**Санкт-Петербургский государственный политехнический университет**

**Факультет технической кибернетики**

**Кафедра компьютерных систем и программных технологий**

**Курсовая работа по курсу «Программирование»**

**Тема работа: «Разработка приложения с графическим интерфейсом на языке C++ с использованием библиотеки QT»**

**Выполнил: студент Киселёв Антон,**

**Группа 1081/3**

**Преподаватель: доцент Пышкин Е.В.**

**Санкт-Петербург, 2012**

1. **Описание и задачи проекта:**

**В качестве практического задания я решил взять игру «Судоку». Смысл игры в заполнении пустых клеток квадратной матрицы чисел 9 на 9. В игре предусмотрено несколько различных уровней сложностей. Разрабатывая данный проект, передо мной стояло несколько задач.**

**Во-первых, сгенерировать игровое поле; для этого я сформировал двумерный массив 9 на 9, который заполняется случайными числами, с учетом необходимых требований строки, столбца и квадрата – объектов поля, которые и определяют условия заполнения поля случайными числами.  
Во-вторых, разработать игровую механику. Я создал небольшое меню и в игре установил порядок заполнения поля игроком и проверку поля на ошибки, которые может совершать игрок.**

**В-третьих, разработать прототип графического интерфейса. Я реализовал методы вывода игрового поля, выбора меню, появления диалогов и окон.**

1. **Описание работы бизнес логики приложения.**

**Бизнес логика приложения состоит из 4 классов: Field, SavedGames, Game, Menu.   
Класс игровое поле – Field – отвечает за генерацию математической модели игрового поля. Сгенерированное поле сохраняется в массиве – GameField, который и будет изменяться при каждой команде от пользователя. Также создано еще один массив – FieldVictory – для хранения игрового поля, еще на заполненного нулями, оно используется для выявления ошибок игрока. Массив parameters используется для сохранения координат клеток и текста в графическом игровом поле для определения координаты клетки при нажатии мыши.**#ifndef FIELD\_H

#define FIELD\_H

#include <stdlib.h>

class Field

{

void InitField();

void Generation(int chislo);

int IsSq(int ixRow, int ixCol, int chislo);

int IsVer(int ixRow, int chislo);

int IsHor(int ixCol, int chislo);

public:

//Графические параметры

struct Graphics

{

int point\_x;//Строковая координата начала клетки

int point\_y;//Столбцовая координата начала клетки

int p\_x;//Строковая координата начала текста

int p\_y;//Столбцовая координата начала текста

};

int xx;

int yy;

int\* \*FieldVictory;//Поле для проверки на ошибки

int\* \*GameField;//Игровое поле

Graphics parameters[ 9 ][ 9 ];//Массив графических данных

Field(void);

int CountZero();

void Fill\_Zero(int NumberOfZero);

void Fill\_Zero\_Square(int a, int b, int NumberOfZero);

void InsertChislo(int ixRow, int ixCol, int chislo);

int CountOfMistakes();

bool ControlOfChisel(int ixRow, int ixCol);

~Field(void);

int GetCell(int ixRow, int ixCol);

bool Define\_Victory();

void Graphic\_Parameters(int ixRow, int ixCol, int point\_1, int point\_2, int p\_1, int p\_2);

};

#endif // FIELD\_H  
#include "field.h"

using namespace std;

Field::Field()

{

GameField = new int \*[ 9 ];

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

GameField[ i ] = new int [ 9 ];

FieldVictory = new int \*[ 9 ];

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

FieldVictory[ i ] = new int [ 9 ];

InitField();

int chislo = 3;

Generation(chislo);

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

FieldVictory[ ixRow ][ ixCol ] = GameField[ ixRow ][ ixCol ];

}

//Заполнение массива готовым образцом

void Field::InitField()

{

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

GameField[ ixRow ][ ixCol ] = -1;

}

//Генерация полного поля

void Field::Generation(int chislo)

{

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

int count = 9;

for (int var = 1; var < 10; ++var)

{

if (IsSq(ixRow, ixCol, var) || IsHor(ixCol, var) || IsVer(ixRow, var))

--count;

}

if (!count)

{

InitField();

Generation(chislo);

return;

}

while (IsSq(ixRow, ixCol, chislo) || IsHor(ixCol, chislo) || IsVer(ixRow, chislo))

chislo = rand()%9+1;

GameField[ ixRow ][ ixCol ] = chislo;

}

}

}

//Определение количества пропусков

int Field::CountZero()

{

int count = 0;

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField[ ixRow ][ ixCol ] == 0)

count++;

}

}

return count;

}

//Заполнение пропусками

void Field::Fill\_Zero(int NumberOfZero)

{

Fill\_Zero\_Square(0, 0, NumberOfZero);

Fill\_Zero\_Square(0, 3, NumberOfZero);

Fill\_Zero\_Square(0, 6, NumberOfZero);

Fill\_Zero\_Square(3, 0, NumberOfZero);

Fill\_Zero\_Square(3, 3, NumberOfZero);

Fill\_Zero\_Square(3, 6, NumberOfZero);

Fill\_Zero\_Square(6, 0, NumberOfZero);

