**airbase.h**

#ifndef AIRBASE\_H

#define AIRBASE\_H

struct airbase{

QString name;

qreal lon;

qreal lat;

qreal course;

QVector <qreal> Svrvb\_vec;

qreal airbase\_point\_range;

};

#endif // AIRBASE\_H

**animatedlabel.cpp**

#include "animatedlabel.h"

AnimatedLabel::AnimatedLabel(QWidget \*parent, QString path\_to\_gif, QString whats\_this, int minw, int minh) : QLabel(parent)

{

this->setMinimumSize(minw,minh);

this->setMaximumSize(minw,minh);

movie = new QMovie(path\_to\_gif);

this->setMovie(movie);

//this->setStyleSheet("background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(74, 74, 74, 255),stop:1 rgba(49, 49, 49, 255));"

// "border: 0px solid #fff;");

this->setStyleSheet("background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(184, 184, 184, 255),stop:1 rgba(159, 159, 159, 255));"

"border-style: solid;"

"border-width: 1px;"

"border-color: #051a39;"

"padding: 5px;");

this->setScaledContents(true);

this->setCursor(Qt::PointingHandCursor);

this->setMouseTracking(true);

Filter\* filter\_start\_stop = new Filter(this);

this->installEventFilter(filter\_start\_stop);

this->setStatusTip(whats\_this);

//this->setStatusTip("Привет");

connect(filter\_start\_stop,SIGNAL(start\_movie\_signal()),this,SLOT(start\_movie()));

connect(filter\_start\_stop,SIGNAL(stop\_movie\_signal()),this,SLOT(stop\_movie()));

movie->start();

movie->stop();

}

AnimatedLabel::start\_movie()

{

movie->start();

this->setStyleSheet("background-color: rgba(70,162,218,50%);"

"border: 1px solid #46a2da;");

}

AnimatedLabel::stop\_movie()

{

movie->stop();

movie->start();

movie->stop();

this->setStyleSheet("background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(184, 184, 184, 255),stop:1 rgba(159, 159, 159, 255));"

"border-style: solid;"

"border-width: 1px;"

"border-color: #051a39;"

"padding: 5px;");

this->setFrameShape(WinPanel);

}

void AnimatedLabel::mousePressEvent(QMouseEvent \*event)

{

emit clicked();

}

void AnimatedLabel::mouseMoveEvent(QMouseEvent \*event)

{

}

bool Filter::eventFilter(QObject \*obj, QEvent \*e)

{

if (!obj->isWidgetType()) return true;

QWidget\* w = (QWidget\*) obj;

if (e->type() == QEvent::Enter) {

emit start\_movie\_signal();

}

else if (e->type() == QEvent::Leave) {

emit stop\_movie\_signal();

}

return false;

}

**animatedlabel.h**

#ifndef ANIMATEDLABEL\_H

#define ANIMATEDLABEL\_H

#include <QObject>

#include <QLabel>

#include <QMouseEvent>

#include <QFocusEvent>

#include <QDebug>

#include <QMovie>

class AnimatedLabel : public QLabel

{

Q\_OBJECT

public:

AnimatedLabel(QWidget \*parent, QString path\_to\_gif, QString whats\_this, int minw, int minh);

QMovie \*movie;

signals:

clicked();

public slots:

start\_movie();

stop\_movie();

protected:

virtual void mousePressEvent(QMouseEvent \*event) Q\_DECL\_OVERRIDE;

virtual void mouseMoveEvent(QMouseEvent \*event) Q\_DECL\_OVERRIDE;

};

class Filter : public QObject

{

Q\_OBJECT

public:

Filter( QObject\* parent = 0 ) : QObject(parent) {}

bool eventFilter( QObject\* obj, QEvent\* e );

signals:

start\_movie\_signal();

stop\_movie\_signal();

};

#endif // ANIMATEDLABEL\_H

**calc.cpp**

#include "calc.h"

calc::calc()

{

}

QVector<plane> calc::count\_all\_ranges\_plane\_airbase(QVector<plane> cur\_planes, QVector<airbase> cur\_airbases)

{

for (int i=0;i<cur\_planes.size();i++)

{

cur\_planes[i].range\_to\_airbases.clear();

foreach (auto one\_airbase, cur\_airbases)

{

cur\_planes[i].range\_to\_airbases.append(get\_range(cur\_planes[i],one\_airbase));

}

}

return get\_main\_plane(cur\_planes);

}

QVector<plane> calc::get\_main\_plane(QVector<plane> cur\_planes)

{

QVector <qreal> avg\_ranges;

foreach (auto p, cur\_planes)

{

qreal sum = 0;

foreach (auto range, p.range\_to\_airbases)

{

sum+=range;

qDebug()<<"Дальность"<<range;

}

avg\_ranges.append(sum/p.range\_to\_airbases.size());

}

int max\_index = get\_index\_of\_max(avg\_ranges);

qDebug()<<max\_index;

for (int i=0;i<cur\_planes.size();i++)

{

if (i==max\_index) cur\_planes[i].main\_plane=true;

else cur\_planes[i].main\_plane=false;

}

return cur\_planes;

}

int calc::get\_index\_of\_max(QVector<qreal> vec)

{

qreal max = vec.first();

int index\_of\_max = 0;

for (int i=0;i<vec.size();i++)

{

if (vec[i]>max)

{

max=vec[i];

index\_of\_max = i;

}

}

return index\_of\_max;

}

qreal calc::get\_width\_line\_atack(QVector<plane> cur\_planes)

{

QVector <qreal> temp;

foreach (auto one\_plane, cur\_planes)

{

foreach (auto p, cur\_planes)

{

temp.append(get\_range(one\_plane,p));

}

}

qreal sum = 0;

temp = remove\_duplicates(temp);

foreach (auto val, temp)

{

sum+=val;

qDebug()<<"Расстояние между самолетами"<<val;

}

qDebug()<<"Ширина полосы налета"<<sum/cur\_planes.size();

return sum/temp.size();

}

QVector<qreal> calc::remove\_duplicates(QVector<qreal> vec)

{

QStringList vec\_str;

foreach (auto value, vec)

vec\_str.append(QString("%1").arg(value));

vec\_str.removeDuplicates();

QVector <qreal> vec\_for\_return;

foreach (auto string, vec\_str)

{

if (string.toDouble()>1)

vec\_for\_return.append(string.toDouble());

}

return vec\_for\_return;

}

qreal calc::get\_bearning(qreal lon\_start, qreal lat\_start, qreal lon\_finish, qreal lat\_finish)

{

qreal y = qSin(lon\_finish-lon\_start) \* qCos(lat\_finish);

qreal x = qCos(lat\_start)\*qSin(lat\_finish) -

qSin(lat\_start)\*qCos(lat\_finish)\*qCos(lon\_finish-lon\_start);

qreal bearning = qAtan2(y, x);

return bearning;

//return bearning\*180/3.14159265;

}

qreal calc::get\_bearning\_for\_planes(qreal lon\_start, qreal lat\_start, qreal lon\_finish, qreal lat\_finish)

{

qreal y = qSin(lon\_finish-lon\_start) \* qCos(lat\_finish);

qreal x = qCos(lat\_start)\*qSin(lat\_finish) -

qSin(lat\_start)\*qCos(lat\_finish)\*qCos(lon\_finish-lon\_start);

qreal bearning = qAtan2(y, x)\*180/3.14159265;

if (bearning<0) return 180+(180+bearning);

else return bearning;

}

QVector<std::pair<qreal, qreal> > calc::get\_line\_atack\_coordinates(qreal bearning, qreal width, qreal lon\_start, qreal lat\_start, qreal lon\_finish, qreal lat\_finish)

{

QVector <std::pair<qreal,qreal>> for\_return;

for\_return.append(get\_one\_coor\_for\_line(bearning+1.570796327,lon\_start,lat\_start,width));

for\_return.append(get\_one\_coor\_for\_line(bearning-1.570796327,lon\_start,lat\_start,width));

for\_return.append(get\_one\_coor\_for\_line(bearning+1.570796327,lon\_finish,lat\_finish,width));

for\_return.append(get\_one\_coor\_for\_line(bearning-1.570796327,lon\_finish,lat\_finish,width));

return for\_return;

}

std::pair<qreal, qreal> calc::get\_one\_coor\_for\_line(qreal bearning, qreal lon, qreal lat, qreal width)

{

qreal radius = 6371;

qreal lat\_finish = qAsin( qSin(lat)\*qCos(width/radius) +

qCos(lat)\*qSin(width/radius)\*qCos(bearning) );

qreal lon\_finish = lon + qAtan2(qSin(bearning)\*qSin(width/radius)\*qCos(lat),

qCos(width/radius)-qSin(lat)\*qSin(lat\_finish));

std::pair<qreal,qreal> coor;

coor.first = lon\_finish;

coor.second = lat\_finish;

return coor;

}

QVector<plane> calc::update\_all\_plane\_course(QVector<plane> cur\_planes, qreal lon\_finish, qreal lat\_finish)

{

for (int i=0;i<cur\_planes.size();i++)

{

cur\_planes[i].course = get\_bearning\_for\_planes(cur\_planes[i].lon,cur\_planes[i].lat,lon\_finish,lat\_finish);

qDebug()<<"Новый курс самолета"<<cur\_planes[i].course;

}

return cur\_planes;

}

qreal calc::get\_range(plane p, airbase a)

{

qreal rad90 = 1.570796327;

qreal Earth\_rad = 6371;

qreal range = Earth\_rad\*(qAcos(qCos((rad90-p.lat))\*qCos((rad90-a.lat))+qSin((rad90-p.lat))\*qSin((rad90-a.lat))\*qCos((a.lon-p.lon))));

return range;

}

qreal calc::get\_range(plane p, plane p\_sec)

{

qreal rad90 = 1.570796327;

qreal Earth\_rad = 6371;

qreal range = Earth\_rad\*(qAcos(qCos((rad90-p.lat))\*qCos((rad90-p\_sec.lat))+qSin((rad90-p.lat))\*qSin((rad90-p\_sec.lat))\*qCos((p\_sec.lon-p.lon))));

return range;

}

qreal calc::get\_range(plane p, qreal lon, qreal lat)

{

qreal rad90 = 1.570796327;

qreal Earth\_rad = 6371;

qreal range = Earth\_rad\*(qAcos(qCos((rad90-p.lat))\*qCos((rad90-lat))+qSin((rad90-p.lat))\*qSin((rad90-lat))\*qCos((lon-p.lon))));

return range;

}

qreal calc::get\_range(airbase a, qreal lon, qreal lat)

{

qreal rad90 = 1.570796327;

qreal Earth\_rad = 6371;

qreal range = Earth\_rad\*(qAcos(qCos((rad90-a.lat))\*qCos((rad90-lat))+qSin((rad90-a.lat))\*qSin((rad90-lat))\*qCos((lon-a.lon))));

return range;

