COLEGIUL NAȚIONAL
GHEORGHE ȘINCAI
BAIA MARE

Nr. / /

LUCRARE PENTRU OBȚINEREA ATESTATULUI PROFESIONAL LA INFORMATICĂ

SUDOKU

PROFESOR ÎNDRUMĂTOR:
Prof. Kalmar Violeta

ELEV:
Tiut Cristian
Clasa a XII-a D

-Baia Mare-
2023

COLEGIUL NAȚIONAL
GHEORGHE ȘINCAI
BAIA MARE

LUCRARE PENTRU OBȚINEREA ATESTATULUI PROFESIONAL LA INFORMATICĂ

“Sudoku”

-joc de sudoku-

PROFESOR ÎNDRUMĂTOR:
Prof. Kalmar Violeta

ELEV
Tiut Cristian
Clasa a XII-a D

-Baia Mare-
2023

Cuprins

I. Introducere

a. Limbaj de programare și framework

Limbajul C++ a fost inventat de către Bjarne Stroustrup în 1979, ca o extindere a limbajului C. Acesta a adus o serie de îmbunătățiri ale limbajului C, printre care: adăugarea noțiunii de clase, de funcții virtuale, suprascrierea operatorilor, moștenirea multiplă (multiple ingeritance) și șabloanele (templates).

Motivul principal pentru care am ales utilizarea acestui limbaj de programare este faptul că suportă multiple tipuri de programare, printre care: funcțională, procedurală și orientată pe obiect.

Haavard Nord și Eirik Chamge-Eng (primii programatori ai Qt, respectiv directorul executiv și președintele Trolltech) au început dezvoltarea Qt în 1991. Qt este un framework C++ folosit atât pentru crearea programelor cu interfață grafică, cât și pentru programe fără interfață grafică, cum sunt programele care rulează pe servere (backend).

Cele mai cunoscute utilizări ale Qt sunt KDE, browserul web Opera, Google Earth, Skype sau Qtopia. Qt este produs de firma norvegiană Trolltech.

Motivul pentru care am ales utilizarea acestui framework este ușurința realizării interfeței grafice pentru aplicație și intuitivitatea utilizării diferitelor obiecte și funcții prestabilite.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

b. Descrierea jocului

Sudoku este un joc ce consta în completarea unei grile de 9x9 celule cu numerele de la 1-9 (unele numere fiind completate de la început pentru ca grila să aibă soluție unică), astfel încât acestea să nu se repete pe coloana, pe linie sau în interiorul unuia dintre cele 9 careuri determinate de grupări de 3x3 celule ca în figura alăturată. Sudoku a devenit popular în întreaga lume ca un joc de puzzle și este cunoscut pentru faptul că dezvoltă abilități de gândire critică și rezolvare de probleme.

c. Motiv de alegere a temei

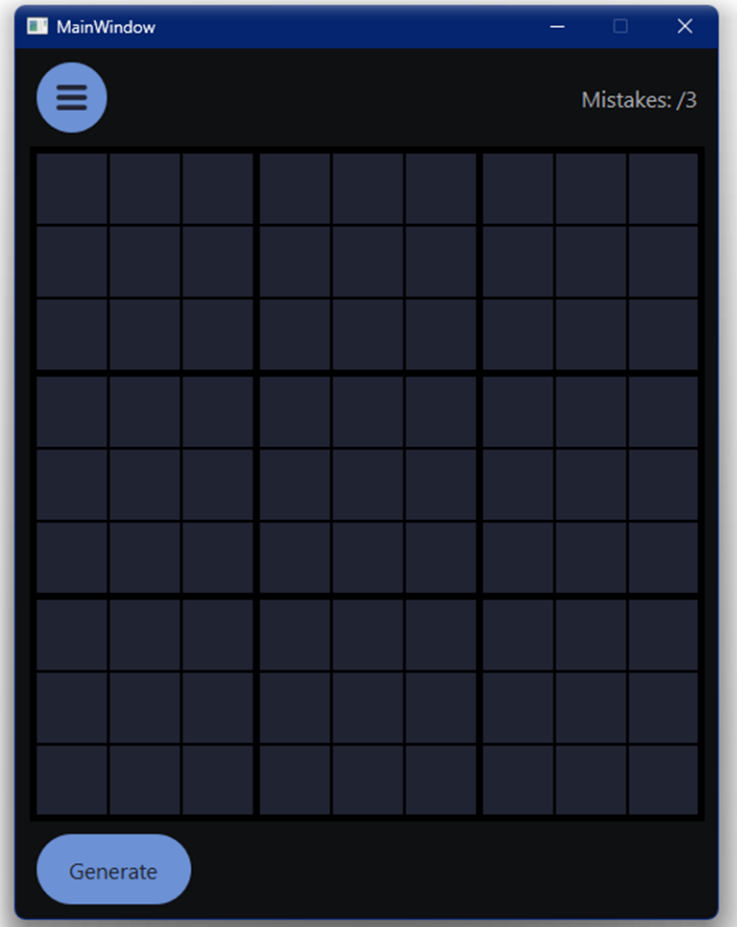
Deși este, aparent, un joc banal, Sudoku devine din ce în ce mai complicat cu cât numărul elementelor date de la început este mai mic. În acest sens, rezolvitorii trebuie să apeleze la anumite tactici și strategii astfel încat să ajungă să completeze grila.

Am făcut inițial un program care, fiindu-i dată o grilă incompletă de Sudoku, o rezolvă dacă este posibil și afișează rezultatul. Apoi mi s-a părut interesant modul în care grilele au numerele si celulele goale aleatorii și, totuși, au o singură soluție, motiv pentru care mi-am propus să creez un generator de Sudoku.

Complexitatea acestui joc, ascunsă de banalitatea aparentă a sa reprezintă motivul principal ce m-a determinat să realizez acest proiect.

II. Prezentarea proiectului

a. Privire de ansamblu

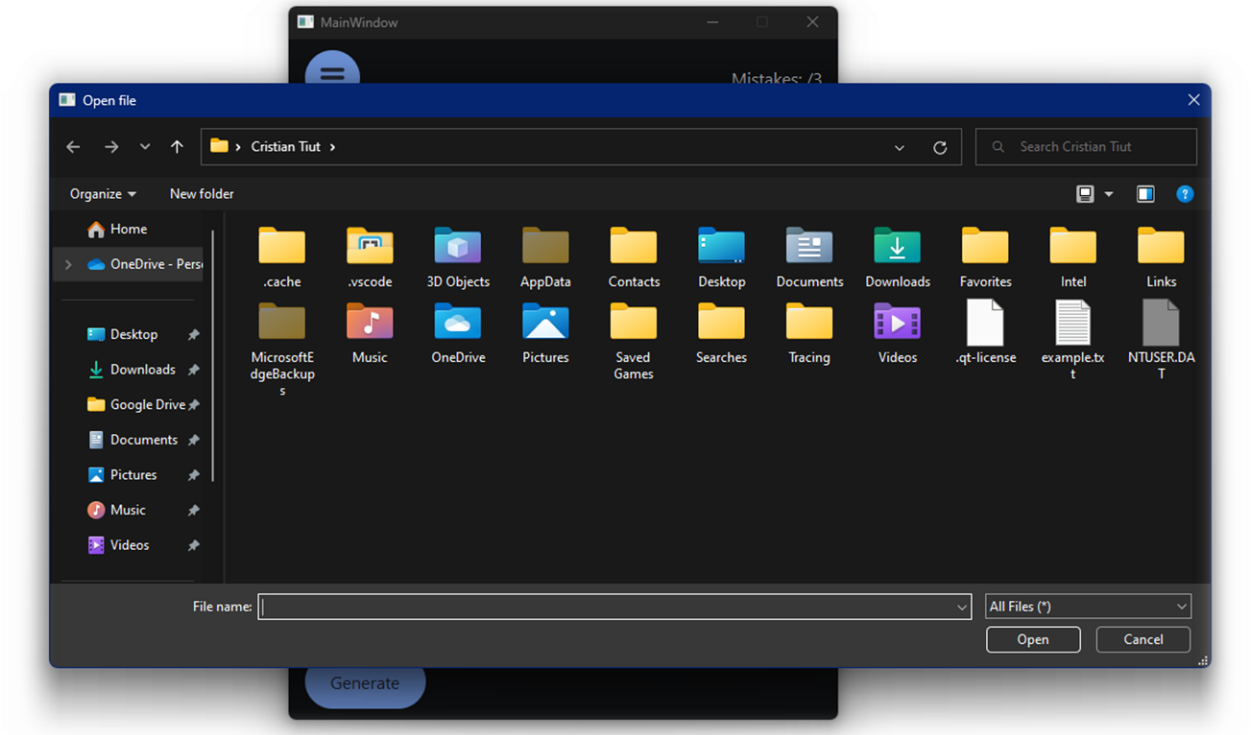


La deschiderea aplicației, suntem întâmpinați de ecranul de start, care cuprinde grila de sudoku, butonul de deschidere a meniului de unelte, butonul de deschidere a meniului „Generate” și eticheta de text care indică numărul de greșeli curent (inițial indicând doar numărul de greșeli după care se încheie un joc).

b. Meniul de unelte

Meniul de unelte este accesat printr-un click pe butonul situat in partea superioara a grilei. Astfel, apar 5 noi butoane: „deschidere”, „salvare”, „resetare”, „închidere”, respectiv „ajutor”.