Fill\_Zero\_Square(6, 3, NumberOfZero);

Fill\_Zero\_Square(6, 6, NumberOfZero);

}

//Заполнение каждого отдельного квадрата 3 на 3 пропусками

void Field::Fill\_Zero\_Square(int a, int b, int NumberOfZero)

{

for (int i = 0; i < NumberOfZero; i++)

{

int ixRow = rand()%3+a;

int ixCol = rand()%3+b;

GameField[ ixRow ][ ixCol ] = 0;

}

}

//Проверка по квадрату

int Field::IsSq(int ixRow, int ixCol, int chislo)

{

int SqIxRow = ixRow/3;

int SqIxCol = ixCol/3;

for (int \_b(SqIxRow\*3), ddi = \_b; ddi < \_b+3; ++ddi)

for (int \_\_b(SqIxCol\*3), ddj = \_\_b; ddj < \_\_b+3; ++ddj)

if (GameField[ ddi ][ ddj ] == chislo)

return 1;

return 0;

}

//Проверка по столбцу

int Field::IsVer(int ixRow, int chislo)

{

for (int SqIxCol = 0; SqIxCol < 9; ++SqIxCol)

if (GameField[ ixRow ][ SqIxCol ] == chislo)

return 1;

return 0;

}

//Проверка по строке

int Field::IsHor(int ixCol, int chislo)

{

for (int SqIxRow = 0; SqIxRow < 9; ++SqIxRow)

if (GameField[ SqIxRow ][ ixCol ] == chislo)

return 1;

return 0;

}

//Подсчет количества ошибок в поле

int Field::CountOfMistakes()

{

int count = 0;

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField[ ixRow ][ ixCol ] != FieldVictory[ ixRow ][ ixCol ])

{

count++;

}

}

}

return count;

}

//Проверка чисел

bool Field::ControlOfChisel(int ixRow, int ixCol)

{

if (GameField[ ixRow ][ ixCol ] != FieldVictory[ ixRow ][ ixCol ])

{

xx = ixRow;

yy = ixCol;

return 1;

}

else

{

return 0;

}

}

//Определение победы игрока

bool Field::Define\_Victory()

{

int count = CountOfMistakes();

if (count == 0)

{

return 1;

}

else

{

return 0;

}

}

//Возврат значения в клетке

int Field::GetCell(int ixRow, int ixCol)

{

return GameField[ ixRow ][ ixCol ];

}

//Вставка значения

void Field::InsertChislo(int ixRow, int ixCol, int chislo)

{

GameField[ ixRow ][ ixCol ] = chislo;

}

//Записываем графические параметры для каждой клетки игрового поля

void Field::Graphic\_Parameters(int ixRow, int ixCol, int point\_1, int point\_2, int p\_1, int p\_2)

{

parameters[ ixRow ][ ixCol ].point\_x = point\_1;

parameters[ ixRow ][ ixCol ].point\_y = point\_2;

parameters[ ixRow ][ ixCol ].p\_x = p\_1;

parameters[ ixRow ][ ixCol ].p\_y = p\_2;

}

Field::~Field(void)

{

}

**Класс SavedGames используется для сохранения модели текущего игрового поля и его загрузке. Все это используется для реализации в игре команд на продолжение игры и сохранения текущей.**#include "savedgames.h"

#include <QFile>

#include <QTextStream>

SavedGames::SavedGames()

{

}

//Сохранение сгенерированного поля

void SavedGames::SaveReadyField(Field\* GameField, char name[])

{

QFile file("C:\\QtProjects\\SUDOKU\\CurrentGameField.txt");

file.*open*(QIODevice::WriteOnly);

QTextStream out(&file);

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

out << GameField->FieldVictory[ ixRow ][ ixCol ] << " ";

out << "\n";

}

file.*close*();

}

//Запрос на сохранение игры

void SavedGames::SaveRequest(Field\* GameField)

{

RecordData("C:\\QtProjects\\SUDOKU\\game.txt", GameField);

}

//Запись текущей игры

void SavedGames::RecordData(char name[], Field\* GameField)

{

QFile file(name);

file.*open*(QIODevice::WriteOnly);

QTextStream out(&file);

/\*if (!out)

{

cout << "Файл не открывается!\n";

}\*/

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

out << GameField->GameField[ ixRow ][ ixCol ];

out << " ";

}

out << "\n";

}

out << "\n";

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

out << GameField->FieldVictory[ ixRow ][ ixCol ];

out << " ";

}

out << "\n";

}

file.*close*();

}

//Запрос на загрузку игры

void SavedGames::LoadRequest(Field\* GameField)

{

LoadGame("C:\\QtProjects\\SUDOKU\\game.txt", GameField);

}

//Загрузка сохранённой игры

void SavedGames::LoadGame(char name[], Field\* GameField)

{

QFile infile(name);

infile.*open*(QIODevice::ReadOnly);

QTextStream in(&infile);

/\*if (!in)

{

cout << "Файл не открылся!\n";

return;

}\*/

QFile outfile("C:\\QtProjects\\SUDOKU\\CurrentGameField.txt");

outfile.open(QIODevice::ReadOnly, QIODevice::WriteOnly);

