**3 Реализация и тестирование программного продукта**

Реализация программного продукта – это та часть процесса, при которой программист непосредственно создает программный продукт.

Тестирование и отладка программы являются наиважнейшими этапами разработки любых программных продуктов. **Под тестированием понимают** проверку соответствия между реальным и ожидаемым поведением программы, осуществляемую на конечном наборе тестов, выбранном определенным образом. В случае выявления не корректной работы программного продукта необходимо проведение работ по их исправлению.

Наиболее целесообразный метод тестирования – проверка работоспособности основных функций программного продукта. Программа должна нормально функционировать при внесении информации, ее просмотре и т.п.

**3.1 Реализация программного продукта**

Процесс создания программного продукта делится на несколько стадий:

– изучение выбранной предметной области,

– обработка полученной информации,

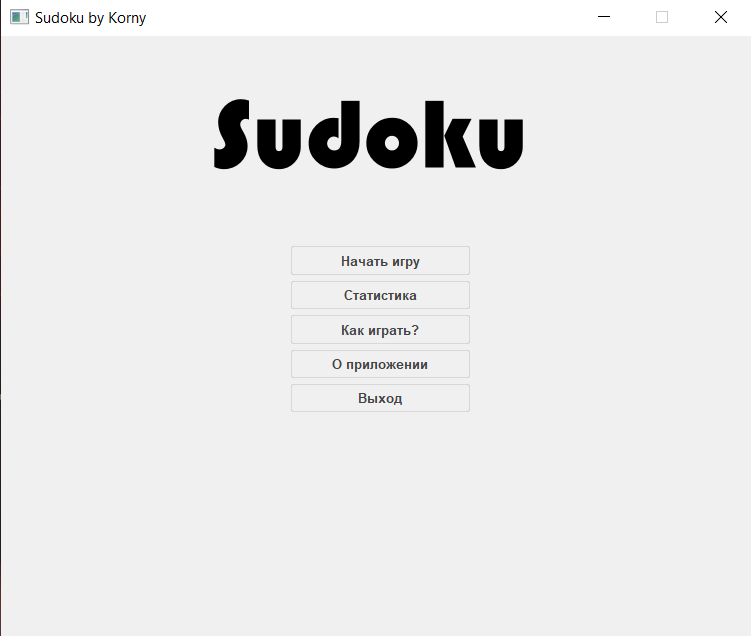
– выбор средств реализации,

– программирование,

– тестирование,

– эксплуатация.

Так выглядит первая форма «Главная форма» (рисунок 3.1).

**Рисунок 3.1 – Главная форма**

На главной форме пользователь может выбрать дальнейшее действие:

– начать игру,

– вызвать форму со статистикой

– вызвать форму со справкой,

– вызвать форму с информацией о программе,

– выйти из приложения.

Текст программного кода, отвечающего за работу некоторых кнопок, конструктора расположенных на главной форме, представлен в листингах 3.1. – 3.5.

**Листинг 3.1 – Организация конструктора основного окна**

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

setFixedSize(750,600);

setWindowIcon(QIcon("png/sudk.png"));

formgm = new Form();

diff = new difficulty();

connect(formgm, &Form::firstwindow,this, &MainWindow::show);

connect(diff, &difficulty::firstwindoww,this, &MainWindow::show);

htp = new howtoplay();

statt = new stats();

QFile styleFile( "styleshits.qss" );

styleFile.open( QFile::ReadOnly );

QString style( styleFile.readAll() );

setStyleSheet( style );

}

**Листинг 3.2 – Организация работы кнопки «Начать игру»**

void MainWindow::on\_pushButton\_2\_clicked()

{

diff->show();

this->close();

}

**Листинг 3.3 – Организация работы кнопки «Статистика»**

void MainWindow::on\_pushButton\_5\_clicked()

{

statt->funcobn();

statt->show();

QFile easyin("stats/easydiff.txt");

if (easyin.open(QIODevice::ReadOnly)){

QTextStream stream(&easyin);

int mas[1000];

int i = 0;

while(!easyin.atEnd()){

mas[i] = easyin.readLine().toInt();

i++;

qDebug() << i << "easy";