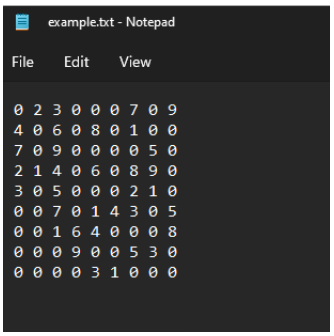
- Deschidere

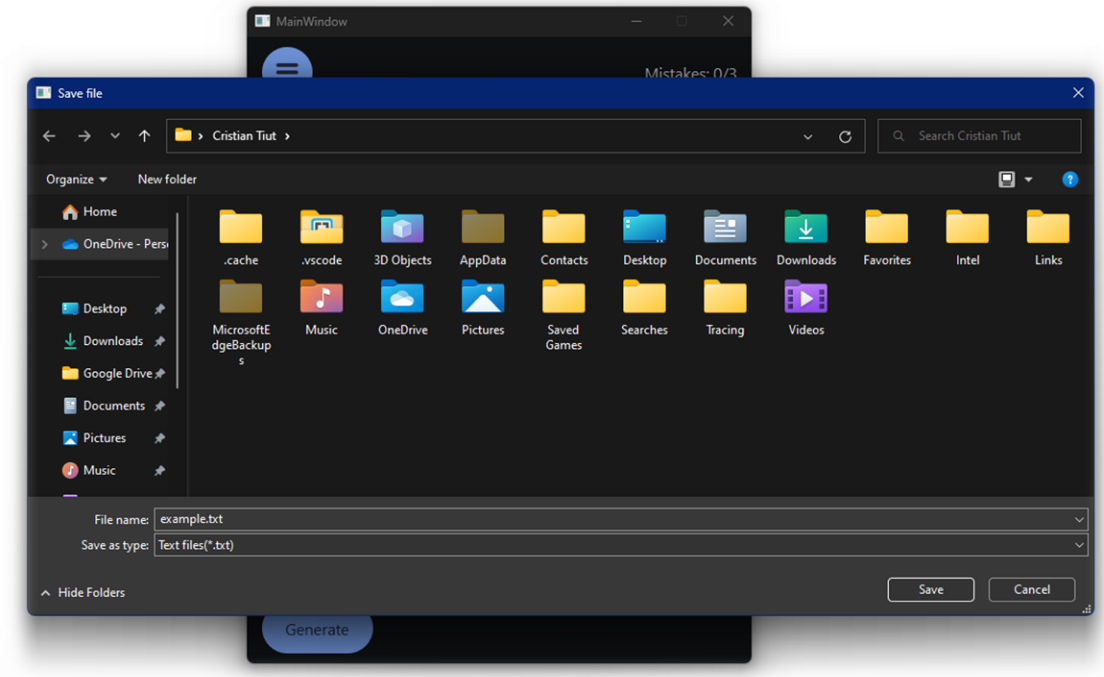


Butonul „deschidere” este primul din meniul de unelte, iar apăsarea lui determină deschiderea unei ferestre „dialog” de unde utilizatorul poate alege un fisier de tip text ce conține datele unei grile de sudoku în forma prezentată la următorul punct.

- Salvare

Butonul „salvare” este al doilea din meniul de unelte, iar apăsarea lui determină deschiderea unei ferestre „dialog” unde utilizatorul poate alege un folder unde să fie creat un fișier text nou. Acesta conține numerele din grila de sudoku din momentul apăsării butonului, astfel: pe fiecare linie din fișier apar numerele de pe linia respectivă din grilă, separate printr-un spațiu (celulele necompletate din grilă sunt marcate cu 0 în fișier).





- Resetare

Butonul „resetare” este al treilea din meniul de unelte, iar apăsarea lui determină ștergerea tuturor numerelor din celulele grilei, aducând astfel grila în starea inițială.

- Închidere

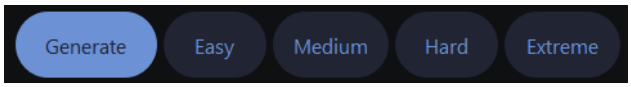
Butonul „închidere” este al patrulea din meniul de unelte, iar apăsarea lui determină închiderea aplicației fără a salva grila în lucru.

- Ajutor

Butonul „ajutor” este al cincilea din meniul de unelte, iar apăsarea lui determină dechiderea unei ferestre noi unde apar regulile jocului și modul de utilizare a aplicației.

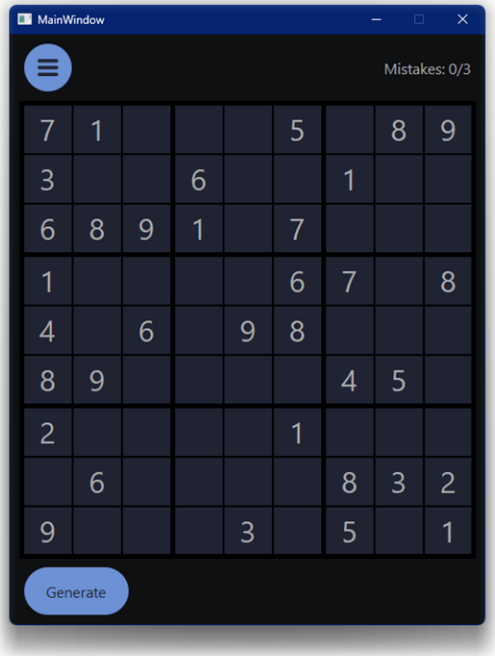
c. Meniul “Generate”

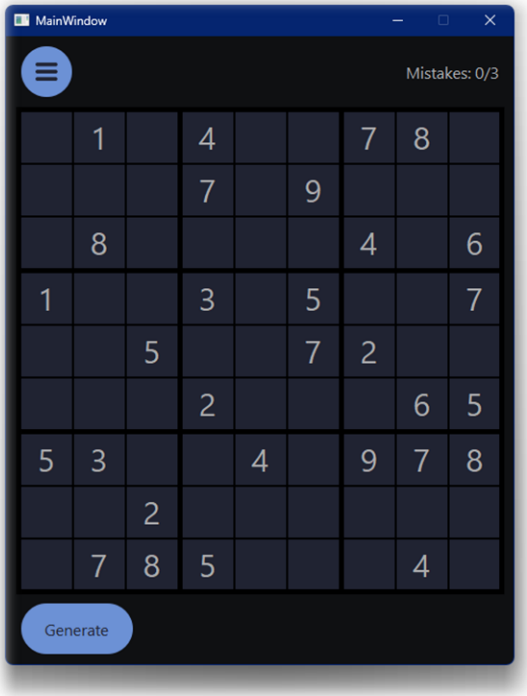
Meniul „Generate” este accesat printr-un click pe butonul situat in partea inferioară a grilei. Astfel, apar 4 noi butoane: „Easy”, „Medium”, „Hard”, respectiv „Extreme”.



- Easy

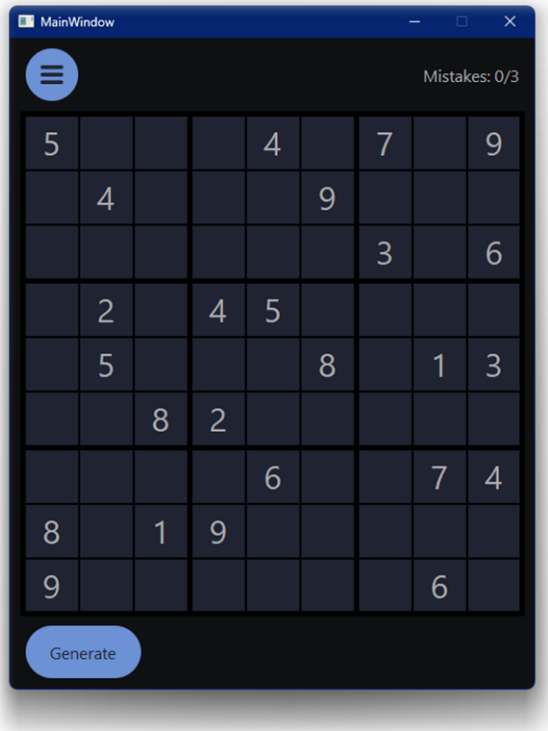
Butonul „Easy” este primul din meniul „Generate”, iar apăsarea lui determină generarea unei grile de sudoku aleatorii cu 35 de numere completate și fără linii, coloane sau careuri complete.





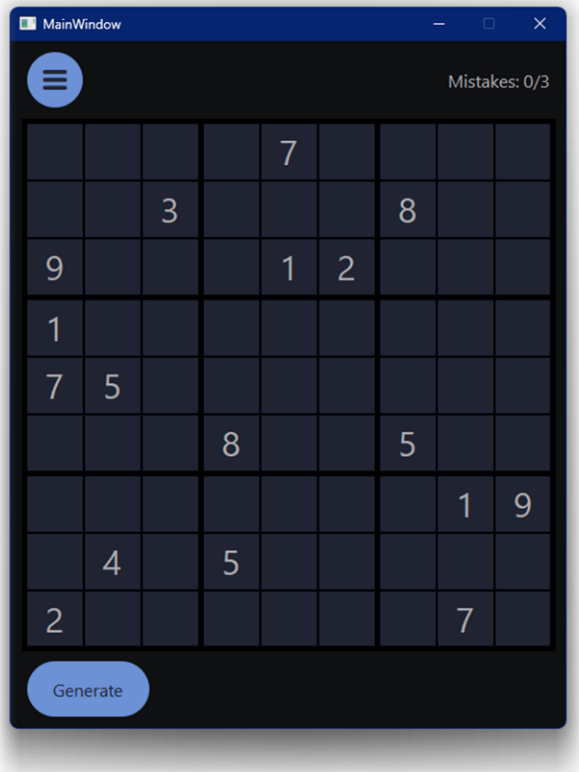
- Medium

Butonul „Medium” este al doilea din meniul „Generate”, iar apăsarea lui determină generarea unei grile de sudoku aleatorii cu 30 de numere completate și fără linii, coloane sau careuri complete.