QTextStream out(&outfile);

/\*if (!out)

{

cout << "Файл не открылся!\n";

return;

}\*/

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

in >> GameField->GameField[ ixRow ][ ixCol ];

}

}

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

in >> GameField->FieldVictory[ ixRow ][ ixCol ];

out << GameField->FieldVictory[ ixRow ][ixCol];

out << " ";

}

out << "\n";

}

infile.*close*();

outfile.*close*();

}

SavedGames::~SavedGames(void)

{

}

**Для работы приложения в консоли я создал специальный класс – Game – в нем производится проверка различных команд на создание поле, выбора уровня сложности, сохранения поля, загрузке поле, вставки значения в ячейку, проверки ошибок и выявления победы игрока.**#ifndef GAME\_H

#define GAME\_H

#include "field.h"

#include "savedGames.h"

#include <iostream>

#include <fstream>

#include <time.h>

#include <cstdlib>

class Games

{

SavedGames \*SavedGame;

void StartGame();

void OutOfField();

void PlayGame();

void GameStep();

void EndGame();

void ControlField();

void SaveGame();

void LoadGame();

public:

Field \*GameField;

Games(void);

int GameLevel(int choice);

void Menu();

~Games(void);

friend class Test;

};

#endif // GAME\_H  
#include "game.h"

using namespace std;

Games::Games(void)

{

srand((unsigned)time(0));

int chislo = rand()%9+1;

char name[ 100 ] = "CurrentGameField.txt";

GameField = new Field;

SavedGame = new SavedGames();

SavedGame->SaveReadyField(GameField, name);

}

Games::~Games(void)

{

delete[] GameField;

delete[] SavedGame;

}

//Игровое меню

void Games::Menu()

{

cout << "Игровое меню: \n";

cout << "1. Играть\n";

cout << "2. Продолжить\n";

cout << "3. Выйти\n";

int choice = 0;

cin >> choice;

if (choice == 1)

{

StartGame();

PlayGame();

EndGame();

}

else if (choice == 2)

{

int choice = 1;

SavedGame->LoadRequest(GameField);

OutOfField();

PlayGame();

EndGame();

}

else if (choice == 3)

{

exit(0);

}

}

//Выбор уровня сложности

int Games::GameLevel(int choice)

{

int NumberOfZero = 0;

if (choice == 1)

{

NumberOfZero = 4;

return NumberOfZero;

}

else if (choice == 2)

{

NumberOfZero = 5;

return NumberOfZero;

}

else if (choice == 3)

{

NumberOfZero = 6;

return NumberOfZero;

}

}

//Начало игры

void Games::StartGame()

{

int CountZero = GameLevel(1);

GameField->Fill\_Zero(CountZero);

OutOfField();

}

//Игровой ход

void Games::GameStep()

{

cout << "Введите координаты и значение, которое вы хотите вставить: \n";

int ixRow = 0;

int ixCol = 0;

int chislo = 0;

cin >> ixRow >> ixCol >> chislo;

ixRow--;

ixCol--;

GameField->InsertChislo(ixRow, ixCol, chislo);

OutOfField();

}

//Проверка поля на ошибки

void Games::ControlField()

{

cout << "Вы желаете проверить текущую игру? \n";

cout << "1. Да\n";

cout << "2. Нет\n";

int choice = 0;

cin >> choice;

if (choice == 1)

{

int count = 0;

cout << "Есть ошибка в клетках с координатами: \n";

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

bool Control = GameField->ControlOfChisel(ixRow, ixCol);

if (Control == 1)

{

int Row = GameField->xx;

int Col = GameField->yy;

Row++;

Col++;

cout << Row << " " << Col << "\n";

}

}

}

}

}

//Игра

void Games::PlayGame()

{

int count = GameField->CountZero();

do

{

GameStep();

int choice = 0;

cout << "Вы желаете сохранить текущую игру?\n";

cout << "1.Да\n";

cout << "2.Нет\n";

cin >> choice;

SavedGame->SaveRequest(GameField);

ControlField();

count--;

} while(count != 0);

}

//Конец игры

void Games::EndGame()

{

int count = 0;

bool victory = GameField->Define\_Victory();

if (victory == 1)

cout << "Вы выиграли!\n";

else if (victory == 0)

{

cout << "Есть ошибка в клетке с координатами: " << "\n";

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

GameField->ControlOfChisel(ixRow, ixCol);

int Row = GameField->xx;

int Col = GameField->yy;

Row++;

Col++;

cout << Row << " " << Col << "\n";

}

}

count = GameField->CountOfMistakes();

do

{

GameStep();

int choice = 0;

cout << "Вы желаете сохранить текущую игру?\n";

cout << "1.Да\n";

cout << "2.Нет\n";

cin >> choice;

SavedGame->SaveRequest(GameField);

ControlField();

count--;

} while(count != 0);

}

OutOfField();

int choice = 0;

cout << "\n";

cout << "\n";

cout << "\n";

cin >> choice;

SavedGame->LoadRequest(GameField);

OutOfField();

}

//Вывод поля

void Games::OutOfField()

{

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

cout << " " << GameField->GameField[ ixRow ][ ixCol ] << " ";