}

qreal calc::get\_hit\_on\_radius(qreal plane\_point\_range, qreal airbase\_point\_range, qreal plane\_airbase\_range, qreal radius)

{

qreal alfa = (qAcos((pow(airbase\_point\_range,2)+pow(plane\_point\_range,2)-pow(plane\_airbase\_range,2))/(2\*airbase\_point\_range\*plane\_point\_range)));

qreal gamma = qAsin((airbase\_point\_range\*qSin(alfa))/(radius));

qreal beta = (3.14159265)-alfa-gamma;

qreal d\_fake = (radius\*qSin(beta))/(qSin(alfa));

return (plane\_point\_range - d\_fake);

}

plane calc::create\_current\_plane(plane data, QString name, int alt\_type, int sfer\_type, qreal V, qreal S, qreal D)

{

plane p;

p.name = name;

p.tsum = get\_tsum(data,alt\_type,sfer\_type);

p.Ssum = get\_ssum(data,alt\_type,sfer\_type);

p.Vc = V;

p.Sish = S;

p.Dk = get\_D(sfer\_type,D);

p.alt\_type = alt\_type;

p.sfer\_type = sfer\_type;

p.Vgp = data.Vgp;

p.lon = 0;

p.lat = 0;

p.course = 0;

p.Svrvb = get\_Svrvb(p);

p.color = get\_random\_color();

p.main\_plane = false;

p.hit = false;

return p;

}

qreal calc::get\_tsum(plane data, int alt\_type, int sfer\_type)

{

if (sfer\_type==0)

{

switch (alt\_type) {

case 0:

return data.tpass+data.tn+data.tr+data.t180+data.tvm;

break;

case 1:

return data.tpass+data.tn+data.t180;

break;

case 2:

return data.tpass+data.tn+data.tvm+data.t180;

break;

default:

break;

}

}

else

{

switch (alt\_type) {

case 0:

return data.tpass+data.tn+data.tr+data.tvm;

break;

case 1:

return data.tpass+data.tn;

break;

case 2:

return data.tpass+data.tn+data.tvm;

break;

default:

break;

}

}

}

qreal calc::get\_ssum(plane data, int alt\_type, int sfer\_type)

{

if (sfer\_type==0)

{

switch (alt\_type) {

case 0:

return data.Sn+data.Sr-data.Lsm-data.Svm;

break;

case 1:

return data.Sn;

break;

case 2:

return data.Sn+data.Svm;

break;

default:

break;

}

}

else

{

switch (alt\_type) {

case 0:

return data.Sn+data.Sr+data.Svm;

break;

case 1:

return data.Sn;

break;

case 2:

return data.Sn+data.Svm;

break;

default:

break;

}

}

}

qreal calc::get\_D(int sfer\_type, qreal D)

{

switch (sfer\_type) {

case 0:

return -D;

break;

case 1:

return D;

break;

default:

break;

}

}

qreal calc::get\_Svrvb(plane p)

{

qreal Svrvb = (p.Sish-p.Vc/100\*p.tsum+(p.Vc/p.Vgp)\*(p.Ssum+p.Dk))/((p.Vc/p.Vgp)+1);

qDebug()<<"Sврвб"<<Svrvb<<"Sб"<<Svrvb-p.Ssum+p.Dk;

if ((Svrvb-p.Ssum+p.Dk)>0)

{

qDebug()<<"Расчет по первой формуле";

return Svrvb;

}

else

{

qDebug()<<"Расчет по второй формуле";

return p.Sish-p.Vc/100\*p.tsum;

}

}

QColor calc::get\_random\_color()

{

int r = Random::get(0,255);

int g = Random::get(0,255);

int b = Random::get(0,255);

return QColor(r,g,b);

}

**calc.h**

#ifndef CALC\_H

#define CALC\_H

#include <QString>

#include <plane.h>

#include <airbase.h>

#include <QDebug>

#include "random.h"

#include <QColor>

#include <QtMath>

#include <includes.h>

class calc

{

public:

calc();

QVector <plane> count\_all\_ranges\_plane\_airbase(QVector <plane> cur\_planes,QVector <airbase> cur\_airbases);

QVector <plane> get\_main\_plane(QVector <plane> cur\_planes);

int get\_index\_of\_max(QVector <qreal> vec);

qreal get\_width\_line\_atack(QVector <plane> cur\_planes);

QVector <qreal> remove\_duplicates(QVector <qreal> vec);

qreal get\_bearning(qreal lon\_start, qreal lat\_start, qreal lon\_finish, qreal lat\_finish);

qreal get\_bearning\_for\_planes(qreal lon\_start, qreal lat\_start, qreal lon\_finish, qreal lat\_finish);

QVector <std::pair<qreal,qreal>> get\_line\_atack\_coordinates(qreal bearning, qreal width, qreal lon\_start,

qreal lat\_start,

qreal lon\_finish,

qreal lat\_finish);

std::pair<qreal,qreal> get\_one\_coor\_for\_line(qreal bearning, qreal lon, qreal lat, qreal width);

QVector <plane> update\_all\_plane\_course(QVector <plane> cur\_planes, qreal lon\_finish, qreal lat\_finish);

qreal get\_range(plane p, airbase a);

qreal get\_range(plane p, plane p\_sec);

qreal get\_range(plane p, qreal lon, qreal lat);

qreal get\_range(airbase a, qreal lon, qreal lat);

qreal get\_hit\_on\_radius(qreal plane\_point\_range, qreal airbase\_point\_range, qreal plane\_airbase\_range, qreal radius);

plane create\_current\_plane(plane data, QString name, int alt\_type, int sfer\_type, qreal V, qreal S, qreal D);

qreal get\_tsum(plane data,int alt\_type, int sfer\_type);

qreal get\_ssum(plane data,int alt\_type, int sfer\_type);

qreal get\_D(int sfer\_type, qreal D);

qreal get\_Svrvb(plane p);

QColor get\_random\_color();

};

#endif // CALC\_H

**datawindow.cpp**

#include "datawindow.h"

#include "ui\_datawindow.h"

DataWindow::DataWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::DataWindow)

{

ui->setupUi(this);

//buttons

create\_button\_add();

create\_button\_remove();

//tableView

open\_model("planes", ui->tableView);

}

DataWindow::~DataWindow()

{

delete ui;

}

void DataWindow::create\_button\_add()

{

AnimatedLabel\* addButton = new AnimatedLabel(this,":/new/prefix1/icons/edit.gif","Нажмите для создания типового варианта самолета",64,64);

ui->lay\_but\_add->addWidget(addButton);

connect(addButton,SIGNAL(clicked()),SLOT(on\_pushButton\_add()));

}

void DataWindow::create\_button\_remove()

{

AnimatedLabel\* removeButton = new AnimatedLabel(this,":/new/prefix1/icons/remove.gif","Нажмите для создания удаления типового варианта самолета",64,64);

ui->lay\_but\_remove->addWidget(removeButton);

connect(removeButton,SIGNAL(clicked()),SLOT(on\_pushButton\_remove()));

}

void DataWindow::open\_DB()

{

db = QSqlDatabase::addDatabase("QSQLITE");

QString path = qApp->applicationDirPath();

qDebug()<<path;

path.push\_back("/database.db");

db.setDatabaseName(path);

if (db.open()) qDebug("DB open");

else qDebug("DB not open");

}

void DataWindow::open\_model(QString table\_name, QTableView \*&tableView)

{

open\_DB();

model = new QSqlTableModel(this,db);

model->setTable(table\_name);

model->select();

tableView->setModel(model);

QStringList horiz\_header;

horiz\_header<<"Наименование"

<<"tпасс, мин"

<<"tн, мин"

<<"tр, мин"

<<"t180, мин"

<<"tвм, мин"

<<"Sн, км"

<<"Sр, км"

<<"Lсм"

<<"Sвм, км"

<<"Vц, км/ч"

<<"Sисх, км"

<<"Dk, км"

<<"Vгп, км/ч";

int i=0;

foreach (QString str, horiz\_header)

{

model->setHeaderData(i,Qt::Horizontal,str);

i++;

}

correct\_visual(tableView);

}

void DataWindow::correct\_visual(QTableView \*&tableView)

{

tableView->horizontalHeader()->resizeContentsPrecision();

tableView->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);

tableView->verticalHeader()->resizeContentsPrecision();

tableView->verticalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableView->horizontalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableView->resizeColumnsToContents();

tableView->resizeRowsToContents();

}

void DataWindow::closeEvent(QCloseEvent \*event)

{

emit update\_window();

}

void DataWindow::on\_pushButton\_add()

{

model->insertRow(model->rowCount());

}

void DataWindow::on\_pushButton\_remove()

{

if (row\_to\_remove==-1)

{

QMessageBox::critical(this,"Ошибка","Выберите строку для удаления");

}

else

{

QMessageBox \*msgBox = new QMessageBox(QMessageBox::Warning,"Внимание",QString("Вы действительно хотите удалить: %1").arg(model->data(model->index(row\_to\_remove,0),Qt::DisplayRole).toString()),

QMessageBox::Yes| QMessageBox::No);

if(msgBox->exec() == QMessageBox::Yes)

{

model->removeRow(row\_to\_remove);

}

delete msgBox;

}

}

void DataWindow::on\_tableView\_clicked(const QModelIndex &index)

{

row\_to\_remove=index.row();

}

**datawindow.h**

#ifndef DATAWINDOW\_H

#define DATAWINDOW\_H

#include <QMainWindow>

#include <includes.h>

#include <animatedlabel.h>

#include <QTableView>

namespace Ui {

class DataWindow;

}

class DataWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit DataWindow(QWidget \*parent = nullptr);

~DataWindow();

private:

Ui::DataWindow \*ui;

int row\_to\_remove = -1;

//buttons

void create\_button\_add();

void create\_button\_remove();

//Db

QSqlDatabase db;

QSqlTableModel \*model;

void open\_DB();

void open\_model(QString table\_name, QTableView \*&tableView);

//visual

void correct\_visual(QTableView \*&tableView);

//close

void closeEvent(QCloseEvent\* event);

private slots:

void on\_pushButton\_add();

void on\_pushButton\_remove();

void on\_tableView\_clicked(const QModelIndex &index);

signals:

void update\_window();

};

#endif // DATAWINDOW\_H

**datawindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>DataWindow</class>

<widget class="QMainWindow" name="DataWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>Редактирование базы данных</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QTableView" name="tableView"/>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<layout class="QGridLayout" name="lay\_but\_add"/>

</item>

<item>

<layout class="QGridLayout" name="lay\_but\_remove"/>

</item>

</layout>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

**equipwindow.cpp**

#include "equipwindow.h"

#include "ui\_equipwindow.h"

EquipWindow::EquipWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::EquipWindow)