· Hard

Butonul „Hard” este al treilea din meniul „Generate”, iar apăsarea lui determină generarea unei grile de sudoku aleatorii cu 25 de numere completate și fără linii, coloane sau careuri complete.



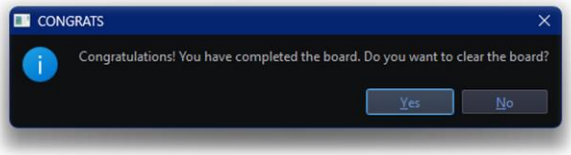
· Extreme

Butonul „Extreme” este al patrulea din meniul „Generate”, iar apăsarea lui determină generarea unei grile de sudoku aleatorii cu 17 numere completate (numărul minim de celule completate pentru care grila poate avea soluție unică) și fără linii, coloane sau careuri complete.

d. Casete de mesaje

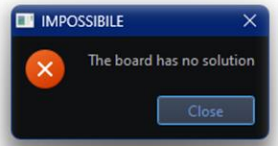
- „Completed”

Această casetă apare în momentul în care grila este completată, iar utilizatorului i se pune la dispoziție opțiunea de a șterge grila, ajungând în starea inițială.



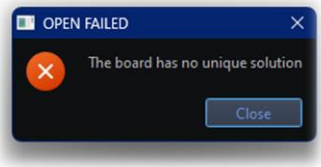
- „Impossible”

Această casetă apare în momentul în care utilizatorul intenționează să deschidă un fișier conținând o grilă care nu are soluție.



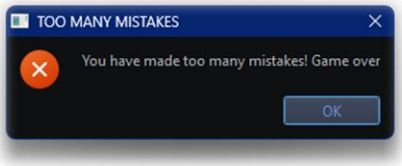
- „Open failed”

Această casetă apare în momentul în care utilizatorul intenționează să deschidă un fișier conținând o grilă care nu are soluție unică sau un fișier care nu are formatul de la punctul „Salvare” din descrierea meniului de unelte.



- „Too many mistakes”

Această casetă apare în momentul în care utilizatorul a făcut cea de a III-a greșeală în timpul unui joc Afișarea casetei este precedată de ștergerea grilei și revenirea ei la starea inițială.



III. Codul sursă

board.h

```
#ifndef BOARD_H_INCLUDED
#define BOARD_H_INCLUDED

class Board {
    private:
        int val[9][9];
        int bitmask_line[9];
        int bitmask_col[9];
        int bitmask_square[9];
        int nr_zero;

    public:
        Board();
        void set(int l, int c, int v);
        int get(int l, int c) { return val[l][c]; }
        int get_posib(int l, int c) { return (1<<9) - 1 - (bitmask_line[l] | bitmask_col[c] | bitmask_square[l/3*3 + c/3]); }
        bool is_posib(int l, int c, int v) { return ((bitmask_line[l] | bitmask_col[c] | bitmask_square[l/3*3 + c/3]) & nr2bit(v)) == 0;  }
        inline int get_nr_posib(int l, int c);
        bool is_complete() { return nr_zero == 0; }
        static int nr2bit(int nr) { return 1 << (nr - 1); }
        static int bit2nr(int bit);
        void swap(Board &other);
        int get_nr_zero(){ return nr_zero; }
        bool no_complete_lcs();
};

#endif // BOARD_H_INCLUDED
```

board.cpp

```
#include "board.h"
#include <utility>

Board::Board() : nr_zero(9*9)
{
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
            val[i][j] = 0;
    for (int i = 0; i < 9; i++)
        bitmask_line[i] = bitmask_col[i] = bitmask_square[i] = 0;
}

void Board::set(int l, int c, int v)
{
    int old = val[l][c];

    if (old == v)
        return;

    if (old != 0)
    {
        val[l][c] = 0;
        int oldbit = nr2bit(old);

        bitmask_line[l] -= oldbit;
        bitmask_col[c] -= oldbit;
        bitmask_square[l/3*3 + c/3] -= oldbit;

        nr_zero++;
    }

    if (v != 0)
    {
        nr_zero--;

        int vbit = nr2bit(v);
        val[l][c] = v;

        bitmask_line[l] += vbit;
        bitmask_col[c] += vbit;
        bitmask_square[l/3*3 + c/3] += vbit;
    }
}

int Board::get_nr_posib(int l, int c)
{
    int nr = 0, p = get_posib(l,c);
    while (p != 0)
    {
        nr++;
        p &= (p-1);
    }
    return nr;
}

int Board::bit2nr(int bit)
{
    for(int nr = 1; nr <= 9; nr++)
        if(bit & (1 << (nr-1)))
            return nr;
    return 0;
}
```

```
void Board::swap(Board &other)
{
    for(int l = 0; l < 9; l++)
        for(int c = 0; c < 9; c++)
            std::swap(val[l][c], other.val[l][c]);

    std::swap(nr_zero, other.nr_zero);

    for(int i = 0; i < 9; i++)
    {
        std::swap(bitmask_line[i], other.bitmask_line[i]);
        std::swap(bitmask_col[i], other.bitmask_col[i]);
        std::swap(bitmask_square[i], other.bitmask_square[i]);
    }
}

bool Board::no_complete_lcs()
{
    int val = (1<<9) - 1;
    for(int i = 0; i < 9; i++)
        if(bitmask_col[i] == val || bitmask_line[i] == val || bitmask_square[i] == val)
            return false;
    return true;
}
```

generator.h

```
#ifndef GENERATOR_H
#define GENERATOR_H

#include "board.h"
#include <vector>

Board generate(int nr_filled);
Board generate_extreme();

#endif // GENERATOR_H
```

generator.cpp

```
#include "generator.h"
#include "solver.h"
#include <algorithm>
#include <vector>

using std::vector;

Board generate_completed_board()
{
    Board board;
    solve_random(board);
    return board;
}

std::vector<std::pair<int, int>> get_non_empty_positions(Board &board)
{
    std::vector<std::pair<int, int>> result;
    result.reserve(81);
    for(int l = 0; l < 9; l++)
        for(int c = 0; c < 9; c++)
            if(board.get(l, c))
                result.push_back({l, c});
    return result;
}

std::vector<std::pair<int, int>> get_positions()
{
    std::vector<std::pair<int, int>> result;
    result.reserve(81);
    for(int l = 0; l < 9; l++)
        for(int c = 0; c < 9; c++)
            result.push_back({l, c});
    return result;
}

Board generate_solveable_board(int nr_filled)
{
    while(true)
    {
        Board board = generate_completed_board();
        auto pos = get_positions();
        std::random_shuffle(pos.begin(), pos.end());

        int cnt = 0;
        for(int i = 0; i < 81 && cnt < 81-nr_filled; i++)
        {
            Board copy = board;
            copy.set(pos[i].first, pos[i].second, 0);
            auto result = solve(copy);
            if(result == REZOLVAT)
            {
                board.set(pos[i].first, pos[i].second, 0);
                cnt++;
            }
        }
        if(cnt == 81 - nr_filled)
            return board;
    }
}
```

```
Board generate(int nr_filled)
{
    while(true)
    {
        Board board = generate_solveable_board(nr_filled);
        if(board.no_complete_lcs())
            return board;
    }
}

vector<vector<int>> fill_extreme_board()
{
    vector<vector<int>> extreme_board = {
        {9, 0, 0,   0, 0, 0,   0, 6, 0},
        {0, 2, 0,   7, 0, 0,   0, 0, 0},
        {0, 0, 0,   0, 0, 0,   0, 5, 4},

        {0, 0, 0,   3, 0, 0,   7, 0, 0},
        {6, 7, 0,   0, 0, 0,   0, 0, 0},
        {5, 0, 0,   0, 0, 0,   0, 0, 0},

        {4, 0, 0,   0, 5, 9,   0, 0, 0},
        {0, 0, 8,   0, 0, 0,   3, 0, 0},
        {0, 0, 0,   0, 6, 0,   0, 0, 0}
    };
    return extreme_board;
}

void apply_random_permutation(vector<vector<int>> &v)
{
    int nr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    std::random_shuffle(nr+1, nr+10);
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            v[i][j] = nr[ v[i][j] ];
}

void apply_90_left_rotation(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[j][8-i];
    v.swap(board);
}

void apply_90_right_rotation(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[8-j][i];
    v.swap(board);
}

void apply_180_rotation(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[8-i][8-j];
    v.swap(board);
}

void apply_horizontal_flip(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[8-i][j];
    v.swap(board);
}

void apply_vertical_flip(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[i][8-j];
    v.swap(board);
}

void apply_main_diagonal_flip(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[j][i];
    v.swap(board);
}

void apply_secondary_diagonal_flip(vector<vector<int>> &v)
{
    vector<vector<int>> board(9, vector<int>(9));
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board[i][j] = v[8-j][8-i];
    v.swap(board);
}

Board generate_extreme()
{
    vector<vector<int>> extreme_board = fill_extreme_board();
    apply_random_permutation(extreme_board);
    int var = std::rand() % 8;

    switch (var) {
```

```
    case 0:
        apply_90_left_rotation(extreme_board);
        break;
    case 1:
        apply_90_right_rotation(extreme_board);
        break;
    case 2:
        apply_180_rotation(extreme_board);
        break;
    case 3:
        apply_horizontal_flip(extreme_board);
        break;
    case 4:
        apply_vertical_flip(extreme_board);
        break;
    case 5:
        apply_main_diagonal_flip(extreme_board);
        break;
    case 6:
        apply_secondary_diagonal_flip(extreme_board);
        break;
    default:
        break;
    }

    Board board;
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board.set(i, j, extreme_board[i][j]);
    return board;
}
```