}

cout << "(" << ixRow+1 << ")\n\n";

}

for (int ixRow = 0; ixRow < 9; ixRow++)

cout << "(" << ixRow+1 << ")";

cout << "\n";

}

**Тестовый класс – Test – разработан для проверки методов классов Field и SavedGames.**

#pragma once

#include "Field.h"

#include "SavedGames.h"

#include "Game.h"

class Test

{

Game \*Sudoku;

Field \*GameField;

SavedGames \*SavedGame;

public:

void RecordFirstFile();

void RecordSecondFile();

bool Test1();

bool Test2();

bool Test3();

bool Test4();

bool Test5();

bool Test6();

bool Test7();

bool Test8();

bool Test9();

bool Test10();

Test(void);

bool FullTest();

~Test(void);

};

#include "Test.h"

#include <iostream>

#include <fstream>

#include <time.h>

using namespace std;

Test::Test(void)

{

srand((unsigned)time(0));

int chislo = rand()%9+1;

GameField = new Field(chislo);

SavedGame = new SavedGames();

Sudoku = new Game();

}

Test::~Test(void)

{

delete[] GameField;

delete[] SavedGame;

delete[] Sudoku;

}

/\*Вспомогательные функции тестирования\*/

//Запись в файл

void Test::RecordFirstFile()

{

int ixRow = 0;

int ixCol = 0;

ofstream out("TestField.txt");

if (!out)

{

cout << "Файл не открывается!\n";

}

for (ixRow = 0; ixRow < 9; ixRow++)

{

for (ixCol = 0; ixCol < 9; ixCol++)

{

out << GameField->GameField[ ixRow ][ ixCol ];

out << " ";

}

out << "\n";

}

}

//Считывание из файла

void Test::RecordSecondFile()

{

int ixRow = 0;

int ixCol = 0;

ifstream in("TestField.txt");

if (!in)

{

cout << "Файл не открывается!\n";

}

for (ixRow = 0; ixRow < 9; ixRow++)

{

for (ixCol = 0; ixCol < 9; ixCol++)

{

in >> GameField->FieldVictory[ ixRow ][ ixCol ];

}

}

}

/\*Блок тестов класса Field\*/

//Тест функции определения победы

bool Test::Test1()

{

RecordFirstFile();

RecordSecondFile();

int count = GameField->CountOfMistakes();

if (count == 0)

return 1;

return 0;

}

//Тест функции вставки значения

bool Test::Test2()

{

GameField->InsertChislo(0, 0, 0);

RecordFirstFile();

RecordSecondFile();

if (GameField->FieldVictory[ 0 ][ 0 ] == GameField->GameField[ 0 ][ 0 ])

return 1;

return 0;

}

//Тест функции возврата клетки из поля с данными координатами

bool Test::Test3()

{

int chislo = 0;

GameField->InsertChislo(0, 0, chislo);

if (GameField->GetCell(0, 0) == chislo)

return 1;

return 0;

}

//Тест проверки значений в поле

bool Test::Test4()

{

int count = 0;

int Row = 0;

int Col = 0;

GameField->InsertChislo(0, 5, 0);

GameField->InsertChislo(3, 4, 0);

GameField->InsertChislo(4, 8, 0);

GameField->ControlOfChisel(0, 5);

Row = GameField->xx;

Col = GameField->yy;

if (Row == 0 && Col == 5)

count++;

GameField->ControlOfChisel(3, 4);

Row = GameField->xx;

Col = GameField->yy;

if (Row == 3 && Col == 4)

count++;

GameField->ControlOfChisel(4, 8);

Row = GameField->xx;

Col = GameField->yy;

if (Row == 4 && Col == 8)

count++;

if (count == 3)

return 1;

return 0;

}

//Тест на проверку взятия значения из конкретной клетки

bool Test::Test5()

{

int chislo = GameField->GetCell(0, 0);

if (chislo == GameField->GameField[ 0 ][ 0 ])

return 1;

return 0;

}

//Проверка функции победы в игре на истинность

bool Test::Test6()

{

RecordFirstFile();

RecordSecondFile();

if (GameField->Define\_Victory() == 1)

return 1;

return 0;

}

//Проверка функции победы в игре на ложность

bool Test::Test7()

{

RecordFirstFile();

GameField->InsertChislo(0, 0, 9);

RecordSecondFile();

if (GameField->Define\_Victory() == 0)

return 1;

return 0;

}

/\*Блок тестов класса SavedGames\*/

//Проверка сохранения данных в файле

bool Test::Test8()

{

int Array1[ 9 ][ 9 ];

int Array2[ 9 ][ 9 ];

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

Array1[ ixRow ][ ixCol ] = 1;

Array2[ ixRow ][ ixCol ] = 2;

}

}

int count1 = 0;

int count2 = 0;

GameField = new Field(1);

GameField->Fill\_Zero(1);

SavedGame->RecordData("TestField.txt", GameField);

ifstream in("TestField.txt");

if (!in)

{

cout << "Файл не открывается!\n";

}

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

in >> Array1[ ixRow ][ ixCol ];

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

in >> Array2[ ixRow ][ ixCol ];