{

ui->setupUi(this);

//Db

open\_DB();

//buttons

create\_button\_data();

create\_button\_add();

//table

create\_table\_current(ui->tableWidget\_current);

create\_table\_airbase(ui->tableWidget\_airbase);

//listWidget data

add\_item\_in\_data\_listWidget(load\_DB("planes"),ui->listWidget\_data);

}

EquipWindow::~EquipWindow()

{

delete ui;

}

QVector<airbase> EquipWindow::create\_current\_airbases(QTableWidget \*&tableWidget, QVector<plane> cur\_planes)

{

QVector <airbase> cur\_air;

if (tableWidget->rowCount()==0) return cur\_air;

else

{

for (int i=0;i<tableWidget->rowCount();i++)

{

airbase a;

a.name = tableWidget->item(i,0)->text();

a.lon = 0;

a.lat = 0;

a.course = 0;

foreach (auto S, cur\_planes)

{

a.Svrvb\_vec.append(S.Svrvb);

}

cur\_air.append(a);

}

return cur\_air;

}

}

void EquipWindow::create\_button\_data()

{

AnimatedLabel\* dataButton = new AnimatedLabel(this,":/new/prefix1/icons/database.gif","Нажмите для редактирования базы данных типовых самолетов",64,64);

ui->toolBar->addWidget(dataButton);

connect(dataButton,SIGNAL(clicked()),SLOT(on\_pushButton\_data()));

}

void EquipWindow::create\_button\_add()

{

AnimatedLabel\* addButton = new AnimatedLabel(this,":/new/prefix1/icons/add.gif","Нажмите для добавления самолета в текущее оснащение",128,128);

ui->lay\_but\_add->addWidget(addButton);

connect(addButton,SIGNAL(clicked()),SLOT(on\_pushButton\_add()));

}

void EquipWindow::add\_item\_in\_data\_listWidget(QVector<plane> data, QListWidget \*&listWidget)

{

listWidget->clear();

QStringList items;

foreach (auto item, data)

items.append(item.name);

listWidget->addItems(items);

}

void EquipWindow::insert\_data\_in\_parms\_on\_click(QString name, QVector<plane> data)

{

int index\_of\_cur\_plane = index\_of\_plane(name,data);

ui->doubleSpinBox\_V->setValue(data.value(index\_of\_cur\_plane).Vc);

ui->doubleSpinBox\_S->setValue(data.value(index\_of\_cur\_plane).Sish);

ui->doubleSpinBox\_D->setValue(data.value(index\_of\_cur\_plane).Dk);

}

void EquipWindow::create\_table\_current(QTableWidget \*&tableWidget)

{

QStringList horiz\_header;

horiz\_header<<"Наименование"

<<"Высота\nатаки"

<<"Атака в"

<<"Vц, км/ч"

<<"Sисх, км"

<<"tΣ, мин"

<<"SΣ, км"

<<"Dk, км"

<<"Sврвб, км";

tableWidget->setRowCount(0);

tableWidget->setColumnCount(horiz\_header.size());

tableWidget->setHorizontalHeaderLabels(horiz\_header);

correct\_visual(tableWidget);

}

void EquipWindow::correct\_visual(QTableWidget \*&tableWidget)

{

tableWidget->horizontalHeader()->resizeContentsPrecision();

tableWidget->horizontalHeader()->setSectionResizeMode(0,QHeaderView::Stretch);

tableWidget->horizontalHeader()->setSectionResizeMode(1,QHeaderView::Stretch);

tableWidget->horizontalHeader()->setSectionResizeMode(2,QHeaderView::Stretch);

tableWidget->verticalHeader()->resizeContentsPrecision();

tableWidget->verticalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->horizontalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->resizeColumnsToContents();

tableWidget->resizeRowsToContents();

}

void EquipWindow::update\_table(QTableWidget \*&tableWidget, QVector<plane> data)

{

tableWidget->setRowCount(data.size());

int i=0;

foreach (auto p, data)

{

tableWidget->setItem(i,0,new QTableWidgetItem(p.name));

tableWidget->setItem(i,1,new QTableWidgetItem(ui->comboBox\_alt->itemText(p.alt\_type)));

tableWidget->setItem(i,2,new QTableWidgetItem(ui->comboBox\_sfer->itemText(p.sfer\_type)));

tableWidget->setItem(i,3,new QTableWidgetItem(QString::number(p.Vc)));

tableWidget->setItem(i,4,new QTableWidgetItem(QString::number(p.Sish)));

tableWidget->setItem(i,5,new QTableWidgetItem(QString::number(p.tsum)));

tableWidget->setItem(i,6,new QTableWidgetItem(QString::number(p.Ssum)));

tableWidget->setItem(i,7,new QTableWidgetItem(QString::number(p.Dk)));

tableWidget->setItem(i,8,new QTableWidgetItem(QString::number(p.Svrvb)));

i++;

}

}

void EquipWindow::create\_table\_airbase(QTableWidget \*&tableWidget)

{

QStringList horiz\_header;

horiz\_header<<"Наименоваине\nаэродрома";

tableWidget->setRowCount(0);

tableWidget->setColumnCount(horiz\_header.size());

tableWidget->setHorizontalHeaderLabels(horiz\_header);

tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

}

QVector<plane> EquipWindow::load\_DB(QString table\_name)

{

QVector <plane> data;

query = new QSqlQuery(db);

query->exec(QString("SELECT \* FROM %1;").arg(table\_name));

while (query->next())

{

plane p;

p.name = query->value(0).toString();

p.tpass = query->value(1).toDouble();

p.tn = query->value(2).toDouble();

p.tr = query->value(3).toDouble();

p.t180 = query->value(4).toDouble();

p.tvm = query->value(5).toDouble();

p.Sn = query->value(6).toDouble();

p.Sr = query->value(7).toDouble();

p.Lsm = query->value(8).toDouble();

p.Svm = query->value(9).toDouble();

p.Vc = query->value(10).toDouble();

p.Sish = query->value(11).toDouble();

p.Dk = query->value(12).toDouble();

p.Vgp = query->value(13).toDouble();

data.push\_back(p);

}

return data;

}

void EquipWindow::open\_DB()

{

db = QSqlDatabase::addDatabase("QSQLITE");

QString path = qApp->applicationDirPath();

path.push\_back("/database.db");

db.setDatabaseName(path);

if (db.open()) qDebug("DB open");

else qDebug("DB not open");

}

int EquipWindow::index\_of\_plane(QString name, QVector<plane> data)

{

for (int i=0;i<data.size();i++)

if (data.value(i).name == name)

return i;

}

void EquipWindow::closeEvent(QCloseEvent \*event)

{

current\_airbases = create\_current\_airbases(ui->tableWidget\_airbase,current\_planes);

emit send\_data(current\_planes, current\_airbases);

}

void EquipWindow::update\_slot()

{

qDebug()<<"Слот обновления";

add\_item\_in\_data\_listWidget(load\_DB("planes"),ui->listWidget\_data);

}

void EquipWindow::on\_pushButton\_data()

{

datawin = new DataWindow(this);

datawin->setWindowModality(Qt::WindowModal);

connect(datawin,SIGNAL(update\_window()),this,SLOT(update\_slot()));

datawin->showMaximized();

}

void EquipWindow::on\_pushButton\_add()

{

if (currentName.isEmpty())

QMessageBox::critical(this,"Ошибка","Выберите один из типовых вариантов оснащения!");

else

{

if ((ui->doubleSpinBox\_D->value()==0)||(ui->doubleSpinBox\_S->value()==0)||(ui->doubleSpinBox\_V->value()==0))

QMessageBox::critical(this,"Ошибка","Нулевые параметры не возможны!");

else

{

current\_planes.append(c.create\_current\_plane(load\_DB("planes").value(index\_of\_plane(currentName,load\_DB("planes"))),

currentName,

ui->comboBox\_alt->currentIndex(),

ui->comboBox\_sfer->currentIndex(),

ui->doubleSpinBox\_V->value(),

ui->doubleSpinBox\_S->value(),

ui->doubleSpinBox\_D->value()));

update\_table(ui->tableWidget\_current,current\_planes);

}

}

}

void EquipWindow::on\_tableWidget\_current\_cellDoubleClicked(int row, int column)

{

ui->tableWidget\_current->removeRow(row);

current\_planes.remove(row);

update\_table(ui->tableWidget\_current,current\_planes);

}

void EquipWindow::on\_tableWidget\_current\_cellClicked(int row, int column)

{

ui->tableWidget\_current->selectRow(row);

update\_table(ui->tableWidget\_current,current\_planes);

}

void EquipWindow::on\_listWidget\_data\_currentTextChanged(const QString &currentText)

{

currentName = currentText;

insert\_data\_in\_parms\_on\_click(currentName,load\_DB("planes"));

}

void EquipWindow::on\_spinBox\_airbase\_valueChanged(int arg1)

{

ui->tableWidget\_airbase->setRowCount(arg1);

for (int i=0;i<ui->tableWidget\_airbase->rowCount();i++)

{

qDebug()<<i;

ui->tableWidget\_airbase->setItem(i,0,new QTableWidgetItem(QString("Аэродром №%1").arg(i+1)));

}

}

**equipwindow.h**

#ifndef EQUIPWINDOW\_H

#define EQUIPWINDOW\_H

#include <QMainWindow>

#include <includes.h>

#include <animatedlabel.h>

#include <QListWidgetItem>

#include <QTableWidget>

#include "datawindow.h"

#include "plane.h"

#include "calc.h"

#include "airbase.h"

namespace Ui {

class EquipWindow;

}

class EquipWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit EquipWindow(QWidget \*parent = nullptr);

~EquipWindow();

private:

Ui::EquipWindow \*ui;

DataWindow\* datawin;

QString currentName;

calc c;

//data

QVector <plane> data\_of\_plane;

QVector <plane> current\_planes;

QVector <airbase> current\_airbases;

QVector <airbase> create\_current\_airbases(QTableWidget \*&tableWidget, QVector <plane> cur\_planes);

//buttons

void create\_button\_data();

void create\_button\_add();

//listWidget

void add\_item\_in\_data\_listWidget(QVector <plane> data, QListWidget \*&listWidget);

void insert\_data\_in\_parms\_on\_click(QString name, QVector <plane> data);

//table

void create\_table\_current(QTableWidget \*&tableWidget);

void correct\_visual(QTableWidget \*&tableWidget);

void update\_table(QTableWidget \*&tableWidget, QVector<plane>data);

void create\_table\_airbase(QTableWidget \*&tableWidget);

//Db

QSqlDatabase db;

QSqlQuery \*query;

QVector <plane> load\_DB(QString table\_name);

void open\_DB();

int index\_of\_plane(QString name, QVector <plane> data);