help.h

```
#ifndef HELP_H
#define HELP_H

#include <QDialog>

namespace Ui {
class Help;
}

class Help : public QDialog
{
    Q_OBJECT

public:
    explicit Help(QWidget *parent = nullptr);
    ~Help();

private:
    Ui::Help *ui;
};

#endif // HELP_H
```

help.cpp

```
#include "help.h"
#include "ui_help.h"
#include <QVBoxLayout>

Help::Help(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Help)
{
    ui->setupUi(this);

    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addWidget(ui->rulesTitle);
    layout->addWidget(ui->rules);
    layout->addWidget(ui->useTitle);
    layout->addWidget(ui->use);
    ui->scrollWidget->setLayout(layout);
}

Help::~Help()
{
    delete ui;
}
```

help.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Help</class>
 <widget class="QDialog" name="Help">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>780</width>
    <height>630</height>
   </rect>
  </property>
```

```
  <property name="minimumSize">
   <size>
    <width>780</width>
    <height>630</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>780</width>
    <height>630</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Help</string>
  </property>
  <widget class="QScrollArea" name="scrollArea">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>10</y>
     <width>761</width>
     <height>601</height>
    </rect>
   </property>
   <property name="widgetResizable">
    <bool>true</bool>
   </property>
   <widget class="QWidget" name="scrollWidget">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>759</width>
      <height>599</height>
     </rect>
    </property>
    <widget class="QLabel" name="rulesTitle">
     <property name="geometry">
      <rect>
       <x>10</x>
       <y>10</y>
       <width>131</width>
       <height>31</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <pointsize>16</pointsize>
       <kerning>true</kerning>
      </font>
     </property>
     <property name="text">
      <string>Sudoku rules: </string>
     </property>
     <property name="textFormat">
      <enum>Qt::AutoText</enum>
     </property>
     <property name="alignment">
      <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignTop</set>
     </property>
    </widget>
    <widget class="QLabel" name="rules">
     <property name="geometry">
      <rect>
       <x>30</x>
       <y>40</y>
       <width>711</width>
       <height>261</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <pointsize>12</pointsize>
      </font>
     </property>
     <property name="text">
      <string>Sudoku is a logic-based number placement puzzle. The objective of the puzzle is to fill a 9x9 grid with digits so that each column, each row,
and each of the nine 3x3 sub-grids (also known as boxes or regions) contains all of the digits from 1 to 9. The rules for solving a Sudoku puzzle are as
follows:

      1. Each row of the 9x9 grid must contain all of the digits from 1 to 9.
      2. Each column of the 9x9 grid must contain all of the digits from 1 to 9.
      3. Each of the nine 3x3 sub-grids of the 9x9 grid must contain all of the digits from 1 to 9.
      4. Each digit can only appear once in each row, column, and 3x3 sub-grid.
      5. Some cells in the grid may already be filled with numbers. These are called &quot;givens&quot;. The solver's goal is to fill in the remaining
cells with digits from 1 to 9 so that the above rules are satisfied.

It's important to note that a valid Sudoku puzzle has only one unique solution.

</string>
     </property>
     <property name="alignment">
      <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignTop</set>
     </property>
     <property name="wordWrap">
      <bool>true</bool>
     </property>
    </widget>
    <widget class="QLabel" name="useTitle">
     <property name="geometry">
      <rect>
       <x>10</x>
       <y>310</y>
       <width>191</width>
       <height>31</height>
      </rect>
     </property>
```

```
     <property name="font">
      <font>
       <pointsize>16</pointsize>
       <kerning>true</kerning>
      </font>
     </property>
     <property name="text">
      <string>How to use the app:</string>
     </property>
     <property name="textFormat">
      <enum>Qt::AutoText</enum>
     </property>
     <property name="alignment">
      <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignTop</set>
     </property>
    </widget>
    <widget class="QLabel" name="use">
     <property name="geometry">
      <rect>
       <x>30</x>
       <y>340</y>
       <width>711</width>
       <height>491</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <pointsize>12</pointsize>
      </font>
     </property>
     <property name="text">
      <string>Opening the app, you are shown the main window, which consists of: the tools menu, the board, the
mistakes counter and the &amp;quot;Generate&amp;quot; menu.

To start playing you have to either click on &amp;quot;Generate&amp;quot; and select the type of game you want to
play or click on the tools menu button and then click on the first button (open button) which opens
up a Windows File Dialog where you can search for the file you want to open. This file should be a
text file that contains of matrix of numbers from 0 to 9 displyed in a sudoku grid: 9 lines, each
of them containing 9 numbers with a space in between. If a number is 0, the board will leave that
cell empty, otherwise it will put the number in the cell and disable the editing of that cell.

In each game you are allowed to make 2 mistakes and when you make the third one, the board will be
cleared and you will be shown a message box telling you that you have made too many mistakes.
The numbers you have entered correctly will be displayed with a light blue color and the wrong ones
with a red one.

Once you have completed the board, you are shown a message box asking you whether you want the
board to be cleared or not.

If you want to quit and save the game you were working at click the second button in the tools menu
(save button) and it will open up a Windows Save File Dialog where you can select the path where
you want to save the file and its name and it will generate a text file with the format mentioned
above.

The third button in the tools menu is the clear button and it clears the board, bringing it to the
initial state.

The fourth button in the tools menu is the quit button.

The fifth in the tools menu is the help button, which opens up this window.</string>
     </property>
     <property name="alignment">
      <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignTop</set>
     </property>
    </widget>
   </widget>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