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

if (GameField->GameField[ ixRow ][ ixCol ] != Array1[ ixRow ][ ixCol ])

count1++;

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

if (GameField->FieldVictory[ ixRow ][ ixCol ] != Array2[ ixRow ][ ixCol ])

count2++;

if (count1 == 0 && count2 == 0)

return 1;

return 0;

}

//Проверка на загрузку массива из файла

bool Test::Test9()

{

SavedGame->LoadGame("game.txt", GameField);

int Array1[ 9 ][ 9 ];

int Array2[ 9 ][ 9 ];

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

Array1[ ixRow ][ ixCol ] = 1;

Array2[ ixRow ][ ixCol ] = 2;

}

}

ifstream in("game.txt");

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

in >> Array1[ ixRow ][ ixCol ];

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

in >> Array2[ ixRow ][ ixCol ];

int count1 = 0;

int count2 = 0;

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

if (GameField->FieldVictory[ ixRow ][ ixCol ] != Array2[ ixRow ][ ixCol ])

count1++;

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

if (GameField->GameField[ ixRow ][ ixCol ] != Array1[ ixRow ][ ixCol ])

count2++;

if (count1 == 0 && count2 == 0)

return 1;

return 0;

}

//Проверка записи готового сгенерированного поля

bool Test::Test10()

{

int count = 0;

GameField = new Field(1);

SavedGame->SaveReadyField(GameField, "CurrentGameField.txt");

int Array[ 9 ][ 9 ];

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

Array[ ixRow ][ ixCol ] = 1;

}

}

ifstream in("CurrentGameField.txt");

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

in >> Array[ ixRow ][ ixCol ];

for (int ixRow = 0; ixRow < 9; ixRow++)

for (int ixCol = 0; ixCol < 9; ixCol++)

if (GameField->GameField[ ixRow ][ ixCol ] != Array[ ixRow ][ ixCol ])

count++;

if (count == 0)

return 1;

return 0;

}

//Итоговый тест

bool Test::FullTest()

{

if (Test1() == 1 && Test2() == 1 && Test3() == 1 && Test4() == 1 && Test5() == 1 && Test6() == 1 && Test7() == 1 && Test8() == 1 && Test9() == 1 && Test10() == 1)

return 1;

return 0;

}

1. **Описание работы графического интерфейса.**

**Главный модуль – main.**#include <QtGui/QApplication>

#include "mainwindow.h"

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

{

QApplication a(argc, argv);

MainWindow window;

window.setWindowTitle("SUDOKU");

window.show();

return a.exec();

}

**Графический интерфейс включает два класс – mainwindow и windialog.  
MainWindow.**#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPushButton>

#include <QLayout>

#include <QEvent>

#include "savedgames.h"

#include "game.h"

class MainWindow : public QMainWindow

{

Q\_OBJECT

//Игровое меню:

QMenu \*Game;

QAction \*newGameAction;

QAction \*loadGame;

QAction \*saveAction;

QAction \*checkingAction;

QAction \*exitAction;

QMenuBar \*bar;

public:

//Флаги событий

bool is\_New\_Game\_Activated;

bool is\_Load\_Game\_Activated;

bool is\_Mouse\_Clicked;

bool is\_Checking;

Field\* GameField;//Игровое поле

SavedGames \*save\_game;//Сохранение/загрузка игр

//Окно уровня сложности

QPushButton \*easy, \*normal, \*hard;

QGridLayout \*lay, \*savelay;

//Окно сохранения игры

QPushButton \*save, \*cancel;

SavedGames\* SaveGame;

QDialog\* leveldialog;

QDialog\* savedialog;

MainWindow();

~*MainWindow*();

//Определение координаты клетки

int defined\_ixRow;

int defined\_ixCol;

//Графическое поле

void *paintEvent*(QPaintEvent \*);

//Мышь

void *mousePressEvent*(QMouseEvent \* ev);

void insert\_in\_cell(int ixRow, int ixCol);

void CreateNewGame(int choice);

void Save\_Graphic\_Options();

private:

void CreateActions();

void CreateMenu();

private slots:

void SaveDialog();

void LoadGame();

void Checking();

void ChoiceLevel();

void firstlevel();

void secondlevel();

void thirdlevel();

void save\_pressed();

void cancel\_press();

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "savedgames.h"

#include "game.h"

#include "windialog.h"

#include <QWidget>

#include <QEvent>

#include <QMouseEvent>

#include <QtGui>

MainWindow::MainWindow()

{

CreateActions();

CreateMenu();

}

//Действия в меню

void MainWindow::CreateActions()

{

is\_New\_Game\_Activated = false;

is\_Load\_Game\_Activated = false;

is\_Mouse\_Clicked = false;

is\_Checking = false;

newGameAction = new QAction("New Game",this);

connect(newGameAction,SIGNAL(triggered()),this,SLOT(ChoiceLevel()));

loadGame = new QAction("Load Game", this);

connect(loadGame, SIGNAL(triggered()),this,SLOT(LoadGame()));

saveAction = new QAction("Save Game",this);

connect(saveAction,SIGNAL(triggered()),this,SLOT(SaveDialog()));

checkingAction = new QAction("Check",this);

connect(checkingAction,SIGNAL(triggered()),this,SLOT(Checking()));

exitAction = new QAction("Exit",this);

connect(exitAction,SIGNAL(triggered()),this,SLOT(close()));