//close

void closeEvent(QCloseEvent\* event);

private slots:

void update\_slot();

void on\_pushButton\_data();

void on\_pushButton\_add();

void on\_tableWidget\_current\_cellDoubleClicked(int row, int column);

void on\_tableWidget\_current\_cellClicked(int row, int column);

void on\_listWidget\_data\_currentTextChanged(const QString &currentText);

void on\_spinBox\_airbase\_valueChanged(int arg1);

signals:

void send\_data(QVector <plane> current\_planes, QVector <airbase> current\_airbases);

};

#endif // EQUIPWINDOW\_H

**equipwindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>EquipWindow</class>

<widget class="QMainWindow" name="EquipWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1309</width>

<height>634</height>

</rect>

</property>

<property name="windowTitle">

<string>Выбор оснащения</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QHBoxLayout" name="horizontalLayout\_2">

<item>

<widget class="QFrame" name="frame\_current">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Sunken</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_7">

<item>

<widget class="QGroupBox" name="groupBox">

<property name="title">

<string>Текущее оснащение противника</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QTableWidget" name="tableWidget\_current">

<property name="statusTip">

<string>Нажмите два раза для удаления из текущего оснащения противника</string>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QFrame" name="frame\_data">

<property name="maximumSize">

<size>

<width>400</width>

<height>16777215</height>

</size>

</property>

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Raised</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_6">

<item>

<widget class="QGroupBox" name="groupBox\_2">

<property name="title">

<string>Возможные варианты оснащения</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_5">

<item>

<widget class="QListWidget" name="listWidget\_data">

<property name="statusTip">

<string>Нажмите для выбора варианта оснащения</string>

</property>

</widget>

</item>

<item>

<widget class="QGroupBox" name="groupBox\_3">

<property name="title">

<string>Дополнительные параметры</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QGroupBox" name="groupBox\_4">

<property name="title">

<string>Высота атаки</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QComboBox" name="comboBox\_alt">

<property name="minimumSize">

<size>

<width>0</width>

<height>0</height>

</size>

</property>

<item>

<property name="text">

<string>Стратосфера</string>

</property>

</item>

<item>

<property name="text">

<string>Большая и средняя, без вертикального маневра</string>

</property>

</item>

<item>

<property name="text">

<string>Большая и средняя, с вертикальным маневром</string>

</property>

</item>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QGroupBox" name="groupBox\_5">

<property name="title">

<string>Атака в</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_4">

<item>

<widget class="QComboBox" name="comboBox\_sfer">

<item>

<property name="text">

<string>Заднюю полусферу</string>

</property>

</item>

<item>

<property name="text">

<string>Переднюю полусферу</string>

</property>

</item>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<layout class="QGridLayout" name="gridLayout">

<item row="0" column="0">

<widget class="QLabel" name="label">

<property name="statusTip">

<string>Скорость цели</string>

</property>

<property name="text">

<string>Vц</string>

</property>

</widget>

</item>

<item row="0" column="1">

<widget class="QDoubleSpinBox" name="doubleSpinBox\_V">

<property name="statusTip">

<string>Скорость цели</string>

</property>

<property name="maximum">

<double>100000000000000000000.000000000000000</double>

</property>

</widget>

</item>

<item row="1" column="0">

<widget class="QLabel" name="label\_2">

<property name="statusTip">

<string>исходный рубеж</string>

</property>

<property name="text">

<string>Sисх</string>

</property>

</widget>

</item>

<item row="1" column="1">

<widget class="QDoubleSpinBox" name="doubleSpinBox\_S">

<property name="statusTip">

<string>исходный рубеж</string>

</property>

<property name="maximum">

<double>100000000000000000000.000000000000000</double>

</property>

</widget>

</item>

<item row="2" column="0">

<widget class="QLabel" name="label\_3">

<property name="statusTip">

<string>В момент ввода в бой необходимо, чтобы цель находилась впереди истребителя на расстоянии Dk</string>

</property>

<property name="text">

<string>Dk</string>

</property>

</widget>

</item>

<item row="2" column="1">

<widget class="QDoubleSpinBox" name="doubleSpinBox\_D">

<property name="statusTip">

<string>В момент ввода в бой необходимо, чтобы цель находилась впереди истребителя на расстоянии Dk</string>

</property>

<property name="maximum">

<double>100000000000000000000.000000000000000</double>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</widget>

</item>

<item>

<layout class="QGridLayout" name="lay\_but\_add"/>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QFrame" name="frame\_airbase">

<property name="maximumSize">

<size>

<width>250</width>

<height>16777215</height>

</size>

</property>

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Raised</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_9">

<item>

<widget class="QGroupBox" name="groupBox\_6">

<property name="title">

<string>Свои войска</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_8">

<item>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QLabel" name="label\_4">

<property name="text">

<string>Количество аэродромов</string>

</property>

</widget>

</item>

<item>

<widget class="QSpinBox" name="spinBox\_airbase"/>

</item>

</layout>

</item>

<item>

<widget class="QTableWidget" name="tableWidget\_airbase">

<property name="statusTip">

<string>Нажмите два раза для удаления из текущего оснащения противника</string>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1309</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

<widget class="QToolBar" name="toolBar">

<property name="windowTitle">

<string>toolBar</string>

</property>

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

</widget>

<resources/>

<connections/>

</ui>

**includes.h**

#ifndef INCLUDES\_H

#define INCLUDES\_H

#include <QMovie>

#include <QLabel>

#include <QMessageBox>

#include <QDebug>

#include <QSqlDatabase>

#include <QSqlTableModel>

#include <QSqlQuery>

#include <QVector>

#include <QDesktopWidget>

#include <MarbleWidget.h>

#include <MarbleModel.h>

#include <GeoDataTreeModel.h>

#include <GeoDataPlacemark.h>

#include <GeoDataCoordinates.h>

#include <GeoDataStyle.h>

#include <GeoDataIconStyle.h>

#include <MarbleWidgetPopupMenu.h>

#include <ViewportParams.h>

#include <GeoDataRegion.h>

#include <GeoDataLineString.h>

#include <MarbleWidgetInputHandler.h>

#include <GeoDataLatLonBox.h>

#include <GeoDataLatLonAltBox.h>

#include <GeoDataLinearRing.h>

#include <AbstractFloatItem.h>

#include <GeoPainter.h>

#include <QContextMenuEvent>

#include <GeoDataDocument.h>

#include <QPaintEvent>

#include <QtMath>

#include <QColorDialog>

#include <QGridLayout>

#include <LayerInterface.h>

#include <GeoDataPolygon.h>

#include <QtXml/QtXml>

#include <QSvgRenderer>

#include <MarbleGlobal.h>

#include <QImage>

#include <QColor>

#include <MarbleWidget.h>

#include <MarbleMap.h>

#include <MarbleModel.h>

#include <GeoPainter.h>

#include <LayerInterface.h>

#include <QPainter>

#include <QComboBox>

#endif // INCLUDES\_H

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QFile>

#include <QTranslator>

#include <QLibraryInfo>

#include <QDebug>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

//QSS

QFile styleSheetFile(":/new/prefix1/HackBook.qss");

styleSheetFile.open(QFile::ReadOnly);

QString styleSheet = QLatin1String(styleSheetFile.readAll());

a.setStyleSheet(styleSheet);

//Localization

QTranslator qtTranslator;

qtTranslator.load("qt\_" + QLocale::system().name(),

QLibraryInfo::location(QLibraryInfo::TranslationsPath));

a.installTranslator(&qtTranslator);

MainWindow w;

w.showMaximized();

return a.exec();

}

**mainico.rc**

IDI\_ICON1 ICON DISCARDABLE "icon.ico"

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

//visual on start

ui->frame\_table->hide();

ui->widget\_legend->xAxis->setVisible(false);

ui->widget\_legend->yAxis->setVisible(false);

ui->widget\_legend->hide();

//buttons

create\_button\_equip();

create\_button\_result();

create\_button\_settings();

create\_button\_line();

//map

load\_map(ui->map\_lay,"earth/plain/plain.dgml",0);

//Db

open\_DB();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::show\_result\_window()

{

resultwin = new ResultWindow(this,current\_planes,current\_airbases);

resultwin->setWindowModality(Qt::WindowModal);

resultwin->show();

}

void MainWindow::create\_button\_equip()

{

AnimatedLabel\* equipmentButton = new AnimatedLabel(this,":/new/prefix1/icons/equip.gif","Нажмите для создания оснащения",64,64);

ui->mainToolBar->addWidget(equipmentButton);

connect(equipmentButton,SIGNAL(clicked()),SLOT(on\_pushButton\_equipment()));

}

void MainWindow::create\_button\_settings()

{

AnimatedLabel\* settingsButton = new AnimatedLabel(this,":/new/prefix1/icons/settings.gif","Нажмите управления настройками карты",64,64);

ui->mainToolBar->addWidget(settingsButton);

connect(settingsButton,SIGNAL(clicked()),SLOT(on\_pushButton\_settings()));

}

void MainWindow::create\_button\_line()

{

AnimatedLabel\* lineButton = new AnimatedLabel(this,":/new/prefix1/icons/line.gif","Нажмите для задания полосы налета",64,64);

ui->lay\_but\_line->addWidget(lineButton);

connect(lineButton,SIGNAL(clicked()),SLOT(on\_pushButton\_line()));

}

void MainWindow::create\_button\_result()

{

AnimatedLabel\* resultButton = new AnimatedLabel(this,":/new/prefix1/icons/result.gif","Нажмите для показа резульатов моделирования",64,64);

ui->mainToolBar->addWidget(resultButton);

connect(resultButton,SIGNAL(clicked()),SLOT(on\_pushButton\_result()));

}

void MainWindow::closeEvent(QCloseEvent \*event)

{

db.close();

delete map;

delete this;

}

void MainWindow::load\_map(QGridLayout \*&lay, QString map\_theme, int projection)

{

map=new MarbleWidget();

map->setMapThemeId(map\_theme);

map->setShowCrosshairs(false);

foreach(AbstractFloatItem \*floatItem, map->floatItems())

floatItem->hide();

foreach(RenderPlugin\* plugin, map->renderPlugins())

if(plugin->nameId() == "stars") plugin->setEnabled(false);

map->setShowCities(true);

layer = new TestLayer();

map->addLayer(layer);

lay->addWidget(map);

GeoDataCoordinates Moscow(37.61/57.3,55.75/57.3,GeoDataCoordinates::Radian);

map->centerOn(Moscow);

map->setProjection(projection);

connect(map,SIGNAL(mouseMoveGeoPosition(QString)),SLOT(mouse\_move\_on\_map(QString)));

connect(map,SIGNAL(zoomChanged(int)),this,SLOT(zoom\_map(int)));

map->zoomView(1500);

layer->maximum\_zoom=map->maximumZoom();

layer->minimum\_zoom=map->minimumZoom();

layer->zoom\_vector=layer->get\_zoom\_vector(layer->maximum\_zoom,layer->minimum\_zoom);

map->update();