main.cpp

```
#include "mainwindow.h"
#include <cstdlib>
#include <ctime>
#include <QApplication>
#include <QStyleFactory>
#include <QFile>

void setPalette(QApplication &a)
{
    a.setStyle(QStyleFactory::create("Fusion"));
    QPalette p = qApp->palette();
    p.setColor(QPalette::Window, QColor(15,16,18));
    p.setColor(QPalette::Button, QColor(33,36,51));
    p.setColor(QPalette::ButtonText, QColor(107,146,215));
    p.setColor(QPalette::WindowText, QColor(177,178,183));
    a.setPalette(p);
}

int main(int argc, char *argv[])
{
    srand(time(NULL));

    QApplication a(argc, argv);
    setPalette(a);

    MainWindow w;
    w.show();
    return a.exec();
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <vector>
#include "qtextedit.h"
#include <QMainWindow>
#include "board.h"
#include "ui_mainwindow.h"

using namespace std;

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    //variables
    Ui::MainWindow *ui;
    vector<vector<QTextEdit *>> board_inputs;
    vector<vector<int>> board_numbers;
    Board solution;
    int mistakes = 0;
    bool menuButtonSelected = 0;
    bool generateButtonSelected = 0;

    //methods
    void update_ui_board(Board &board);
    void update_mistakes_label(){ ui->mistakesLabel->setText(("Mistakes: " + to_string(mistakes) + "/3").c_str()); }
    void expand_tools_menu();
    void shrink_tools_menu();
    void expand_generate_menu();
    void shrink_generate_menu();
    void connect_components();

    //style methods
    void solved_style();
    void disabled_style();
    void initial_style();
    void board_style();
    void tools_menu_style();
    void generate_menu_style();

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    //methods
    void increase_mistakes();
    void is_complete();

public slots:
    //tools menu slots
    void toolsMenuPressed();
    void openPressed();
    void savePressed();
    void resetPressed();
    void quitPressed();

    //generate menu slots
    void generatePressed();
    void easyPressed();
    void mediumPressed();
    void hardPressed();
    void extremePressed();
private slots:
    void on_helpButton_clicked();
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "qtextedit.h"
#include "textchangedhandler.h"
#include "ui_mainwindow.h"
#include "generator.h"
#include "solver.h"
#include <iostream>
#include <fstream>
#include <cctype>
#include <vector>
#include <QFont>
#include <QMessageBox>
#include <QFileDialog>
#include <QPropertyAnimation>
#include "help.h"


// ------------------------------------- private methods -------------------------------------

void MainWindow::update_ui_board(Board &board)
{
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
```

```
        {
            board_numbers[i][j] = board.get(i, j);
            std::string str = board_numbers[i][j] == 0 ? "" : std::to_string(board_numbers[i][j]);
            board_inputs[i][j]->setText(str.c_str());
            TextChangedHandler::alignCenter(board_inputs[i][j]);
        }
}

void MainWindow::expand_tools_menu()
{
    auto anim1 = new QPropertyAnimation(ui->openButton, "geometry");
    anim1->setDuration(300);
    anim1->setStartValue(ui->openButton->geometry());

    auto anim2 = new QPropertyAnimation(ui->saveButton, "geometry");
    anim2->setDuration(300);
    anim2->setStartValue(ui->saveButton->geometry());

    auto anim3 = new QPropertyAnimation(ui->resetButton, "geometry");
    anim3->setDuration(300);
    anim3->setStartValue(ui->resetButton->geometry());

    auto anim4 = new QPropertyAnimation(ui->quitButton, "geometry");
    anim4->setDuration(300);
    anim4->setStartValue(ui->quitButton->geometry());

    auto anim5 = new QPropertyAnimation(ui->helpButton, "geometry");
    anim5->setDuration(300);
    anim5->setStartValue(ui->helpButton->geometry());

    menuButtonSelected = 1;
    anim1->setEndValue(QRect(70, 10, 50, 50));
    anim2->setEndValue(QRect(125, 10, 50, 50));
    anim3->setEndValue(QRect(180, 10, 50, 50));
    anim4->setEndValue(QRect(235, 10, 50, 50));
    anim5->setEndValue(QRect(290, 10, 50, 50));

    anim1->start();
    anim2->start();
    anim3->start();
    anim4->start();
    anim5->start();
}

void MainWindow::shrink_tools_menu()
{
    auto anim1 = new QPropertyAnimation(ui->openButton, "geometry");
    anim1->setDuration(300);
    anim1->setStartValue(ui->openButton->geometry());

    auto anim2 = new QPropertyAnimation(ui->saveButton, "geometry");
    anim2->setDuration(300);
    anim2->setStartValue(ui->saveButton->geometry());

    auto anim3 = new QPropertyAnimation(ui->resetButton, "geometry");
    anim3->setDuration(300);
    anim3->setStartValue(ui->resetButton->geometry());

    auto anim4 = new QPropertyAnimation(ui->quitButton, "geometry");
    anim4->setDuration(300);
    anim4->setStartValue(ui->quitButton->geometry());

    auto anim5 = new QPropertyAnimation(ui->helpButton, "geometry");
    anim5->setDuration(300);
    anim5->setStartValue(ui->helpButton->geometry());

    menuButtonSelected = 0;
    anim1->setEndValue(QRect(15, 10, 50, 50));
    anim2->setEndValue(QRect(15, 10, 50, 50));
    anim3->setEndValue(QRect(15, 10, 50, 50));
    anim4->setEndValue(QRect(15, 10, 50, 50));
    anim5->setEndValue(QRect(15, 10, 50, 50));

    anim1->start();
    anim2->start();
    anim3->start();
    anim4->start();
    anim5->start();
}

void MainWindow::expand_generate_menu()
{
    auto anim1 = new QPropertyAnimation(ui->easyButton, "geometry");
    anim1->setDuration(300);
    anim1->setStartValue(ui->easyButton->geometry());

    auto anim2 = new QPropertyAnimation(ui->mediumButton, "geometry");
    anim2->setDuration(300);
    anim2->setStartValue(ui->mediumButton->geometry());

    auto anim3 = new QPropertyAnimation(ui->hardButton, "geometry");
    anim3->setDuration(300);
    anim3->setStartValue(ui->hardButton->geometry());

    auto anim4 = new QPropertyAnimation(ui->extremeButton, "geometry");
    anim4->setDuration(300);
    anim4->setStartValue(ui->extremeButton->geometry());

    generateButtonSelected = 1;
    anim1->setEndValue(QRect(130, 560, 80, 50));
    anim2->setEndValue(QRect(215, 560, 90, 50));
    anim3->setEndValue(QRect(310, 560, 80, 50));
    anim4->setEndValue(QRect(395, 560, 90, 50));

    anim1->start();
    anim2->start();
    anim3->start();
```

```
    anim4->start();
}

void MainWindow::shrink_generate_menu()
{
    auto anim1 = new QPropertyAnimation(ui->easyButton, "geometry");
    anim1->setDuration(300);
    anim1->setStartValue(ui->easyButton->geometry());

    auto anim2 = new QPropertyAnimation(ui->mediumButton, "geometry");
    anim2->setDuration(300);
    anim2->setStartValue(ui->mediumButton->geometry());

    auto anim3 = new QPropertyAnimation(ui->hardButton, "geometry");
    anim3->setDuration(300);
    anim3->setStartValue(ui->hardButton->geometry());

    auto anim4 = new QPropertyAnimation(ui->extremeButton, "geometry");
    anim4->setDuration(300);
    anim4->setStartValue(ui->extremeButton->geometry());

    generateButtonSelected = 0;
    anim1->setEndValue(QRect(15, 560, 80, 50));
    anim2->setEndValue(QRect(15, 560, 90, 50));
    anim3->setEndValue(QRect(15, 560, 80, 50));
    anim4->setEndValue(QRect(15, 560, 90, 50));

    anim1->start();
    anim2->start();
    anim3->start();
    anim4->start();
}

void MainWindow::connect_components()
{
    for(int l = 0; l < 9; l++)
        for(int c = 0; c < 9; c++)
        {
            TextChangedHandler *handler = new TextChangedHandler(board_inputs[l][c], board_numbers, l, c,
                                                                 solution, this);
            connect(board_inputs[l][c], SIGNAL(textChanged()), handler, SLOT(textEditChanged()));
        }

    connect(ui->toolsMenuButton, SIGNAL( pressed() ), this, SLOT(toolsMenuPressed()));
    connect(ui->openButton, SIGNAL( pressed() ), this, SLOT(openPressed()));
    connect(ui->saveButton, SIGNAL ( pressed() ), this, SLOT(savePressed()));
    connect(ui->resetButton, SIGNAL( pressed() ), this, SLOT(resetPressed()));
    connect(ui->quitButton, SIGNAL ( pressed() ), this, SLOT(quitPressed()));

    connect(ui->generateButton, SIGNAL( pressed() ), this, SLOT(generatePressed()));
    connect(ui->easyButton, SIGNAL( pressed() ), this, SLOT(easyPressed()));
    connect(ui->mediumButton, SIGNAL ( pressed() ), this, SLOT(mediumPressed()));
    connect(ui->hardButton, SIGNAL( pressed() ), this, SLOT(hardPressed()));
    connect(ui->extremeButton, SIGNAL ( pressed() ), this, SLOT(extremePressed()));
}


// ---------------------------------- private style methods ----------------------------------

void MainWindow::solved_style()
{
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
        {
            board_inputs[i][j]->setStyleSheet("border: no border;"
                                              "background-color: #212433;"
                                              "color: #ACADB1;");
            board_inputs[i][j]->setReadOnly(true);
            board_inputs[i][j]->setDisabled(true);
        }
}

void MainWindow::disabled_style()
{

    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
        {
            board_inputs[i][j]->setReadOnly(true);
            board_inputs[i][j]->setDisabled(true);
            board_inputs[i][j]->setStyleSheet("background-color: #202332;"
                                              "border: no border;");
        }
}

void MainWindow::initial_style()
{
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            if(board_numbers[i][j] == 0)
            {
                board_inputs[i][j]->setStyleSheet("background-color: #202332;"
                                                  "border: no border;");
                board_inputs[i][j]->setReadOnly(false);
                board_inputs[i][j]->setDisabled(false);
            }
            else
            {
                board_inputs[i][j]->setStyleSheet("color: #ADADAF;"
                                                  "background-color: #202332;"
                                                  "border: no border;");
                board_inputs[i][j]->setReadOnly(true);
                board_inputs[i][j]->setDisabled(true);
            }
}

void MainWindow::board_style()
```

```
{
    ui->boardWidget->setStyleSheet("background-color: #000000;"); //darkgray
    board_inputs.resize(9, vector<QTextEdit *>(9));
    board_numbers.resize(9, vector<int> (9, 0));

    ui->board->setSpacing(5);

    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            auto grid = new QGridLayout(this);
            ui->board->addLayout(grid, i, j);
            grid->setSpacing(2);
            for(int k = 0; k < 3; k++)
            {
                for(int l = 0; l < 3; l++)
                {
                    auto edit = new QTextEdit();
                    grid->addWidget(edit, k, l);
                    edit->setAlignment(Qt::AlignHCenter);
                    edit->setFont(QFont("Segoe UI", 23));
                    edit->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
                    edit->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
                    board_inputs[i*3+k][j*3+l] = edit;
                }
            }
        }
    }
}

void MainWindow::tools_menu_style()
{
    auto cursor = new QCursor;
    cursor->setShape(Qt::PointingHandCursor);

    ui->toolsMenuButton->setCursor(*cursor);
    ui->openButton->setCursor(*cursor);
    ui->saveButton->setCursor(*cursor);
    ui->resetButton->setCursor(*cursor);
    ui->quitButton->setCursor(*cursor);
    ui->helpButton->setCursor(*cursor);

    ui->toolsMenuButton->setIcon(QIcon(":/icons/menuicon.png"));
    ui->openButton->setIcon(QIcon(":/icons/openicon.png"));
    ui->saveButton->setIcon(QIcon(":/icons/saveicon.png"));
    ui->resetButton->setIcon(QIcon(":/icons/reseticon.png"));
    ui->quitButton->setIcon(QIcon(":/icons/quiticon.png"));
}

void MainWindow::generate_menu_style()
{
    auto cursor = new QCursor;
    cursor->setShape(Qt::PointingHandCursor);
    ui->generateButton->setCursor(*cursor);
    ui->easyButton->setCursor(*cursor);
    ui->mediumButton->setCursor(*cursor);
    ui->hardButton->setCursor(*cursor);
    ui->extremeButton->setCursor(*cursor);
}

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    board_style();
    disabled_style();
    tools_menu_style();
    generate_menu_style();
    ui->mistakesLabel->setText("Mistakes: /3");
    connect_components();
}

MainWindow::~MainWindow()
{
    delete ui;
}


// ------------------------------------- public methods -------------------------------------

void MainWindow::increase_mistakes()
{
    mistakes++;
    update_mistakes_label();
    if(mistakes >= 3)
    {
        resetPressed();
        QMessageBox box(QMessageBox::Icon::Critical, "TOO MANY MISTAKES", "You have made too many mistakes! Game over", QMessageBox::Ok, this);
        box.exec();
    }
}

void MainWindow::is_complete()
{
    QMessageBox box(QMessageBox::Icon::Information, "CONGRATS", "Congratulations! You have completed the board. Do you want to clear the board?",
QMessageBox::Yes | QMessageBox::No, this);
    if(box.exec() == QMessageBox::Yes)
        resetPressed();
}


// ------------------------------------- SLOTS -------------------------------------


// ------------------------------------- tools menu -------------------------------------
```

```
void MainWindow::toolsMenuPressed()
{
    shrink_generate_menu();
    if(!menuButtonSelected)
        expand_tools_menu();
    else
        shrink_tools_menu();
}

void MainWindow::openPressed()
{
    shrink_tools_menu();
    QFileDialog dialog(this, "Open file", QDir::homePath());
    if(dialog.exec())
    {
        auto files = dialog.selectedFiles();
        if(files.count() != 1)
            return;
        std::ifstream fin(files[0].toUtf8());

        Board board;
        bool ok = true;
        for(int i = 0; i < 9; i++)
            for(int j = 0; j < 9; j++)
            {
                int nr;
                fin >> nr;
                if(board.is_posib(i, j, nr))
                    board.set(i, j, nr);
                else
                    ok = false;
            }
        if(ok)
        {
            Board copy;
            copy = board;
            auto result = solve(copy);
            if(result == solver_result::IMPOSIBIL)
            {
                QMessageBox box(QMessageBox::Icon::Critical, "IMPOSSIBILE", "The board has no solution", QMessageBox::Close, this);
                box.exec();
            }
            else if(result == solver_result::NEDETERMINAT)
            {
                QMessageBox box(QMessageBox::Icon::Critical, "OPEN FAILED", "The board has no unique solution", QMessageBox::Close, this);
                box.exec();
            }
            else
            {
                solution = copy;
                update_ui_board(board);
                initial_style();
                mistakes = 0;
                update_mistakes_label();
            }
        }
        else
        {
            QMessageBox box(QMessageBox::Icon::Critical, "IMPOSSIBILE", "The board has no solution", QMessageBox::Close, this);
            box.exec();
        }

        fin.close();
    }
}

void MainWindow::savePressed()
{
    shrink_tools_menu();
    QFileDialog dialog(this, tr("Save file"), QDir::homePath(), tr("Text files(*.txt)"));
    dialog.setAcceptMode(QFileDialog::AcceptSave);
    dialog.setDefaultSuffix(".txt");

    if(dialog.exec())
    {
        auto files = dialog.selectedFiles();
        string path = files[0].toStdString();
        std::ofstream fout(path);

        for(int i = 0; i < 9; i++)
        {
            for(int j = 0; j < 9; j++)
                fout << board_numbers[i][j] << " ";
            fout << "\n";
        }

        fout.close();
    }
}

void MainWindow::resetPressed()
{
    shrink_tools_menu();
    Board board;
    update_ui_board(board);
    disabled_style();
    mistakes = 0;
    ui->mistakesLabel->setText("Mistakes: /3");
}

void MainWindow::quitPressed()
{
    QApplication::exit(0);
}
```

```
// ---------------------------------- generate menu ----------------------------------

void MainWindow::generatePressed()
{
    shrink_tools_menu();
    if(!generateButtonSelected)
        expand_generate_menu();
    else
        shrink_generate_menu();
}

void MainWindow::easyPressed()
{
    shrink_generate_menu();
    Board board;
    board = generate(35);
    solution = board;
    solve(solution);
    update_ui_board(board);
    initial_style();
    mistakes = 0;
    update_mistakes_label();
}

void MainWindow::mediumPressed()
{
    shrink_generate_menu();
    Board board;
    board = generate(30);
    solution = board;
    solve(solution);
    update_ui_board(board);
    initial_style();
    mistakes = 0;
    update_mistakes_label();
}

void MainWindow::hardPressed()
{
    shrink_generate_menu();
    Board board;
    board = generate(25);
    solution = board;
    solve(solution);
    update_ui_board(board);
    initial_style();
    mistakes = 0;
    update_mistakes_label();
}

void MainWindow::extremePressed()
{
    shrink_generate_menu();
    Board board;
    board = generate_extreme();
    solution = board;
    solve(solution);
    update_ui_board(board);
    initial_style();
    mistakes = 0;
    update_mistakes_label();
}

void MainWindow::on_helpButton_clicked()
{
    shrink_tools_menu();
    Help help;
    help.setModal(true);
    help.exec();
}
```