}

for (int c = 0;c<i;c++){

qDebug() << mas[c] << "easymid";

}

funcsort(mas,i);

for (int c = 0;c<i;c++){

qDebug() << mas[c] << "funceasymid";

}

for (int k = 0;k<i;k++){

statt->primfunc(mas[k],k);

}

if (i == 1)statt->zamfunc();

else if(i==2) {

statt->zamfunc();

statt->zamfunc2();

}

else if(i==3) {

statt->zamfunc();

statt->zamfunc2();

statt->zamfunc3();

}

easyin.close();

}

/\*выше показан код для реализации считывания информации из файла и обработки этой информации для вывода статистики на экран, но только для лёгкой сложности. Точно такая же реализация для 2 других сложностей, но с заменой названия текстового документа и сменой вызываемых функций\*/

}

**Листинг 3.4 – Организация работы кнопки «Как играть?»**

void MainWindow::on\_pushButton\_3\_clicked()

{

htp->show();

}

**Листинг 3.5 – Организация работы кнопки «О приложении»**

void MainWindow::on\_pushButton\_3\_clicked()

{

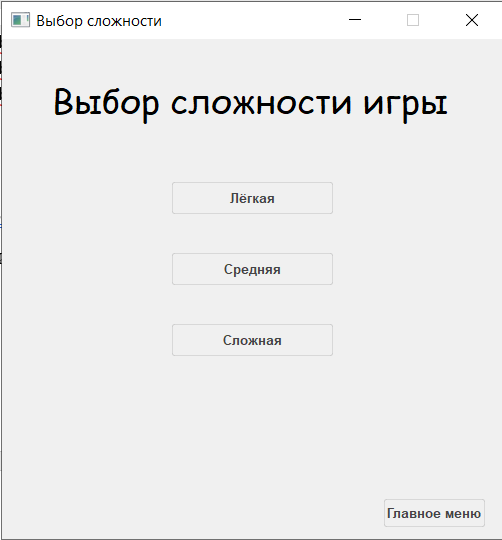
infoboutmain infoboutmain;

infoboutmain.setModal(true);

infoboutmain.exec();

}

Про нажатию на кнопку «Начать игру» открывается форма с выбором сложности игры (рисунок 3.2).

**Рисунок 3.2 – форма «Статистика»**

Текст программного кода, отвечающего за работу каждой кнопки расположен в листинге 3.6.

**Листинг 3.6 – Организация работы кнопок выбора сложности**

void difficulty::on\_pushButton\_clicked()

{

forma->show();

forma->anyfunc(1);

forma->anyfuncname1(ui->pushButton->text());

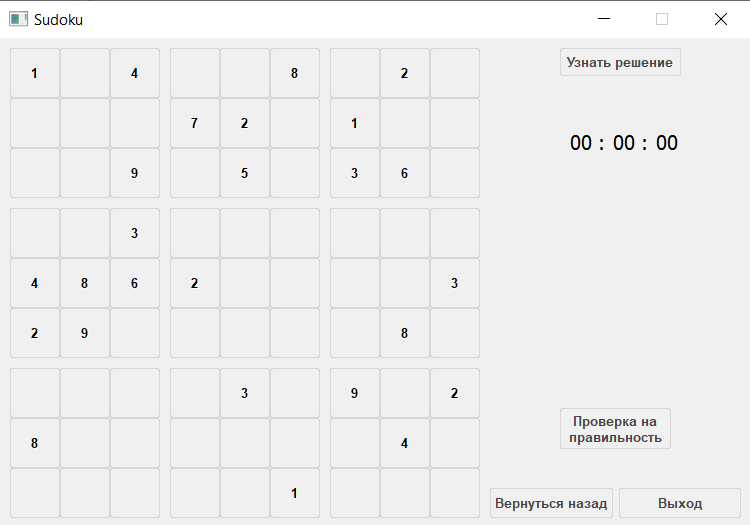
forma->difffunc(ui->pushButton->text());

this->close();

}

/\*Этот листинг показывает работу кнопки «Лёгкая». Для двух других кнопок написан точно такой же код, но изменяется число в функции anyfunc\*/

После выбора сложности открывается игровая форма (рисунок 3.3).