}

void MainWindow::send\_data\_in\_layer(QVector<plane> cur\_p, QVector<airbase> cur\_a, QVector<GeoDataCoordinates> line\_atack)

{

layer->cur\_planes=cur\_p;

layer->cur\_airbases=cur\_a;

layer->l\_atack = line\_atack;

map->update();

}

void MainWindow::get\_line\_atack()

{

qreal bearning = c.get\_bearning(line\_atack.first().longitude(),line\_atack.first().latitude(),line\_atack.last().longitude(),line\_atack.last().latitude());

line\_atack = convert\_pair\_to\_coor(c.get\_line\_atack\_coordinates(bearning,

c.get\_width\_line\_atack(current\_planes),

line\_atack.first().longitude(),

line\_atack.first().latitude(),

line\_atack.last().longitude(),

line\_atack.last().latitude()));

send\_data\_in\_layer(current\_planes,current\_airbases,line\_atack);

}

void MainWindow::w\_calc()

{

if (line\_atack.size()==4)

{

for (int i=0;i<current\_planes.size();i++)

{

current\_planes[i].hit = false;

current\_planes[i].plane\_point\_range = c.get\_range(current\_planes[i],atack\_point.longitude(),atack\_point.latitude());

}

for (int i=0;i<current\_airbases.size();i++)

current\_airbases[i].airbase\_point\_range = c.get\_range(current\_airbases[i],atack\_point.longitude(),atack\_point.latitude());

for (int i=0;i<current\_planes.size();i++)

{

current\_planes[i].airbase\_hit\_index.clear();

for (int j=0;j<current\_airbases.size();j++)

{

qreal range\_plane\_airbase = c.get\_range(current\_planes[i],current\_airbases[j]);

if (range\_plane\_airbase<current\_planes[i].plane\_point\_range)

{

qreal range\_hit = c.get\_hit\_on\_radius(current\_planes[i].plane\_point\_range,current\_airbases[j].airbase\_point\_range,range\_plane\_airbase,current\_airbases[j].Svrvb\_vec[i]);

if (range\_hit>0) //если пролетает через зону и долетел до объекта записываем эффективности поражения

{

qDebug()<<current\_airbases[j].name<<"пересекает траекторию"<<current\_planes[i].name<<"параметр = "<<range\_hit<<"индекс самолета"<<i+1;

current\_planes[i].airbase\_hit\_index.append(j);

current\_planes[i].hit = true;

}

}

}

}

send\_data\_in\_layer(current\_planes, current\_airbases,line\_atack);

show\_result\_window();

}

else

QMessageBox::critical(this,"Ошибка","Заданы не все исходные параметры");

}

void MainWindow::open\_DB()

{

db = QSqlDatabase::addDatabase("QSQLITE");

QString path = qApp->applicationDirPath();

qDebug()<<path;

path.push\_back("/database.db");

db.setDatabaseName(path);

if (db.open()) qDebug("DB open");

else qDebug("DB not open");

}

void MainWindow::create\_table\_position(QTableWidget \*&tableWidget, QVector<plane> current\_planes, QVector <airbase> current\_airbases)

{

create\_table\_interface(tableWidget);

int index = 0;

foreach (auto p, current\_planes)

{

tableWidget->setRowCount(tableWidget->rowCount()+1);

tableWidget->setItem(tableWidget->rowCount()-1,0,new QTableWidgetItem(p.name));

create\_spinBox\_course(index,tableWidget,"plane");

create\_button\_position(index,tableWidget,"plane");

index++;

}

index = 0;

foreach (auto a, current\_airbases)

{

tableWidget->setRowCount(tableWidget->rowCount()+1);

tableWidget->setItem(tableWidget->rowCount()-1,0,new QTableWidgetItem(a.name));

create\_spinBox\_course(index,tableWidget,"airbase");

create\_button\_position(index,tableWidget,"airbase");

index++;

}

corretc\_table\_position\_visual(tableWidget,ui->frame\_table);

}

void MainWindow::create\_spinBox\_course(int index, QTableWidget \*&tableWidget, QString type)

{

QDoubleSpinBox\* spinBox\_course = new QDoubleSpinBox();

spinBox\_course->setMinimum(0);

spinBox\_course->setMaximum(360);

spinBox\_course->setSingleStep(10);

tableWidget->setCellWidget(tableWidget->rowCount()-1,3,spinBox\_course);

tableWidget->cellWidget(tableWidget->rowCount()-1,3)->setObjectName(QString("%1\_%2").arg(type).arg(index));

connect(spinBox\_course,SIGNAL(valueChanged(double)),this,SLOT(on\_doubleSpinBox\_valueChanged(double)));

}

void MainWindow::create\_button\_position(int index, QTableWidget \*&tableWidget, QString type)

{

AnimatedLabel\* coor\_button = new AnimatedLabel(this,":/new/prefix1/icons/coor.gif","Нажмите для ввода координат на карте",47,47);

tableWidget->setCellWidget(tableWidget->rowCount()-1,4,coor\_button);

tableWidget->cellWidget(tableWidget->rowCount()-1,4)->setObjectName(QString("%1\_%2").arg(type).arg(index));

connect(coor\_button,SIGNAL(clicked()),this,SLOT(on\_pushButton\_coor()));

}

QVector<GeoDataCoordinates> MainWindow::convert\_pair\_to\_coor(QVector<std::pair<qreal, qreal> > vec\_of\_pair)

{

QVector<GeoDataCoordinates> vec\_for\_return;

foreach (auto p, vec\_of\_pair)

{

GeoDataCoordinates coor(p.first,p.second);

vec\_for\_return.append(coor);

}

return vec\_for\_return;

}

bool MainWindow::all\_planes\_have\_coor(QVector<plane> current\_planes)

{

foreach (auto p, current\_planes)

{

if ((p.lon==0)||(p.lat==0)) return false;

}

return true;

}

void MainWindow::draw\_legend(QCustomPlot \*&widget, QVector<plane> current\_planes)

{

widget->clearGraphs();

widget->show();

int height = 30;

widget->xAxis->setVisible(false);

widget->yAxis->setVisible(false);

int plane\_index = 1;

foreach (auto a, current\_planes)

{

widget->addGraph();

widget->graph()->setPen(QPen(a.color,50));

widget->graph()->setName(QString("%1").arg(plane\_index));

widget->replot();

plane\_index++;

height+=21;

}

widget->legend->setVisible(true);

widget->replot();

widget->setMaximumHeight(height+30);

}

void MainWindow::on\_pushButton\_coor()

{

if (sender()->objectName().split("\_")[0]=="plane")

{

current\_plane\_index = sender()->objectName().split("\_")[1].toInt();

connect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_plane(qreal,qreal,GeoDataCoordinates::Unit)));

}

else

{

current\_airbase\_index = sender()->objectName().split("\_")[1].toInt();

connect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_airbase(qreal,qreal,GeoDataCoordinates::Unit)));

}

}

void MainWindow::on\_doubleSpinBox\_valueChanged(double arg1)

{

if (sender()->objectName().split("\_")[0]=="plane")

current\_planes[sender()->objectName().split("\_")[1].toInt()].course=arg1;

else

current\_airbases[sender()->objectName().split("\_")[1].toInt()].course=arg1;

send\_data\_in\_layer(current\_planes, current\_airbases,line\_atack);

}

void MainWindow::get\_coordibates\_plane(qreal lon, qreal lat, GeoDataCoordinates::Unit)

{

ui->tableWidget\_position->setItem(current\_plane\_index,1,new QTableWidgetItem(QString::number(lon\*57.2958,'f',2)));

ui->tableWidget\_position->setItem(current\_plane\_index,2,new QTableWidgetItem(QString::number(lat\*57.2958,'f',2)));

current\_planes[current\_plane\_index].lon = lon;

current\_planes[current\_plane\_index].lat = lat;

send\_data\_in\_layer(current\_planes,current\_airbases,line\_atack);

disconnect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_plane(qreal,qreal,GeoDataCoordinates::Unit)));

}

void MainWindow::get\_coordibates\_airbase(qreal lon, qreal lat, GeoDataCoordinates::Unit)

{

ui->tableWidget\_position->setItem(current\_airbase\_index+current\_planes.size(),1,new QTableWidgetItem(QString::number(lon\*57.2958,'f',2)));

ui->tableWidget\_position->setItem(current\_airbase\_index+current\_planes.size(),2,new QTableWidgetItem(QString::number(lat\*57.2958,'f',2)));

current\_airbases[current\_airbase\_index].lon = lon;

current\_airbases[current\_airbase\_index].lat = lat;

send\_data\_in\_layer(current\_planes,current\_airbases,line\_atack);

disconnect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_airbase(qreal,qreal,GeoDataCoordinates::Unit)));

}

void MainWindow::get\_coordibates\_line\_atack(qreal lon, qreal lat, GeoDataCoordinates::Unit)

{

GeoDataCoordinates coor(lon,lat,GeoDataCoordinates::Radian);

line\_atack.append(coor);

if (line\_atack.size()==2)

{

atack\_point = line\_atack.last();

//Обновляем курс исходя из полосы атаки

current\_planes = c.update\_all\_plane\_course(current\_planes,line\_atack.last().longitude(),line\_atack.last().latitude());

get\_line\_atack();

w\_calc();

disconnect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_line\_atack(qreal,qreal,GeoDataCoordinates::Unit)));

}

}

void MainWindow::zoom\_map(int value)

{

layer->zoom\_value=value;

}

void MainWindow::get\_coor\_in\_string(QString string)

{

QString fake\_lon = string.split(", ")[0];

QString fake\_lat = string.split(", ")[1];

fake\_lon.replace("° ",".");

fake\_lat.replace("° ",".");

fake\_lon.replace("' ","");

fake\_lat.replace("' ","");

fake\_lon.replace(",","");

fake\_lat.replace(",","");

fake\_lon.replace("\"","");

fake\_lat.replace("\"","");

qreal lon,lat;

if (fake\_lon.indexOf("E")!=-1)

lon = fake\_lon.replace("E","").toDouble();

else

lon = -fake\_lon.replace("W","").toDouble();

if (fake\_lat.indexOf("N")!=-1)

lat = fake\_lat.replace("N","").toDouble();

else

lat = -fake\_lat.replace("S","").toDouble();

}

void MainWindow::mouse\_move\_on\_map(QString string)

{

if (string!="not available")

{

ui->statusbar->showMessage(QString("%1").arg(string));

get\_coor\_in\_string(string);

}

else ui->statusbar->clearMessage();

}

void MainWindow::create\_table\_interface(QTableWidget \*&tableWidget)

{

tableWidget->setRowCount(0);