}

//Игровое меню

void MainWindow::CreateMenu()

{

bar = new QMenuBar(this);

Game = bar->addMenu("Game");

Game->addAction(newGameAction);

Game->addAction(loadGame);

Game->addSeparator();

Game->addAction(checkingAction);

Game->addAction(saveAction);

Game->addSeparator();

Game->addAction(exitAction);

this->setMenuBar(bar);

}

//Рисование поля

void MainWindow::*paintEvent*(QPaintEvent \*)

{

int pointIxRow = 10;

int pointIxCol = 50;

QPen pen;

pen.setWidth(3);

QPainter\* painter = new QPainter;

painter->begin(this);

painter->setPen(pen);

painter->drawRect(10, 50, 90, 90);

painter->drawRect(100, 50, 90, 90);

painter->drawRect(190, 50, 90, 90);

painter->drawRect(10, 140, 90, 90);

painter->drawRect(100, 140, 90, 90);

painter->drawRect(190, 140, 90, 90);

painter->drawRect(10, 230, 90, 90);

painter->drawRect(100, 230, 90, 90);

painter->drawRect(190, 230, 90, 90);

painter->setBrush(QBrush(Qt::green, Qt::Dense4Pattern));

painter->setFont(QFont("Times", 20, QFont::Normal));

pen.setWidth(1);

painter->setPen(pen);

for (int ixRow = 0 ; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

painter->drawRect(pointIxRow, pointIxCol, 30, 30);

pointIxRow = pointIxRow+30;

}

pointIxRow = 10;

pointIxCol = pointIxCol+30;

}

int p1 = 18;

int p2 = 75;

if (is\_New\_Game\_Activated == true)

{

painter->setFont(QFont("Times", 20, QFont::Normal));

for (int ixRow = 0 ; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField->GetCell(ixRow, ixCol) == 0)

{

painter->drawText(p1, p2, " ");

}

else if (GameField->GetCell(ixRow, ixCol) != 0)

{

QString y(QString::number(GameField->GetCell(ixRow, ixCol), 10));

painter->drawText(p1, p2, y);

}

p1 = p1+30;

}

p1 = 18;

p2 = p2+30;

}

}

p1 = 18;

p2 = 75;

if (is\_Load\_Game\_Activated == true)

{

painter->setFont(QFont("Times", 20, QFont::Normal));

for (int ixRow = 0 ; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField->GetCell(ixRow, ixCol) == 0)

{

painter->drawText(p1, p2, " ");

}

else if (GameField->GetCell(ixRow, ixCol) != 0)

{

QString y(QString::number(GameField->GetCell(ixRow, ixCol), 10));

painter->drawText(p1, p2, y);

}

p1 = p1+30;

}

p1 = 18;

p2 = p2+30;

}

}

pointIxRow = 10;

pointIxCol = 50;

p1 = 18;

p2 = 75;

if (is\_Checking == true)

{

int count = 0;

for (int ixRow = 0 ; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField->GameField[ ixRow ][ ixCol ] != GameField->FieldVictory[ ixRow ][ ixCol ])

{

painter->setBrush(QBrush(Qt::red, Qt::Dense4Pattern));

painter->drawRect(pointIxRow, pointIxCol, 30, 30);

pointIxRow = pointIxRow+30;

count++;

}

if (GameField->GameField[ ixRow ][ ixCol ] == GameField->FieldVictory[ ixRow ][ ixCol ])

{

painter->setBrush(QBrush(Qt::green, Qt::Dense4Pattern));

painter->drawRect(pointIxRow, pointIxCol, 30, 30);

pointIxRow = pointIxRow+30;

}

}

pointIxRow = 10;

pointIxCol = pointIxCol+30;

}

painter->setFont(QFont("Times", 20, QFont::Normal));

for (int ixRow = 0 ; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

if (GameField->GetCell(ixRow, ixCol) == 0)

{

painter->drawText(p1, p2, " ");

}

else if (GameField->GetCell(ixRow, ixCol) != 0)

{

QString y(QString::number(GameField->GetCell(ixRow, ixCol), 10));

painter->drawText(p1, p2, y);

}

p1 = p1+30;

}

p1 = 18;

p2 = p2+30;

}

if (count == 0)

{

WinDialog\* win = new WinDialog;

win->show();

is\_Checking = false;

}

}

if (is\_Mouse\_Clicked == true)

{

QString y(QString::number(GameField->GetCell(defined\_ixRow, defined\_ixCol),10));

painter->drawText(GameField->parameters[ defined\_ixRow ][ defined\_ixCol ].p\_x, GameField->parameters[ defined\_ixRow ][ defined\_ixCol ].p\_y, y);

}

this->resize(300, 350);

painter->end();

}

MainWindow::~*MainWindow*()

{

delete[] easy;

delete[] normal;

delete[] hard;

delete[] lay;