**Рисунок 3.3 – Игровая форма**

На игровой форме пользователь может выполнить следующие действия:

* узнать решение судоку
* вернуться к выбору сложности
* выйти из программы
* проверить свое решение на правильность

Текст программного кода, отвечающего за работу кнопок, расположенных на игровой форме представлен в листингах 3.7-3

**Листинг 3.7 – Организация работы кнопки «Узнать решение»**

void Form::on\_pushButton\_5\_clicked()

{

QMessageBox \*msgBox = new QMessageBox(QMessageBox::Information,"Подсказка","Вы уверены что хотите увидеть решение?",

QMessageBox::Yes| QMessageBox::No);

QFile styleFile( "styleshits.qss" );

styleFile.*open*( QFile::ReadOnly );

QString style( styleFile.readAll() );

msgBox->setStyleSheet(style);

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

{

QPushButton\* my\_array[9][9];

my\_array[0][0] = ui->btn1;/\*эта строчка показывает запись первой кнопки в массив кнопок. Т.к. на поле 81 кнопка, то к остальные кнопки записываются точно так же\*/

for (int i = 0 ;i<9;i++) {

for (int j = 0;j<9;j++) {

grids[i][j] = mas[i][j];

}

}

if (solveSudoku() == true){

for (int i = 0 ;i<9;i++) {

for (int j = 0;j<9;j++) {

my\_array[i][j]->setText(QString::number(grids[i][j]));

}

}

}

delete msgBox;

}

}

**Листинг 3.8 – Организация работы кнопки «Вернуться назад»**

void Form::on\_pushButton\_2\_clicked()

{

this->close();

emit firstwindow();

QPushButton\* my\_array[9][9];

my\_array[0][0] = ui->btn1;/\*эта строчка показывает запись первой кнопки в массив кнопок. Т.к. на поле 81 кнопка, то к остальные кнопки записываются точно так же \*/

for(int i =0;i<9;i++){

for (int j=0;j<9;j++){

my\_array[i][j]->setText("");

my\_array[i][j]->setEnabled(true);

}

}

timesumh = 0;

timesumm = 1;

timesums = 0;

ui->lbls->setText("00");

ui->lblm->setText("00");

ui->lblh->setText("00");

}

**Листинг 3.9 – Организация работы кнопки «Проверка на правильность»**

void Form::on\_pushButton\_4\_clicked()

{

for (int i = 0 ;i<9;i++) {

for (int j = 0;j<9;j++) {

grids[i][j] = mas[i][j];

}

}

solveSudoku();

QPushButton\* my\_array[9][9];

my\_array[0][0] = ui->btn1; /\*эта строчка показывает запись первой кнопки в массив кнопок. Т.к. на поле 81 кнопка, то к остальные кнопки записываются точно так же \*/

for (int i = 0 ;i<9;i++) {

for (int j = 0;j<9;j++) {

if (grids[i][j] != my\_array[i][j]->text().toInt()){

my\_array[i][j]->setStyleSheet("color: red;");

if (my\_array[i][j]->text()==""){

my\_array[i][j]->setStyleSheet("color: black;");

}

}

else {

if (my\_array[i][j]->isEnabled()){

my\_array[i][j]->setStyleSheet("color: green;");

}else my\_array[i][j]->setStyleSheet("color: black;");

if (my\_array[i][j]->text()==""){

my\_array[i][j]->setStyleSheet("color: black;");

}

}

}

}

for (int i = 0 ;i<9;i++) {

for (int j = 0;j<9;j++) {

qDebug() << mas[i][j];

if (grids[i][j]!=my\_array[i][j]->text().toInt()){

QMessageBox::information(this,"Ошибка","Судоку решено неверно");

goto link;

}

}

}

tmr->stop();

allsum = ui->lbls->text().toInt() + ui->lblm->text().toInt()\*60 + ui->lblh->text().toInt()\*3600;

if (diffname == "Лёгкая"){

QFile out("stats/easydiff.txt");

if(out.*open*(QIODevice::Append)){

QTextStream stream(&out);

stream << allsum << "\n";

out.*close*();