QStringList horiz\_header;

horiz\_header<<"Наименование"<<"Широта"<<"Долгота"<<"Курс"<<"";

tableWidget->setColumnCount(horiz\_header.size());

tableWidget->setHorizontalHeaderLabels(horiz\_header);

}

void MainWindow::corretc\_table\_position\_visual(QTableWidget \*&tableWidget, QFrame \*&frame)

{

tableWidget->horizontalHeader()->resizeContentsPrecision();

tableWidget->verticalHeader()->resizeContentsPrecision();

tableWidget->verticalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->horizontalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->resizeColumnsToContents();

tableWidget->resizeRowsToContents();

tableWidget->setColumnWidth(4,47);

tableWidget->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

int table\_width = 0;

for (int i=0;i<tableWidget->columnCount();i++)

{

table\_width+=tableWidget->columnWidth(i);

}

frame->setMaximumWidth(table\_width+47+21);

}

void MainWindow::on\_pushButton\_equipment()

{

equipwin = new EquipWindow(this);

equipwin->setWindowModality(Qt::WindowModal);

connect(equipwin,SIGNAL(send\_data(QVector<plane>, QVector <airbase>)),this,SLOT(recive\_data(QVector<plane>, QVector <airbase>)));

equipwin->show();

}

void MainWindow::on\_pushButton\_settings()

{

settingswin = new SettingsWindow(this, map->mapThemeId(), map->projection());

settingswin->setWindowModality(Qt::WindowModal);

connect(settingswin,SIGNAL(send\_map\_settings(QString, int)),this,SLOT(set\_map\_settings(QString, int)));

settingswin->show();

}

void MainWindow::on\_pushButton\_result()

{

w\_calc();

}

void MainWindow::set\_map\_settings(QString map\_theme, int projection)

{

delete map;

load\_map(ui->map\_lay,map\_theme,projection);

send\_data\_in\_layer(current\_planes, current\_airbases,line\_atack);

}

void MainWindow::recive\_data(QVector<plane> cur\_planes, QVector<airbase> cur\_airbases)

{

if ((cur\_planes.size()==0)||(cur\_airbases.size()==0))

{

QMessageBox::warning(this,"Внимание","Вы не выбрали никакого оснащения");

}

else

{

ui->frame\_table->show();

line\_atack.clear();

layer->l\_atack.clear();

layer->cur\_planes.clear();

layer->cur\_airbases.clear();

current\_airbases = cur\_airbases;

current\_planes = cur\_planes;

create\_table\_position(ui->tableWidget\_position,current\_planes,current\_airbases);

draw\_legend(ui->widget\_legend,current\_planes);

}

}

void MainWindow::on\_pushButton\_line()

{

if (all\_planes\_have\_coor(current\_planes))

{

//current\_planes = c.count\_all\_ranges\_plane\_airbase(current\_planes,current\_airbases);

line\_atack.clear();

connect(map,SIGNAL(mouseClickGeoPosition(qreal,qreal,GeoDataCoordinates::Unit)),this,SLOT(get\_coordibates\_line\_atack(qreal,qreal,GeoDataCoordinates::Unit)));

}

else

QMessageBox::critical(this,"Ошибка","Сначала задайте координаты всех самолетов");

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <includes.h>

#include <animatedlabel.h>

#include <QDoubleSpinBox>

#include "maplayer.h"

#include "settingswindow.h"

#include "equipwindow.h"

#include "plane.h"

#include "airbase.h"

#include "calc.h"

#include "resultwindow.h"

#include "qcustomplot.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

SettingsWindow\* settingswin;

EquipWindow\* equipwin;

ResultWindow\* resultwin;

void show\_result\_window();

calc c;

//data

QVector <plane> current\_planes;

QVector <airbase> current\_airbases;

int current\_plane\_index;

int current\_airbase\_index;

QVector <GeoDataCoordinates> line\_atack;

GeoDataCoordinates atack\_point;

//buttons

void create\_button\_equip();

void create\_button\_settings();

void create\_button\_line();

void create\_button\_result();

//close

void closeEvent(QCloseEvent\* event);

//MAP

TestLayer\* layer;

MarbleWidget\* map;

void load\_map(QGridLayout \*&lay, QString map\_theme, int projection);

void send\_data\_in\_layer(QVector <plane> cur\_p, QVector <airbase> cur\_a,QVector <GeoDataCoordinates> line\_atack);

void get\_line\_atack();

void w\_calc();

//Db

QSqlDatabase db;

QSqlQuery \*query;

void open\_DB();

//table

void create\_table\_position(QTableWidget \*&tableWidget, QVector <plane> current\_planes, QVector <airbase> current\_airbases);

void create\_table\_interface(QTableWidget \*&tableWidget);

void corretc\_table\_position\_visual(QTableWidget \*&tableWidget, QFrame \*&frame);

//Dynamic controls

void create\_spinBox\_course(int index, QTableWidget \*&tableWidget, QString type);

void create\_button\_position(int index, QTableWidget \*&tableWidget, QString type);

QVector <GeoDataCoordinates> convert\_pair\_to\_coor(QVector<std::pair<qreal,qreal>> vec\_of\_pair);

//flags

bool all\_planes\_have\_coor(QVector <plane> current\_planes);

//legend

void draw\_legend(QCustomPlot \*&widget, QVector <plane> current\_planes);

private slots:

void on\_pushButton\_equipment();

void on\_pushButton\_settings();

void on\_pushButton\_result();

void set\_map\_settings(QString map\_theme, int projection);

void recive\_data(QVector <plane> cur\_planes, QVector <airbase> cur\_airbases);

void on\_pushButton\_coor();

void mouse\_move\_on\_map(QString string);

void on\_doubleSpinBox\_valueChanged(double arg1);

void get\_coordibates\_plane(qreal lon,qreal lat,GeoDataCoordinates::Unit);

void get\_coordibates\_airbase(qreal lon,qreal lat,GeoDataCoordinates::Unit);

void get\_coordibates\_line\_atack(qreal lon,qreal lat,GeoDataCoordinates::Unit);

void zoom\_map(int value);

void get\_coor\_in\_string(QString string);

void on\_pushButton\_line();

};

#endif // MAINWINDOW\_H

**mainwindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>Оценка боевых возможностей и эффективности боевых действий иА</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QHBoxLayout" name="horizontalLayout\_3">

<item>

<widget class="QFrame" name="frame\_map">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Sunken</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<layout class="QGridLayout" name="map\_lay"/>

</item>

<item>

<widget class="QCustomPlot" name="widget\_legend" native="true">

<property name="maximumSize">

<size>

<width>110</width>

<height>16777215</height>

</size>

</property>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QFrame" name="frame\_table">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Raised</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QTableWidget" name="tableWidget\_position"/>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_2">

<item>

<widget class="QLabel" name="label">

<property name="text">

<string>Задать полосу налета</string>

</property>

</widget>

</item>

<item>

<layout class="QGridLayout" name="lay\_but\_line"/>

</item>

</layout>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

<widget class="QToolBar" name="mainToolBar">

<property name="windowTitle">

<string>toolBar</string>

</property>

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

</widget>

<customwidgets>

<customwidget>

<class>QCustomPlot</class>

<extends>QWidget</extends>

<header>qcustomplot.h</header>

<container>1</container>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>

**maplayer.cpp**

#include "maplayer.h"

#include "mainwindow.h"

TestLayer::TestLayer():LayerInterface()

{

}

bool TestLayer::render(GeoPainter\* painter, ViewportParams\* viewport,

const QString& /\*renderPos\*/, GeoSceneLayer\* /\*layer\*/)

{

if (l\_atack.size()==4)

{

draw\_line\_atack(painter,l\_atack,QColor(255,0,0,50));

}

if (cur\_planes.size()!=0)

{

int number = 1;

foreach (auto p, cur\_planes)

{

if ((p.lon!=0)&&(p.lat!=0))

{

GeoDataCoordinates center(p.lon,p.lat,GeoDataCoordinates::Radian);

painter->drawImage(center,rotate\_image(p.course,

get\_zoom\_image\_plane(get\_zoom\_interval(zoom\_value,

zoom\_vector))));

if (p.hit)

painter->drawImage(center,rotate\_image(p.course,

get\_zoom\_image\_cross(get\_zoom\_interval(zoom\_value,

zoom\_vector))));

painter->setPen(QColor(Qt::yellow));

painter->drawText(center,QString("%1").arg(number));

}

number++;

}

}

if (cur\_airbases.size()!=0)

{

foreach (auto a, cur\_airbases)

{

if ((a.lon!=0)&&(a.lat!=0))

{

GeoDataCoordinates center(a.lon,a.lat,GeoDataCoordinates::Radian);

painter->drawImage(center,rotate\_image(a.course,

get\_zoom\_image\_airbase(get\_zoom\_interval(zoom\_value,

zoom\_vector))));

painter->setPen(QColor(Qt::black));

painter->drawText(center,a.name,get\_sdvig\_text(a.name));

foreach (auto r, a.Svrvb\_vec)

{

draw\_ring(painter,r,center,cur\_planes.value(a.Svrvb\_vec.indexOf(r)).color);

}

}

}

}

return true;

}

QStringList TestLayer::renderPosition() const

{

return QStringList() << "USER\_TOOLS";

}

QVector<qreal> TestLayer::get\_zoom\_vector(int max\_zoom, int min\_zoom)

{

qreal zoom\_step = (max\_zoom-min\_zoom)/7;

QVector <qreal> zoom\_vector;

zoom\_vector.append(min\_zoom);

for (int i=1;i<8;i++)

zoom\_vector.append(min\_zoom+zoom\_step\*i);

return zoom\_vector;

}

int TestLayer::get\_zoom\_interval(int zoom\_value, QVector<qreal> zoom\_vector)

{

for (int i=0;i<zoom\_vector.size();i++)

{

if (i==zoom\_vector.size()-1) return i;

if ((zoom\_vector[i]<=zoom\_value)&&(zoom\_value<zoom\_vector[i+1])) return i;

}

}

QImage TestLayer::get\_zoom\_image\_plane(int interval)

{

switch (interval) {

case 0:

return QImage(":/new/prefix1/build/images/plane\_32.png");

break;

case 1:

return QImage(":/new/prefix1/build/images/plane\_48.png");

break;

case 2:

return QImage(":/new/prefix1/build/images/plane\_64.png");

break;

case 3:

return QImage(":/new/prefix1/build/images/plane\_72.png");

break;

case 4:

return QImage(":/new/prefix1/build/images/plane\_96.png");

break;

case 5:

return QImage(":/new/prefix1/build/images/plane\_128.png");

break;

case 6:

return QImage(":/new/prefix1/build/images/plane\_256.png");

break;

case 7:

return QImage(":/new/prefix1/build/images/plane\_512.png");

break;

default:

break;

}

}

QImage TestLayer::get\_zoom\_image\_cross(int interval)