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>500</width>
    <height>620</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>500</width>
    <height>620</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>500</width>
    <height>620</height>
   </size>
  </property>
```

```
  <property name="windowTitle">
   <string>Sudoku</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="resetButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            padding-left: 3;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string/>
    </property>
    <property name="iconSize">
     <size>
      <width>22</width>
      <height>22</height>
     </size>
    </property>
   </widget>
   <widget class="QLabel" name="mistakesLabel">
    <property name="geometry">
     <rect>
      <x>395</x>
      <y>10</y>
      <width>91</width>
      <height>51</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="text">
     <string>Mistakes: 0/3</string>
    </property>
    <property name="alignment">
     <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
    </property>
   </widget>
   <widget class="QWidget" name="boardWidget" native="true">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>70</y>
      <width>481</width>
      <height>481</height>
     </rect>
    </property>
    <property name="palette">
     <palette>
      <active>
       <colorrole role="WindowText">
        <brush brushstyle="SolidPattern">
         <color alpha="255">
          <red>0</red>
          <green>0</green>
          <blue>0</blue>
         </color>
        </brush>
       </colorrole>
       <colorrole role="Button">
        <brush brushstyle="SolidPattern">
         <color alpha="255">
          <red>255</red>
          <green>255</green>
          <blue>255</blue>
         </color>
        </brush>
       </colorrole>
       <colorrole role="Light">
        <brush brushstyle="SolidPattern">
         <color alpha="255">
          <red>255</red>
          <green>255</green>
          <blue>255</blue>
         </color>
        </brush>
       </colorrole>
       <colorrole role="Midlight">
        <brush brushstyle="SolidPattern">
         <color alpha="255">
          <red>255</red>
```

```
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Dark">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>127</red>
      <green>127</green>
      <blue>127</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Mid">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>170</red>
      <green>170</green>
      <blue>170</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Text">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="BrightText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="ButtonText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Base">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Window">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Shadow">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="AlternateBase">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="ToolTipBase">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>220</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="ToolTipText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="PlaceholderText">
    <brush brushstyle="SolidPattern">
     <color alpha="127">
```

```
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
  </active>
  <inactive>
   <colorrole role="WindowText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Button">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>240</red>
      <green>240</green>
      <blue>240</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Light">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Midlight">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>227</red>
      <green>227</green>
      <blue>227</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Dark">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>160</red>
      <green>160</green>
      <blue>160</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Mid">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>160</red>
      <green>160</green>
      <blue>160</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Text">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="BrightText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="ButtonText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Base">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>255</red>
      <green>255</green>
      <blue>255</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="Window">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>240</red>
      <green>240</green>
      <blue>240</blue>
     </color>
    </brush>
   </colorrole>
```

```
     <colorrole role="Shadow">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>105</red>
        <green>105</green>
        <blue>105</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="AlternateBase">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>245</red>
        <green>245</green>
        <blue>245</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="ToolTipBase">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>255</red>
        <green>255</green>
        <blue>220</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="ToolTipText">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>0</red>
        <green>0</green>
        <blue>0</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="PlaceholderText">
      <brush brushstyle="SolidPattern">
       <color alpha="128">
        <red>0</red>
        <green>0</green>
        <blue>0</blue>
       </color>
      </brush>
     </colorrole>
    </inactive>
    <disabled>
     <colorrole role="WindowText">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>127</red>
        <green>127</green>
        <blue>127</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Button">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>255</red>
        <green>255</green>
        <blue>255</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Light">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>255</red>
        <green>255</green>
        <blue>255</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Midlight">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>255</red>
        <green>255</green>
        <blue>255</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Dark">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>127</red>
        <green>127</green>
        <blue>127</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Mid">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>170</red>
        <green>170</green>
        <blue>170</blue>
       </color>
      </brush>
     </colorrole>
     <colorrole role="Text">
      <brush brushstyle="SolidPattern">
       <color alpha="255">
        <red>127</red>
        <green>127</green>
        <blue>127</blue>
```