}

} /\*эта проверка служит для записи результатов в файл, но для сложности Лёгкая, для двух других сложностей код точно такой же, но с соответствующей заменой\*/

ui->pushButton\_4->setEnabled(false);

for(int i =0;i<9;i++){

for (int j = 0;j<9;j++){

my\_array[i][j]->setEnabled(false);

}

}

fnl->show();

fnl->anyfunctime(allsum);

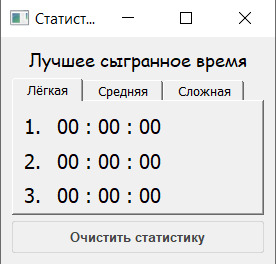
link:

qDebug() << "bad";

}

По нажатию на кнопку «Статистика» открывается форма со статистикой (таблицей рекордов).

Вторая форма – форма «Статистика» (рисунок 3.2).



**Рисунок 3.4 – форма «Статистика»**

На форме «Статистика» пользователь может выполнить следующие действия:

* просмотреть статистику по трем уровням сложности,
* очистить всю статистику,

Текст программного кода, отвечающего за работу кнопки, расположенной на форме «Статистика» представлен в листинге 3.6.

**Листинг 3.6 – Организация работы кнопки «Очистить статистику»**

void stats::on\_pushButton\_clicked()

{

QFile easydiff("stats/easydiff.txt");

if (easydiff.open(QIODevice::WriteOnly | QIODevice::Truncate)){

easydiff.close();

}

QFile middiff("stats/middiff.txt");

if(middiff.open(QIODevice::WriteOnly | QIODevice::Truncate)){

middiff.close();

}

QFile strdiff("stats/strdiff.txt");

if(strdiff.open(QIODevice::WriteOnly | QIODevice::Truncate)){

middiff.close();

}

funcobn();

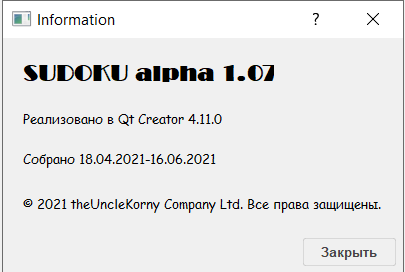
}

При нажатии на кнопку «Как играть?» вызывается новое окно со справкой по игре (рисунок 3.6).



**Рисунок 3.5 – Форма «Как играть?»**

Из главной формы пользователь может вызвать форму «О приложении» (рисунок 3.6).



**Рисунок 3.6 – Форма «О программе»**

После завершения написания кода, разработанный программный продукт готов к тестированию.

**3.2 Тестирование программного продукта**

Тестирование программного продукта – процесс исследования программного обеспечения с целью получения информации о качестве продукта и выявления ошибок.

Существующие на сегодняшний день методы тестирования программного обеспечения не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого программного обеспечения.

Такой процесс формальной проверки или верификации может доказать, что дефекты отсутствуют с точки зрения используемого метода (то есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла программного обеспечения).

Существует множество подходов к решению задачи тестирования и верификации программного обеспечения, но эффективное тестирование сложных программных продуктов – это процесс творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

3.3 Функциональное тестирование

Функциональное тестирование – это [тестирование программного](http://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) обеспечения в целях проверки реализуемости функциональных [требований](http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E), то есть способности программного обеспечения в определённых условиях решать задачи, нужные пользователям. Тестирование программного продукта проводилось методом функционального тестирования.

Разработка программного продукта «Sudoku» производилось на ноутбуке с операционной системой «Windows 10 (x64)», оперативной памятью 8 Гб, процессором «Intel Core I5-8265u 1.6 GHz».

В результате тестирования было выявлено, что для корректной работы системы «Sudoku» компьютер должен обладать следующими минимальными характеристиками:

− процессор с частотой от 1200 Мгц,

− объем оперативной памяти не менее 512 Мб,

− объем внутренней или внешней памяти 40 Мб,

− модем с доступом в сеть Интернет со скоростью не менее 128 Кбит/с,

− операционная система: «Windows 7/8/8.1/10».

При конечном тестировании приложения «Sudoku» было выявлено, что программное средство работает корректно и стабильно при правильном его использовании.