{

switch (interval) {

case 0:

return QImage(":/new/prefix1/build/images/cross\_32.png");

break;

case 1:

return QImage(":/new/prefix1/build/images/cross\_48.png");

break;

case 2:

return QImage(":/new/prefix1/build/images/cross\_64.png");

break;

case 3:

return QImage(":/new/prefix1/build/images/cross\_72.png");

break;

case 4:

return QImage(":/new/prefix1/build/images/cross\_96.png");

break;

case 5:

return QImage(":/new/prefix1/build/images/cross\_128.png");

break;

case 6:

return QImage(":/new/prefix1/build/images/cross\_256.png");

break;

case 7:

return QImage(":/new/prefix1/build/images/cross\_512.png");

break;

default:

break;

}

}

QImage TestLayer::get\_zoom\_image\_airbase(int interval)

{

switch (interval) {

case 0:

return QImage(":/new/prefix1/build/images/airbase\_32.png");

break;

case 1:

return QImage(":/new/prefix1/build/images/airbase\_48.png");

break;

case 2:

return QImage(":/new/prefix1/build/images/airbase\_64.png");

break;

case 3:

return QImage(":/new/prefix1/build/images/airbase\_72.png");

break;

case 4:

return QImage(":/new/prefix1/build/images/airbase\_96.png");

break;

case 5:

return QImage(":/new/prefix1/build/images/airbase\_128.png");

break;

case 6:

return QImage(":/new/prefix1/build/images/airbase\_256.png");

break;

case 7:

return QImage(":/new/prefix1/build/images/airbase\_512.png");

break;

default:

break;

}

}

QImage TestLayer::rotate\_image(qreal course, QImage image)

{

QImage image\_return;

QImage image\_for\_return = image;

QTransform transform;

transform.rotate(course);

image\_return = image.transformed(transform);

return image\_return;

}

int TestLayer::get\_sdvig\_text(QString name)

{

int size = name.size();

return -size\*3;

}

void TestLayer::draw\_ring(GeoPainter \*painter, qreal radius, GeoDataCoordinates center, QColor color)

{

GeoDataLinearRing ring;

for (qreal a=0.0;a<=360.0;a+=1.0) ring.append(center.moveByBearing(a/57.3,(radius)/6371));

painter->setPen(QPen(color,2));

painter->setBrush(QBrush(QColor(0,0,0,0)));

painter->drawPolygon(ring);

}

void TestLayer::draw\_line\_atack(GeoPainter \*painter, QVector<GeoDataCoordinates> l\_atack, QColor color)

{

GeoDataLinearRing polygon\_atack;

polygon\_atack.append(l\_atack.value(0));

polygon\_atack.append(l\_atack.value(1));

polygon\_atack.append(l\_atack.value(3));

polygon\_atack.append(l\_atack.value(2));

painter->setPen(QPen(QColor(0,0,0,0)));

painter->setBrush(QBrush(color));

//painter->setBrush(QBrush(color,Qt::DiagCrossPattern));

painter->drawPolygon(polygon\_atack);

}

QColor TestLayer::get\_random\_color()

{

int r = Random::get(0,255);

int g = Random::get(0,255);

int b = Random::get(0,255);

return QColor(r,g,b);

}

TestLayer::~TestLayer()

{

}

**maplayer.h**

#ifndef MAPLAYER\_H

#define MAPLAYER\_H

#include "includes.h"

#include "plane.h"

#include "airbase.h"

#include "random.h"

using namespace Marble;

class TestLayer : public LayerInterface

{

virtual bool render(GeoPainter\* painter, ViewportParams\* viewport,

const QString& renderPos = "NONE", GeoSceneLayer\* layer = nullptr)override;

virtual QStringList renderPosition() const override;

public:

TestLayer();

~TestLayer()override;

int maximum\_zoom;

int minimum\_zoom;

int zoom\_value;

QVector <qreal> zoom\_vector;

QVector <qreal> get\_zoom\_vector(int max\_zoom, int min\_zoom);

QVector <plane> cur\_planes;

QVector <airbase> cur\_airbases;

QVector <GeoDataCoordinates> l\_atack;

private:

int get\_zoom\_interval(int zoom\_value, QVector<qreal> zoom\_vector);

QImage get\_zoom\_image\_plane(int interval);

QImage get\_zoom\_image\_cross(int interval);

QImage get\_zoom\_image\_airbase(int interval);

QImage rotate\_image(qreal course, QImage image);

int get\_sdvig\_text(QString name);

void draw\_ring(GeoPainter \*painter, qreal radius, GeoDataCoordinates center, QColor color);

void draw\_line\_atack(GeoPainter \*painter, QVector <GeoDataCoordinates> l\_atack, QColor color);

QColor get\_random\_color();

};

#endif // MAPLAYER\_H

**plane.h**

#ifndef PLANE\_H

#define PLANE\_H

#include <QColor>

#include <QVector>

struct plane{

QString name;

qreal tpass;

qreal tn;

qreal tr;

qreal t180;

qreal tvm;

qreal Sn;

qreal Sr;

qreal Lsm;

qreal Svm;

qreal Vc;

qreal Vgp;

qreal Sish;

qreal Dk;

qreal Ssum;

qreal tsum;

qreal Svrvb;

int alt\_type;

int sfer\_type;

qreal lon;

qreal lat;

qreal course;

QVector <qreal> range\_to\_airbases;

bool main\_plane;

QColor color;

qreal plane\_point\_range;

QVector <int> airbase\_hit\_index;

bool hit;

};

#endif // PLANE\_H

**random.h**

#ifndef RANDOM\_HPP

#define RANDOM\_HPP

#include <random>

namespace details

{

/// True if type T is applicable by a std::uniform\_int\_distribution

template<class T>

struct is\_uniform\_int {

static constexpr bool value =

std::is\_same<T, short>::value ||

std::is\_same<T, int>::value ||

std::is\_same<T, long>::value ||

std::is\_same<T, long long>::value ||

std::is\_same<T, unsigned short>::value ||

std::is\_same<T, unsigned int>::value ||

std::is\_same<T, unsigned long>::value ||

std::is\_same<T, unsigned long long>::value;

};

/// True if type T is applicable by a std::uniform\_real\_distribution

template<class T>

struct is\_uniform\_real {

static constexpr bool value =

std::is\_same<T, float>::value ||

std::is\_same<T, double>::value ||

std::is\_same<T, long double>::value;

};

}

class Random

{

template <class T> using IntDist = std::uniform\_int\_distribution<T>;

template <class T> using RealDist = std::uniform\_real\_distribution<T>;

public:

template <class T>

static typename std::enable\_if<details::is\_uniform\_int<T>::value, T>::type get(T from = std::numeric\_limits<T>::min(), T to = std::numeric\_limits<T>::max())

{

if (from > to) std::swap(from, to);

IntDist<T> dist{from, to};

return dist(instance().engine());

}

template <class T>

static typename std::enable\_if<details::is\_uniform\_real<T>::value, T>::type get(T from = std::numeric\_limits<T>::min(), T to = std::numeric\_limits<T>::max())

{

if (from > to) std::swap(from, to);

RealDist<T> dist{from, to};

return dist(instance().engine());

}

std::mt19937& engine() { return m\_mt; }

protected:

static Random& instance()

{

static Random inst;

return inst;

}

private:

std::random\_device m\_rd; // Устройство генерации случайных чисел

std::mt19937 m\_mt; // Стандартный генератор случайных чисел

Random() : m\_mt(m\_rd()) {}

~Random() {}

Random(const Random&) = delete;

Random& operator = (const Random&) = delete;

};

#endif // RANDOM\_HPP

**resultwindow.cpp**

#include "resultwindow.h"

#include "ui\_resultwindow.h"

ResultWindow::ResultWindow(QWidget \*parent, QVector <plane> current\_planes, QVector<airbase> current\_airbase) :

QMainWindow(parent),

ui(new Ui::ResultWindow)

{

ui->setupUi(this);

create\_destroy\_widget(ui->lay\_for\_table,current\_planes,current\_airbase);

update\_frame\_height(row\_heights,ui->frame\_plane\_table);

draw\_graph(ui->widget\_graph,current\_planes,current\_airbase);

}

ResultWindow::~ResultWindow()

{

delete ui;

}

void ResultWindow::create\_destroy\_widget(QHBoxLayout \*&lay, QVector<plane> current\_planes, QVector<airbase> current\_airbase)

{

for (int j=0;j<current\_airbase.size();j++)

{

lay->addWidget(create\_airbase\_table(current\_planes,current\_airbase,j));

}

lay->addWidget(create\_result\_table(current\_planes));

}

void ResultWindow::correct\_visual(QTableWidget \*&tableWidget)

{

tableWidget->horizontalHeader()->resizeContentsPrecision();

tableWidget->horizontalHeader()->setSectionResizeMode(0,QHeaderView::Stretch);

tableWidget->verticalHeader()->resizeContentsPrecision();

tableWidget->verticalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->horizontalHeader()->setDefaultAlignment(Qt::AlignCenter);

tableWidget->resizeColumnsToContents();

tableWidget->resizeRowsToContents();

}

QTableWidget\* ResultWindow::create\_result\_table(QVector<plane> current\_planes)

{

QTableWidget \*tableWidget\_params = new QTableWidget();

QStringList horiz\_header;

tableWidget\_params->setRowCount(3);

tableWidget\_params->setColumnCount(2);

horiz\_header<<"Параметр"<<"Значение";

tableWidget\_params->setHorizontalHeaderLabels(horiz\_header);

tableWidget\_params->verticalHeader()->hide();

tableWidget\_params->setItem(0,0, new QTableWidgetItem("Всего самолетов"));

tableWidget\_params->setItem(0,1, new QTableWidgetItem(QString("%1").arg(current\_planes.size())));

tableWidget\_params->setItem(1,0, new QTableWidgetItem("Уничтожено"));

tableWidget\_params->setItem(1,1, new QTableWidgetItem(QString("%1").arg(get\_count\_destroy\_planes(current\_planes))));

tableWidget\_params->setItem(2,0, new QTableWidgetItem("Эффективность"));

tableWidget\_params->setItem(2,1, new QTableWidgetItem(QString::number(get\_w(get\_count\_destroy\_planes(current\_planes),current\_planes.size()),'f',2)));

correct\_visual(tableWidget\_params);

row\_heights.append(get\_row\_height(tableWidget\_params));

return tableWidget\_params;

}

QTableWidget \*ResultWindow::create\_airbase\_table(QVector<plane> current\_planes, QVector<airbase> current\_airbase, int j)

{

QTableWidget \*tableWidget\_airbase = new QTableWidget();

QStringList horiz\_header;

tableWidget\_airbase->setRowCount(0);

tableWidget\_airbase->setColumnCount(1);

horiz\_header<<current\_airbase[j].name;

tableWidget\_airbase->setHorizontalHeaderLabels(horiz\_header);

int plane\_index = 1;

foreach (auto p, current\_planes)