```
      </color>
     </brush>
    </colorrole>
    <colorrole role="BrightText">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>255</red>
       <green>255</green>
       <blue>255</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="ButtonText">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>127</red>
       <green>127</green>
       <blue>127</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="Base">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>255</red>
       <green>255</green>
       <blue>255</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="Window">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>255</red>
       <green>255</green>
       <blue>255</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="Shadow">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>0</red>
       <green>0</green>
       <blue>0</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="AlternateBase">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>245</red>
       <green>245</green>
       <blue>245</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="ToolTipBase">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>255</red>
       <green>255</green>
       <blue>220</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="ToolTipText">
     <brush brushstyle="SolidPattern">
      <color alpha="255">
       <red>0</red>
       <green>0</green>
       <blue>0</blue>
      </color>
     </brush>
    </colorrole>
    <colorrole role="PlaceholderText">
     <brush brushstyle="SolidPattern">
      <color alpha="128">
       <red>0</red>
       <green>0</green>
       <blue>0</blue>
      </color>
     </brush>
    </colorrole>
   </disabled>
  </palette>
 </property>
 <widget class="QWidget" name="gridLayoutWidget_11">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>481</width>
    <height>481</height>
   </rect>
  </property>
  <layout class="QGridLayout" name="board">
   <property name="leftMargin">
    <number>5</number>
   </property>
   <property name="topMargin">
    <number>5</number>
   </property>
   <property name="rightMargin">
    <number>5</number>
   </property>
   <property name="bottomMargin">
```

```
       <number>5</number>
      </property>
      <property name="spacing">
       <number>0</number>
      </property>
     </layout>
    </widget>
   </widget>
   <widget class="QPushButton" name="toolsMenuButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="sizePolicy">
     <sizepolicy hsizetype="Minimum" vsizetype="Fixed">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
    <property name="font">
     <font>
      <pointsize>30</pointsize>
      <stylestrategy>PreferDefault</stylestrategy>
      <kerning>true</kerning>
     </font>
    </property>
    <property name="focusPolicy">
     <enum>Qt::StrongFocus</enum>
    </property>
    <property name="contextMenuPolicy">
     <enum>Qt::DefaultContextMenu</enum>
    </property>
    <property name="layoutDirection">
     <enum>Qt::LeftToRight</enum>
    </property>
    <property name="autoFillBackground">
     <bool>false</bool>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
    background-color: #6C91D5;
}

QPushButton:hover {
    background-color: #7499DD;
}

QPushButton:pressed{
            background-color: #6C91D5;
}</string>
    </property>
    <property name="text">
     <string/>
    </property>
    <property name="iconSize">
     <size>
      <width>22</width>
      <height>22</height>
     </size>
    </property>
    <property name="checkable">
     <bool>false</bool>
    </property>
    <property name="autoDefault">
     <bool>false</bool>
    </property>
    <property name="default">
     <bool>false</bool>
    </property>
    <property name="flat">
     <bool>false</bool>
    </property>
   </widget>
   <widget class="QPushButton" name="openButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>40</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
```

```
     <string/>
    </property>
    <property name="iconSize">
     <size>
      <width>22</width>
      <height>22</height>
     </size>
    </property>
   </widget>
   <widget class="QPushButton" name="saveButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>40</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string/>
    </property>
    <property name="iconSize">
     <size>
      <width>22</width>
      <height>22</height>
     </size>
    </property>
   </widget>
   <widget class="QPushButton" name="quitButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>30</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            padding-left: 7;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string/>
    </property>
    <property name="iconSize">
     <size>
      <width>25</width>
      <height>25</height>
     </size>
    </property>
   </widget>
   <widget class="QPushButton" name="generateButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>560</y>
      <width>110</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
    background-color: #6C91D5;
            color: #212433;
}
```

```
QPushButton:hover {
    background-color: #7499DD;
}

QPushButton:pressed{
            background-color: #6C91D5;
}</string>
    </property>
    <property name="text">
     <string>Generate</string>
    </property>
   </widget>
   <widget class="QPushButton" name="easyButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>560</y>
      <width>80</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            color: #6b92d7;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string>Easy</string>
    </property>
   </widget>
   <widget class="QPushButton" name="hardButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>560</y>
      <width>80</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            color: #6b92d7;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string>Hard</string>
    </property>
   </widget>
   <widget class="QPushButton" name="mediumButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>560</y>
      <width>90</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            color: #6b92d7;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
```

```
    <property name="text">
     <string>Medium</string>
    </property>
   </widget>
   <widget class="QPushButton" name="extremeButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>560</y>
      <width>90</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>12</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            color: #6b92d7;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string>Extreme</string>
    </property>
   </widget>
   <widget class="QPushButton" name="helpButton">
    <property name="geometry">
     <rect>
      <x>15</x>
      <y>10</y>
      <width>50</width>
      <height>50</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>16</pointsize>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton {
            border-radius: 25px;
            background-color: #212433;
            color: #6C91D5;
}

QPushButton:hover {
            background-color: #283140;
}

QPushButton:pressed{
            background-color: #212433;
}</string>
    </property>
    <property name="text">
     <string>?</string>
    </property>
    <property name="iconSize">
     <size>
      <width>30</width>
      <height>30</height>
     </size>
    </property>
   </widget>
   <zorder>helpButton</zorder>
   <zorder>quitButton</zorder>
   <zorder>saveButton</zorder>
   <zorder>openButton</zorder>
   <zorder>resetButton</zorder>
   <zorder>mistakesLabel</zorder>
   <zorder>boardWidget</zorder>
   <zorder>toolsMenuButton</zorder>
   <zorder>easyButton</zorder>
   <zorder>hardButton</zorder>
   <zorder>mediumButton</zorder>
   <zorder>extremeButton</zorder>
   <zorder>generateButton</zorder>
  </widget>
  <action name="actionOpen">
   <property name="text">
    <string>Open</string>
   </property>
   <property name="shortcut">
    <string>Ctrl+O</string>
   </property>
   <property name="iconVisibleInMenu">
    <bool>true</bool>
   </property>
  </action>
  <action name="actionSave">
   <property name="text">
    <string>Save</string>
   </property>
   <property name="shortcut">
    <string>Ctrl+S</string>
```

```
   </property>
  </action>
  <action name="actionQuit">
   <property name="text">
    <string>Quit</string>
   </property>
   <property name="shortcut">
    <string>Ctrl+Q</string>
   </property>
  </action>
  <action name="actionSolve">
   <property name="text">
    <string>Solve</string>
   </property>
   <property name="shortcut">
    <string>Ctrl+R</string>
   </property>
   <property name="iconVisibleInMenu">
    <bool>true</bool>
   </property>
  </action>
  <action name="actionDocumentation">
   <property name="text">
    <string>Documentation</string>
   </property>
  </action>
  <action name="actionAbout">
   <property name="text">
    <string>About</string>
   </property>
  </action>
 </widget>
 <resources/>
 <connections/>
</ui>
```

resources.qrc

```
<RCC>
    <qresource prefix="/icons">
        <file>menuicon.png</file>
        <file>openicon.png</file>
        <file>saveicon.png</file>
        <file>reseticon.png</file>
        <file>quiticon.png</file>
    </qresource>
</RCC>
```

solver.h

```
#ifndef SOLVER_H
#define SOLVER_H

#include "board.h"

enum solver_result { IMPOSIBIL, REZOLVAT, NEDETERMINAT };
solver_result solve(Board &board);
solver_result solve_random(Board &board);

#endif // SOLVER_H
```

solver.cpp

```
#include "solver.h"
#include <utility>
#include <algorithm>

struct stare_e
{
    bool continua;
    bool incompatibil;
};

stare_e completare_e1(Board &board)
{
    stare_e stare = {false, false};
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            if(board.get(i, j) == 0)
            {
                int p = board.get_posib(i, j);
                if(p == 0)
                    return {.continua = false, .incompatibil = true};
                else if( (p & (p-1)) == 0 ) // are un singur bit de 1
                {
                    board.set(i, j, Board::bit2nr(p));
                    stare.continua = true;
                }
            }
    return stare;
}

stare_e completare_e2(Board &board)
{
    stare_e stare = {false, false};
    for(int l = 0; l < 9; l++)
```

```
    {
        int bitmask = (1<<9) - 1;
        for(int c = 0; c < 9; c++)
            if(board.get(l, c))
                bitmask &= ~Board::nr2bit(board.get(l, c));
        for(int nr = 1; nr <= 9; nr++)
            if(bitmask & Board::nr2bit(nr))
            {
                int nrLocuri = 0;
                int loc;
                for(int j = 0; j < 9; j++)
                    if(board.get(l, j) == 0 && board.is_posib(l, j, nr))
                    {
                        nrLocuri++;
                        loc = j;
                    }
                if(nrLocuri == 0)
                    return {.continua = false, .incompatibil = true};
                else if(nrLocuri == 1)
                {
                    board.set(l, loc, nr);
                    stare.continua = true;
                }
            }
    }
    for(int c = 0; c < 9; c++)
    {
        int bitmask = (1<<9) - 1;
        for(int l = 0; l < 9; l++)
            if(board.get(l, c))
                bitmask &= ~Board::nr2bit(board.get(l, c));
        for(int nr = 1; nr <= 9; nr++)
            if(bitmask & Board::nr2bit(nr))
            {
                int nrLocuri = 0;
                int loc;
                for(int i = 0; i < 9; i++)
                    if(board.get(i, c) == 0 && board.is_posib(i, c, nr))
                    {
                        nrLocuri++;
                        loc = i;
                    }
                if(nrLocuri == 0)
                    return {.continua = false, .incompatibil = true};
                else if(nrLocuri == 1)
                {
                    board.set(loc, c, nr);
                    stare.continua = true;
                }
            }
    }
    for(int ls = 0; ls < 9; ls += 3)
        for(int cs = 0; cs < 9; cs += 3)
        {
            int bitmask = (1<<9) - 1;
            for(int l = ls; l <= ls+2; l++)
                for(int c = cs; c <= cs+2; c++)
                    if(board.get(l, c))
                        bitmask &= ~Board::nr2bit(board.get(l, c));

