp = editW->text();
    int w = temp.toInt(&check, 10);
    if (!check || w == 0) return false;

    temp = editH->text();
    int h = temp.toInt(&check, 10);
    if (!check || h == 0 || h > w) return false;

    temp = editAng->text();
    int angle = temp.toInt(&check, 10);
    if (!check) return false;

    temp = editQ->text();
    int q = temp.toInt(&check, 10);
    if (!check || q == 0 || q > w / 4) return false;

    temp = editX->text();
    int x = temp.toInt(&check, 10);
    if (!check || x == 0 || x > h / 3) return false;

    temp = editAR->text();
    int ar = temp.toInt(&check, 10);
    if (!check || ar == 0 || ar > h / 3) return false;

    temp = editBR->text();
    int br = temp.toInt(&check, 10);
    if (!check || br == 0 || br > h / 3) return false;

    temp = editCR->text();
    int cr = temp.toInt(&check, 10);
    if (!check || cr == 0 || cr > h / 3) return false;

    this->w = w;
    this->h = h;
    this->ang = angle;
    this->q = q;
    this->x = x;
    this->ar = ar;
    this->br = br;
    this->cr = cr;

    return true;
}

int dialog1::getW()
{
    return w;
}

int dialog1::getH()
{
    return h;
}
```

```
int dialog1::getAngle()
{
    return ang;
}

int dialog1::getQ()
{
    return q;
}

int dialog1::getX()
{
    return x;
}

int dialog1::getAR()
{
    return ar;
}

int dialog1::getBR()
{
    return br;
}

int dialog1::getCR()
{
    return cr;
}

int dialog1::getDirect()
{
    return direct;
}

void dialog1::leftClick()
{
    if (left->isChecked())
    {
    if (right->isChecked())
    {
        right->setChecked(false);
    }
    direct = 1;
    }
}

void dialog1::rightClick()
{
    if (right->isChecked())
    {
    if (left->isChecked())
```

```
    {
        left->setChecked(false);
    }
    direct = -1;
    }
}
```

3.10 dialog2.h

```
#ifndef DIALOG2_H
#define DIALOG2_H

#include <QDialog>
#include <QLabel>
#include <QtGui>
class dialog2 : public QDialog
{
Q_OBJECT
private:
        QLineEdit* editW;
        QLineEdit* editH;
        QLineEdit* editAng;
        QLineEdit* editP;
        QLineEdit* editBX;
        QLineEdit* editCX;
        QLineEdit* editDX;


        QPushButton* left;
        QPushButton* right;

        int w;
        int h;
        int ang;
        int p;
        int bx;
        int cx;
        int dx;
        int direct;

    public:
        dialog2();

        void setEdit(int w, int h, int ang, int p, int bx, int
cx, int dx);

        bool isRight();

        int getW();
        int getH();
        int getAngle();
        int getP();
```

```
        int getBX();
        int getCX();
        int getDX();
        int getDirect();

public slots:
    void leftClick();
    void rightClick();

};

#endif // DIALOG2_H
```

3.11 dialog1.cpp

```
#include "dialog2.h"

dialog2::dialog2() : QDialog(0, Qt::WindowTitleHint |
Qt::WindowSystemMenuHint)
{
    direct = 1;
    editW = new QLineEdit;
    editH = new QLineEdit;
    editAng = new QLineEdit;
    editP = new QLineEdit;
    editBX = new QLineEdit;
    editCX = new QLineEdit;
    editDX = new QLineEdit;

    QLabel* labelW = new QLabel("Width");
    QLabel* labelH = new QLabel("Height");
    QLabel* labelAngle = new QLabel("Angle");

    QLabel* labelP= new QLabel("P");
    QLabel* labelBX= new QLabel("BX");
    QLabel* labelCX= new QLabel("CX");
    QLabel* labelDX= new QLabel("DX");

    left = new QPushButton("Left");
    left->setCheckable(true);
    connect(left, SIGNAL(clicked()), this, SLOT(leftClick()));

    right = new QPushButton("Right");
    right->setCheckable(true);
    connect(right, SIGNAL(clicked()), this, SLOT(rightClick()));

    QPushButton* btnOK = new QPushButton("OK");
    QPushButton* btnCancel = new QPushButton("Cancel");

    connect(btnOK, SIGNAL(clicked()), SLOT(accept()));
    connect(btnCancel, SIGNAL(clicked()), SLOT(reject()));
```

```
    QGridLayout* layout = new QGridLayout;
    layout->addWidget(labelW, 0, 0);
    layout->addWidget(editW, 0, 1);

    layout->addWidget(labelH, 1, 0);
    layout->addWidget(editH, 1, 1);

    layout->addWidget(labelAngle, 2, 0);
    layout->addWidget(editAng, 2, 1);

    layout->addWidget(labelP, 3, 0);
    layout->addWidget(editP, 3, 1);

    layout->addWidget(labelBX, 4, 0);
    layout->addWidget(editBX, 4, 1);

    layout->addWidget(labelCX, 5, 0);
    layout->addWidget(editCX, 5, 1);

    layout->addWidget(labelDX, 6, 0);
    layout->addWidget(editDX, 6, 1);

    layout->addWidget(left, 7, 0);
    layout->addWidget(right, 7, 1);

    layout->addWidget(btnOK, 8, 0);
    layout->addWidget(btnCancel, 8, 1);

    setLayout(layout);
}

void dialog2::setEdit(int w, int h, int ang, int p, int bx, int
cx, int dx)
{
    editW->setText(QString::number(w));
    editH->setText(QString::number(h));
    editAng->setText(QString::number(ang));
    editP->setText(QString::number(p));
    editBX->setText(QString::number(bx));
    editCX->setText(QString::number(cx));
    editDX->setText(QString::number(dx));
}

bool dialog2::isRight()
{
    bool check;
    QString temp = editW->text();
    int w = temp.toInt(&check, 10);
    if (!check || w == 0) return false;

    temp = editH->text();
    int h = temp.toInt(&check, 10);
    if (!check || h == 0 || h > w) return false;
```

```
    temp = editAng->text();
    int angle = temp.toInt(&check, 10);
    if (!check) return false;

    temp = editP->text();
    int p = temp.toInt(&check, 10);
    if (!check || p == 0 || p > w / 4) return false;

    temp = editBX->text();
    int bx = temp.toInt(&check, 10);
    if (!check || bx == 0 || bx > h / 3) return false;

    temp = editCX->text();
    int cx = temp.toInt(&check, 10);
    if (!check || cx == 0 || cx > h / 3) return false;

    temp = editDX->text();
    int dx = temp.toInt(&check, 10);
    if (!check || dx == 0 || dx > h / 3) return false;

    this->w = w;
    this->h = h;
    this->ang = angle;
    this->p = p;
    this->bx = bx;
    this->cx = cx;
    this->dx = dx;

    return true;
}

int dialog2::getW()
{
    return w;
}

int dialog2::getH()
{
    return h;
}

int dialog2::getAngle()
{
    return ang;
}

int dialog2::getP()
{
    return p;
}
```

```
int dialog2::getBX()
{
    return bx;
}

int dialog2::getCX()
{
    return cx;
}

int dialog2::getDX()
{
    return dx;
}

int dialog2::getDirect()
{
    return direct;
}

void dialog2::leftClick()
{
    if (left->isChecked())
    {
    if (right->isChecked())
    {
        right->setChecked(false);
    }
    direct = 1;
    }
}

void dialog2::rightClick()
{
    if (right->isChecked())
    {
    if (left->isChecked())
    {
        left->setChecked(false);
    }
    direct = -1;
    }
}
```