{

foreach (auto i, p.airbase\_hit\_index)

{

if (i==j)

{

tableWidget\_airbase->setRowCount(tableWidget\_airbase->rowCount()+1);

tableWidget\_airbase->setItem(tableWidget\_airbase->rowCount()-1,0,new QTableWidgetItem(QString("индекс %1\n%2").arg(plane\_index).arg(p.name)));

}

}

plane\_index++;

}

correct\_visual(tableWidget\_airbase);

row\_heights.append(get\_row\_height(tableWidget\_airbase));

return tableWidget\_airbase;

}

int ResultWindow::get\_count\_destroy\_planes(QVector<plane> current\_planes)

{

int destroy\_planes = 0;

foreach (auto p,current\_planes)

{

if (p.airbase\_hit\_index.size()!=0)

destroy\_planes++;

}

return destroy\_planes;

}

int ResultWindow::get\_row\_height(QTableWidget \*&tableWidget)

{

int row\_height = 0;

for (int i=0;i<tableWidget->rowCount();i++)

{

row\_height+=tableWidget->rowHeight(i);

}

return row\_height;

}

qreal ResultWindow::get\_w(int destroy, int all)

{

qreal dest = destroy;

qreal size = all;

return dest/size;

}

void ResultWindow::update\_frame\_height(QVector<int> row\_heights, QFrame \*&frame)

{

qSort(row\_heights);

qDebug()<<"Высота фрейма"<<row\_heights.last();

frame->setMaximumHeight(row\_heights.last()+95);

}

void ResultWindow::draw\_graph(QCustomPlot \*&widget, QVector<plane> current\_planes, QVector<airbase> current\_airbase)

{

QVector <int> hit\_counts;

widget->setInteraction(QCP::iRangeZoom,true);

widget->setInteraction(QCP::iRangeDrag,true);

widget->xAxis->setLabel("Номер аэродрома");

widget->yAxis->setLabel("Количество уничтоженных самолетов");

widget->xAxis->setRange(0,current\_airbase.size()+1);

widget->yAxis->setRange(0,get\_max\_el(get\_hit\_counts(current\_planes,current\_airbase))+1);

QVector <int> y = get\_hit\_counts(current\_planes,current\_airbase);

QVector <int> x = get\_x\_vector(current\_airbase);

for (int i=0;i<x.size();i++)

{

int r = Random::get(0,255);

int g = Random::get(0,255);

int b = Random::get(0,255);

qreal x\_one = x.at(i);

qreal y\_one = y.at(i);

widget->addGraph();

widget->graph()->addData(x\_one,y\_one);

widget->graph()->setPen(QPen(QColor(r,g,b,200),100));

widget->graph()->setLineStyle((QCPGraph::LineStyle)(5));

widget->graph()->setName(get\_airbase\_names(current\_airbase).at(i));

widget->replot();

}

widget->legend->setVisible(true);

widget->replot();

}

QVector<int> ResultWindow::get\_hit\_counts(QVector<plane> current\_planes, QVector<airbase> current\_airbase)

{

QVector <int> hit\_counts;

for (int j=0;j<current\_airbase.size();j++)

{

int hit\_count = 0;

foreach (auto p, current\_planes)

{

foreach (auto i, p.airbase\_hit\_index)

{

if (i==j)

{

hit\_count++;

}

}

}

qDebug()<<"Сбито самолетов"<<hit\_count;

hit\_counts.append(hit\_count);

}

return hit\_counts;

}

int ResultWindow::get\_max\_el(QVector<int> vector)

{

qSort(vector);

return vector.last();

}

QVector<int> ResultWindow::get\_x\_vector(QVector<airbase> current\_airbase)

{

QVector <int> vec\_for\_return;

int i=1;

foreach (auto iter, current\_airbase) {

vec\_for\_return.append(i);

i++;

}

return vec\_for\_return;

}

QStringList ResultWindow::get\_airbase\_names(QVector<airbase> current\_airbase)

{

QStringList airbase\_names;

foreach (auto a,current\_airbase)

airbase\_names.append(a.name);

return airbase\_names;

}

**resultwindow.h**

#ifndef RESULTWINDOW\_H

#define RESULTWINDOW\_H

#include <QMainWindow>

#include <QTableWidget>

#include <QHBoxLayout>

#include <QHeaderView>

#include <QDebug>

#include "plane.h"

#include "airbase.h"

#include "qcustomplot.h"

#include "random.h"

namespace Ui {

class ResultWindow;

}

class ResultWindow : public QMainWindow

{

Q\_OBJECT

public:

ResultWindow(QWidget \*parent, QVector <plane> current\_planes, QVector <airbase> current\_airbase);

~ResultWindow();

private:

Ui::ResultWindow \*ui;

void create\_destroy\_widget(QHBoxLayout \*&lay, QVector <plane> current\_planes, QVector <airbase> current\_airbase);

void correct\_visual(QTableWidget \*&tableWidget);

QTableWidget\* create\_result\_table(QVector <plane> current\_planes);

QTableWidget\* create\_airbase\_table(QVector <plane> current\_planes, QVector <airbase> current\_airbase, int j);

int get\_count\_destroy\_planes(QVector <plane> current\_planes);

int get\_row\_height(QTableWidget \*&tableWidget);

qreal get\_w(int destroy, int all);

QVector <int> row\_heights;

void update\_frame\_height(QVector <int> row\_heights, QFrame \*&frame);

void draw\_graph(QCustomPlot \*&widget, QVector <plane> current\_planes, QVector <airbase> current\_airbase);

QVector <int> get\_hit\_counts(QVector <plane> current\_planes, QVector <airbase> current\_airbase);

int get\_max\_el(QVector <int> vector);

QVector <int> get\_x\_vector(QVector <airbase> current\_airbase);

QStringList get\_airbase\_names(QVector <airbase> current\_airbase);

};

#endif // RESULTWINDOW\_H

**resultwindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>ResultWindow</class>

<widget class="QMainWindow" name="ResultWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>958</width>

<height>721</height>

</rect>

</property>

<property name="windowTitle">

<string>Результаты моделирования</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QFrame" name="frame\_plane\_table">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Raised</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QHBoxLayout" name="horizontalLayout\_2">

<item>

<widget class="QGroupBox" name="groupBox">

<property name="title">

<string>Распределение потенциально уничтоженных самолетов</string>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<layout class="QHBoxLayout" name="lay\_for\_table"/>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QFrame" name="frame\_graph">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Sunken</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QCustomPlot" name="widget\_graph" native="true"/>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>958</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<customwidgets>

<customwidget>

<class>QCustomPlot</class>

<extends>QWidget</extends>

<header>qcustomplot.h</header>

<container>1</container>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>

**settingswindow.cpp**

#include "settingswindow.h"

#include "ui\_settingswindow.h"

SettingsWindow::SettingsWindow(QWidget \*parent, QString map\_theme\_id, int projection) :

QMainWindow(parent),

ui(new Ui::SettingsWindow)

{

ui->setupUi(this);

qDebug()<<map\_theme\_id<<projection;

QDir dir("./data/maps/earth");

QFileInfoList dirContent = dir.entryInfoList(QStringList()

<< "\*",

QDir::Dirs | QDir::NoDotAndDotDot);

QStringList list\_of\_theme;

foreach (auto d, dirContent)

list\_of\_theme.append(QString("earth/%1/%1.dgml").arg(d.fileName()));

ui->listWidget\_theme->addItems(list\_of\_theme);

ui->listWidget\_theme->setCurrentRow(list\_of\_theme.indexOf(map\_theme\_id));

QStringList list\_of\_projection;

list\_of\_projection<<"Сферическая"

<<"Меркатор"

<<"Плоская";

ui->listWidget\_projection->addItems(list\_of\_projection);

ui->listWidget\_projection->setCurrentRow(projection);

}

SettingsWindow::~SettingsWindow()

{

delete ui;

}

void SettingsWindow::on\_pushButton\_clicked()

{

emit send\_map\_settings(ui->listWidget\_theme->currentItem()->text(),ui->listWidget\_projection->currentRow());

this->close();

}

**settingswindow.h**

#ifndef SETTINGSWINDOW\_H

#define SETTINGSWINDOW\_H

#include <QMainWindow>

#include <includes.h>

namespace Ui {

class SettingsWindow;

}

class SettingsWindow : public QMainWindow

{

Q\_OBJECT

public:

SettingsWindow(QWidget \*parent, QString map\_theme\_id, int projection);

~SettingsWindow();

private slots:

void on\_pushButton\_clicked();

private:

Ui::SettingsWindow \*ui;

signals:

void send\_map\_settings(QString map\_theme, int projection);

};

#endif // SETTINGSWINDOW\_H

**settingswindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>SettingsWindow</class>

<widget class="QMainWindow" name="SettingsWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>468</width>

<height>778</height>

</rect>

</property>

<property name="windowTitle">

<string>Настройка отображения карты</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QVBoxLayout" name="verticalLayout\_4">

<item>

<widget class="QFrame" name="frame">

<property name="frameShape">

<enum>QFrame::Panel</enum>

</property>

<property name="frameShadow">

<enum>QFrame::Raised</enum>

</property>

<property name="lineWidth">

<number>3</number>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QGroupBox" name="groupBox">

<property name="title">

<string>Стиль карты</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QListWidget" name="listWidget\_theme">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>16777215</height>

</size>

</property>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QGroupBox" name="groupBox\_2">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>120</height>

</size>

</property>

<property name="title">

<string>Проекция карты</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QListWidget" name="listWidget\_projection">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>16777215</height>

</size>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton">

<property name="text">

<string>Применить</string>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>468</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

**wIsrt.pro**

QT += core gui printsupport sql axcontainer widgets xml svg network positioning serialport

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

animatedlabel.cpp \

calc.cpp \

datawindow.cpp \

equipwindow.cpp \

main.cpp \

mainwindow.cpp \

maplayer.cpp \

qcustomplot.cpp \

resultwindow.cpp \

settingswindow.cpp

HEADERS += \

airbase.h \

animatedlabel.h \

calc.h \

datawindow.h \

equipwindow.h \

includes.h \

mainwindow.h \

maplayer.h \

plane.h \

qcustomplot.h \

random.h \

resultwindow.h \

settingswindow.h

FORMS += \

datawindow.ui \

equipwindow.ui \

mainwindow.ui \

resultwindow.ui \

settingswindow.ui

win32:RC\_FILE = mainico.rc

INCLUDEPATH += "$$quote(marble)"

DEPENDPATH += $$quote(marble)

LIBS += $$quote(./libmarblewidget-qt5.dll) $$quote(./libastro.dll) $$quote(./libmarbledeclarative.dll)

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

RESOURCES += \

res.qrc