}

//Выбор уровня

void MainWindow::ChoiceLevel()

{

leveldialog = new QDialog;

easy = new QPushButton("&Easy");

normal = new QPushButton("&Normal");

hard = new QPushButton("&Hard");

connect(easy, SIGNAL(clicked()),SLOT(firstlevel()));

connect(normal, SIGNAL(clicked()),SLOT(secondlevel()));

connect(hard, SIGNAL(clicked()),SLOT(thirdlevel()));

lay = new QGridLayout;

lay->addWidget(easy, 0, 0);

lay->addWidget(normal, 1, 0);

lay->addWidget(hard, 2, 0);

leveldialog->setLayout(lay);

leveldialog->show();

}

//Начала новой игры

void MainWindow::CreateNewGame(int choice)

{

GameField = new Field;

Save\_Graphic\_Options();

save\_game = new SavedGames;

GameField->Fill\_Zero(choice);

save\_game->SaveReadyField(GameField, "CurrentGameField.txt");

is\_New\_Game\_Activated = true;

is\_Load\_Game\_Activated = false;

is\_Mouse\_Clicked = false;

is\_Checking = false;

if (is\_New\_Game\_Activated == true)

repaint();

}

//Легкий уровень

void MainWindow::firstlevel()

{

int choice = 4;

CreateNewGame(choice);

leveldialog->close();

}

//Средний уровень

void MainWindow::secondlevel()

{

int choice = 5;

CreateNewGame(choice);

leveldialog->close();

}

//Трудный уровень

void MainWindow::thirdlevel()

{

int choice = 6;

CreateNewGame(choice);

leveldialog->close();

}

//Обработка событий мыши

void MainWindow::*mousePressEvent*(QMouseEvent \*ev)

{

if (ev->button() == Qt::RightButton)

{

is\_Mouse\_Clicked = true;

is\_Load\_Game\_Activated = false;

is\_New\_Game\_Activated = true;

is\_Checking = false;

if (is\_Mouse\_Clicked == true)

{

QPoint p = ev->pos();

int ixRow = (p.y()-50)/30;

int ixCol = (p.x()-10)/30;

defined\_ixRow = ixRow;

defined\_ixCol = ixCol;

int test = QInputDialog::getInt(this, "Enter number", "number: ", 0, 1, 9, 1);

GameField->InsertChislo(ixRow, ixCol, test);

repaint();

}

}

}

//Диалог сохранения игры

void MainWindow::SaveDialog()

{

savedialog = new QDialog;

save = new QPushButton("&Save");

cancel = new QPushButton("&Cancel");

savelay = new QGridLayout;

connect(save,SIGNAL(clicked()),SLOT(save\_pressed()));

connect(cancel,SIGNAL(clicked()),SLOT(cancel\_press()));

savelay->addWidget(save, 0, 0);

savelay->addWidget(cancel, 0, 1);

savedialog->setLayout(savelay);

savedialog->show();

}

void MainWindow::save\_pressed()

{

save\_game->SaveRequest(GameField);

savedialog->close();

}

void MainWindow::cancel\_press()

{

savedialog->close();

}

//Загрузка игры

void MainWindow::LoadGame()

{

GameField = new Field;

Save\_Graphic\_Options();

save\_game = new SavedGames;

save\_game->LoadRequest(GameField);

is\_Load\_Game\_Activated = true;

is\_New\_Game\_Activated = false;

is\_Mouse\_Clicked = false;

is\_Checking = false;

if (is\_Load\_Game\_Activated == true)

repaint();

}

//Проверка поля

void MainWindow::Checking()

{

is\_Checking = true;

is\_Load\_Game\_Activated = false;

is\_Mouse\_Clicked = false;

is\_New\_Game\_Activated = false;

if (is\_Checking == true)

repaint();

}

//Сохранение графичеких опций

void MainWindow::Save\_Graphic\_Options()

{

int pointIxRow = 10;

int pointIxCol = 50;

int p1 = 18;

int p2 = 75;

for (int ixRow = 0; ixRow < 9; ixRow++)

{

for (int ixCol = 0; ixCol < 9; ixCol++)

{

GameField->Graphic\_Parameters(ixRow, ixCol, pointIxRow, pointIxCol, p1, p2);

p1 = p1+30;

pointIxRow = pointIxRow+30;

}

p1 = 18;