            for(int nr = 1; nr <= 9; nr++)
                if(bitmask & Board::nr2bit(nr))
                {
                    int nrLocuri = 0;
                    std::pair<int,int> loc;
                    for(int l = ls; l <= ls+2; l++)
                        for(int c = cs; c <= cs+2; c++)
                            if(board.get(l, c) == 0 && board.is_posib(l, c, nr))
                            {
                                nrLocuri++;
                                loc = {l, c};
                            }
                    if(nrLocuri == 0)
                        return {.continua = false, .incompatibil = true};
                    else if(nrLocuri == 1)
                    {
                        board.set(loc.first, loc.second, nr);
                        stare.continua = true;
                    }
                }
        }
    return stare;
}

bool completare_e(Board &board)
{
    while(true)
    {
        stare_e s1 = completare_e1(board);
        if(s1.incompatibil)
            return false;
        else if(s1.continua)
            continue;

        stare_e s2 = completare_e2(board);
        if(s2.incompatibil)
            return false;

        if(s2.continua == false)
            break;
    }
    return true;
}

void backtracking(int i, int j, Board &board, Board &board_sol, int & sol)
{
    for(int nr = 1; nr <= 9; nr++)
```

```
        if(board.is_posib(i, j, nr))
        {
            Board board2 = board;
            board2.set(i, j, nr);
            bool ok = completare_e(board2);
            if(!ok)
                continue;
            if(board2.is_complete())
            {
                ++sol;
                if(sol == 1)
                    board_sol.swap(board2);
                if(sol >= 2)
                    return;
            }
            else
            {
                int i2 = i;
                int j2 = j;
                do
                {
                    if(j2 < 8)
                        j2++;
                    else
                    {
                        i2++;
                        j2 = 0;
                    }
                }
                while(board2.get(i2, j2));

                backtracking(i2, j2, board2, board_sol, sol);

                if(sol >= 2)
                    return;
            }
        }
}

void backtracking_random(int i, int j, Board &board, Board &board_sol, int & sol)
{
    int numbers[9];
    for (int i = 0; i < 9; ++i)
        numbers[i] = i+1;
    std::random_shuffle(numbers, numbers+9);

    for(int nri = 0; nri < 9; nri++)
    {
        int nr = numbers[nri];
        if(board.is_posib(i, j, nr))
        {
            Board board2 = board;
            board2.set(i, j, nr);
            bool ok = completare_e(board2);
            if(!ok)
                continue;
            if(board2.is_complete())
            {
                ++sol;
                board_sol.swap(board2);
                return;
            }
            else
            {
                int i2 = i;
                int j2 = j;
                do
                {
                    if(j2 < 8)
                        j2++;
                    else
                    {
                        i2++;
                        j2 = 0;
                    }
                }
                while(board2.get(i2, j2));

                backtracking(i2, j2, board2, board_sol, sol);

                if(sol >= 1)
                    return;
            }
        }
    }
}

solver_result solve(Board &board)
{
    bool ok = completare_e(board);
    if (!ok)
        return solver_result::IMPOSIBIL;
    else if (board.is_complete())
        return solver_result::REZOLVAT;
    else
    {
        int l = 0;
        int c = 0;
        while(board.get(l, c))
            if(c < 8)
                c++;
            else if(l < 8)
            {
                l++;
                c = 0;
            }
        int sol = 0;
```

```
        Board board_sol;
        backtracking(l, c, board, board_sol, sol);
        if(sol == 0)
            return solver_result::IMPOSIBIL;
        else
        {
            board.swap(board_sol);
            if(sol == 1)
                return solver_result::REZOLVAT;
            else
                return solver_result::NEDETERMINAT;
        }
    }
}

solver_result solve_random(Board &board)
{
    bool ok = completare_e(board);
    if (!ok)
        return solver_result::IMPOSIBIL;
    else if (board.is_complete())
        return solver_result::REZOLVAT;
    else
    {
        int l = 0;
        int c = 0;
        while(board.get(l, c))
            if(c < 8)
                c++;
            else if(l < 8)
            {
                l++;
                c = 0;
            }
        int sol = 0;
        Board board_sol;
        backtracking_random(l, c, board, board_sol, sol);
        if(sol == 0)
            return solver_result::IMPOSIBIL;
        else
        {
            board.swap(board_sol);
            if(sol == 1)
                return solver_result::REZOLVAT;
            else
                return solver_result::NEDETERMINAT;
        }
    }
}
```

test.pro

```
QT += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17



# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    board.cpp \
    generator.cpp \
    help.cpp \
    main.cpp \
    mainwindow.cpp \
    solver.cpp \
    textchangedhandler.cpp

HEADERS += \
    board.h \
    generator.h \
    help.h \
    mainwindow.h \
    solver.h \
    textchangedhandler.h

FORMS += \
    help.ui \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

DISTFILES +=

RESOURCES += \
    resources.qrc
```

textchangedhandler.h

```
#ifndef TEXTCHANGEDHANDLER_H
#define TEXTCHANGEDHANDLER_H

#include "board.h"
#include "mainwindow.h"
#include "qtextedit.h"
#include <vector>

using namespace std;

class TextChangedHandler : public QObject
{
    Q_OBJECT

private:
    //variables
    QTextEdit *edit;
    vector<vector<int>> &board_numbers;
    int l, c;
    Board &solution;
    MainWindow* main_window;

    //functions
    bool verif(vector<vector<int>> board_numbers);
    bool complete();

public:
    TextChangedHandler(QTextEdit* edit, vector<vector<int>> &board_numbers,int l, int c,
                       Board &solution, MainWindow* main_window);
    //methods
    static void alignCenter(QTextEdit* edit);

public slots:
    void textEditChanged();
};

#endif // TEXTCHANGEDHANDLER_H
```

textchangedhandler.cpp

```
#include "textchangedhandler.h"
#include "board.h"
#include <iostream>

TextChangedHandler::TextChangedHandler(QTextEdit* edit, vector<vector<int>> &board_numbers, int l, int c,
                                       Board &solution, MainWindow* main_window) :
    edit(edit),
    board_numbers(board_numbers),
    l(l),
    c(c),
    solution(solution),
    main_window(main_window)
{}

// ------------------------------- private functions ------------------------------

bool TextChangedHandler::verif(vector<vector<int>> board_numbers)
{
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            if(board_numbers[l][c] && board_numbers[l][c] != solution.get(l, c))
                return false;
    return true;
}

bool TextChangedHandler::complete()
{
    Board board;
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            board.set(i, j, board_numbers[i][j]);
    return board.is_complete();
}

// ------------------------------- public methods -------------------------------

void TextChangedHandler::alignCenter(QTextEdit* edit)
{
    QTextCursor cursor = edit->textCursor();
    QTextBlockFormat textBlockFormat = cursor.blockFormat();
    textBlockFormat.setAlignment(Qt::AlignHCenter);
    cursor.mergeBlockFormat(textBlockFormat);
    cursor.movePosition(QTextCursor::End);
    edit->setTextCursor(cursor);
}

// ------------------------------- public slots -------------------------------

void TextChangedHandler::textEditChanged()
{
    auto str = edit->toPlainText().toStdString();

    if (str.length() == 0)
    {
        if(board_numbers[l][c] == 0)
            return;
        board_numbers[l][c] = 0;
    }
    else if (str.length() == 1 && std::isdigit(str[0]) && str[0] != '0')
    {
```

```
        if(board_numbers[l][c] == str[0] - '0')
            return;
        board_numbers[l][c] = str[0] - '0';
    }
    else if (str.length() == 2
        && std::isdigit(str[0])
        && std::isdigit(str[1])
        && str[0] != '0'
        && str[1] != '0')
    {
        if (str[0] - '0' != board_numbers[l][c])    //daca s-a adaugat la stanga
            board_numbers[l][c] = str[0]-'0';
        else                                        //daca s-a adaugat la dreapta
            board_numbers[l][c] = str[1]-'0';
        edit->setText(QString((char)(board_numbers[l][c] + '0')));
    }
    else
    {
        if (board_numbers[l][c] == 0)
            edit->setText("");
        else
            edit->setText(QString((char)(board_numbers[l][c] + '0')));
    }
    alignCenter(edit);

    if(board_numbers[l][c] != 0)
    {
        if(verif(board_numbers))
            edit->setStyleSheet("color: #6B92D7;"
                                "background-color: #212433;"
                                "border: no border");
        else
        {
            edit->setStyleSheet("color: #DA5858;"
                                "background-color: #212433;"
                                "border: no border;");
            main_window->increase_mistakes();
        }
    }
    else
        edit->setStyleSheet("background-color: #212433;"
                            "border: no border;");
    if(complete())
        main_window->is_complete();
}
```

IV. Siteografie

- https://doc.qt.io/qt-6/
- https://devdocs.io/qt-qt-core/
- https://cplusplus.com/reference/
- https://en.cppreference.com/w/
- https://devdocs.io/cpp/