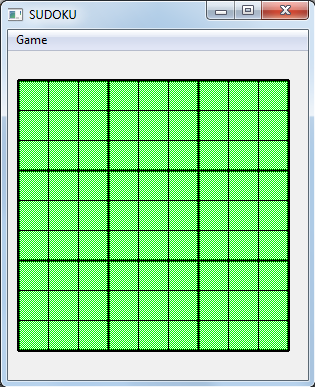
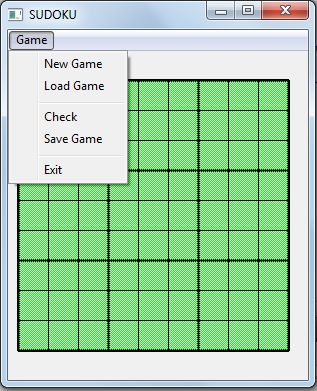
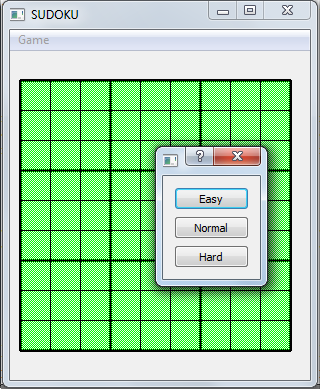
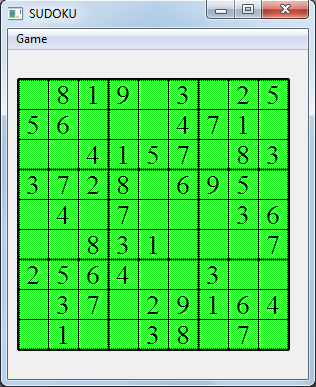
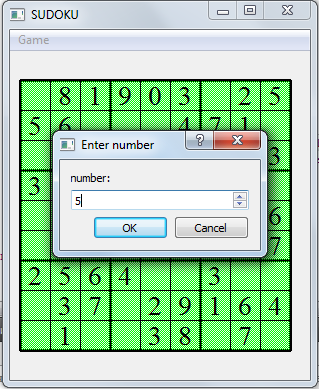
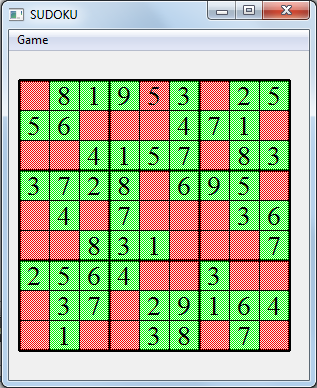
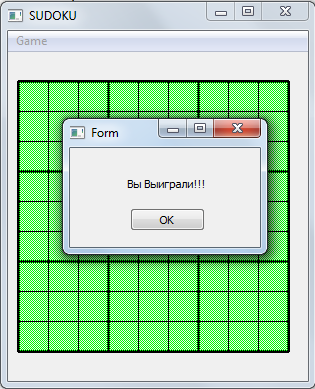
pointIxRow = 10;

pointIxCol = pointIxCol+30;

p2 = p2+30;

}

}

**При запуске приложения окно выглядит следующим образом:  
  
При нажатии кнопки меню:  
  
При выборе новой игры:  
  
При нажатии кнопки – Easy:  
  
В классе я формирую игровое меню содержащее действия: начать новую игру, загрузить, сохранить, проверить и выйти. Для того чтобы вывести игровое поле на экран существует метод paintEvent(). Изначально поле клеток уже присутствует на экране, а для появления текста – рисуется число в клетке с тем значением, что и соответствующее значение в GameField. Управляется появление цифр с помощью флажка – is\_New\_Game\_Activated, которому изначально присвоено значение false, а при нажатии кнопки «новая игра» ему присваивается значение true, и появляются цифры на экране. Чтобы вставить значение в клетку, существует флажок – is\_Mouse\_Clicked, которому изначально присвоено значение false, а при срабатывании щелчка мыши правой кнопкой по полю появляется окно вставки значения, игрок вводит значение и оно отображается в соответствующей клетке, а в соответствующую ячейку объекта GameField вставляется выбранное значение.   
  
При нажатии кнопки Load Game появляется сохранённое игровое поле ранее сохраненной игры. При этом флажок is\_Load\_Game\_Activated меняет свое значение на true.  
При нажатии кнопки – Checking происходит сравнение текущего игрового поля с полем победы – FieldVictory. Клетки, в которых оказываются ошибки, окрашиваются в красный цвет. При этом флажок is\_Checking меняет свое значение на true.  
  
Если все значения оказываются правильными, то появляется сообщение о победе – windialog.  
**

#ifndef WINDIALOG\_H

#define WINDIALOG\_H

#include <QWidget>

namespace Ui {

class WinDialog;

}

class WinDialog : public QWidget

{

Q\_OBJECT

public:

explicit WinDialog(QWidget \*parent = 0);

~*WinDialog*();

protected:

private slots:

void on\_pushButton\_clicked();

private:

Ui::WinDialog \*ui;

};

#endif // WINDIALOG\_H  
#include "windialog.h"

#include "ui\_windialog.h"

WinDialog::WinDialog(QWidget \*parent) :

QWidget(parent),

ui(new Ui::WinDialog)

{

ui->setupUi(this);

QObject::connect(ui->pushButton, SIGNAL(clicked()), this,SLOT(on\_pushButton\_clicked()));

}

WinDialog::~*WinDialog*()

{

delete ui;

}

void WinDialog::on\_pushButton\_clicked()

{

this->close();

}

1. **Тестирование графического интерфейса.  
   Тестирование графического интерфейса происходило вручную.  
   Я несколько раз проходил игру, с целью выявления недочётов. Вставлял в клетку значения, проверял как работают методы проверки поля, сохранения и загрузки игры, правильно ли выводится поле, соответствует ли оно математической модели, как срабатывают игровые уровни.  
   В целом проверка графического интерфейса пройдена. Нарушений не выявлено.**