Izrada aplikacije za rješavanje Sudoku zagonetke

Haramustek, David

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

IZRADA APLIKACIJE ZA RJEŠAVANJE SUDOKU ZAGONETKE

Rijeka, rujan 2022.

David Haramustek

0069080482

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

IZRADA APLIKACIJE ZA RJEŠAVANJE SUDOKU ZAGONETKE

Mentor: Doc. dr. sc. Ivan Volarić

Rijeka, rujan 2022.

David Haramustek

0069080482

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Programiranje**
Grana: **2.03.03 elektronika**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **David Haramustek (0069080482)**
Studij: Preddiplomski sveučilišni studij elektrotehnike

Zadatak: **Izrada aplikacije za rješavanje Sudoku zagonetke / Sudoku solver application**

Opis zadatka:

U sklopu završnog rada potrebno je u Qt razvojnom okruženju izraditi aplikaciju za rješavanje Sudoku zagonetke. Grafičko sučelje mora sadržavati 9x9 mrežu u koju korisnik unosi ulazne podatke. Program treba ispuniti nepopunjena mjesta s brojevima na način da tvore ispravno rješenje Sudoku zagonetke. Ukoliko postoji više validnih rješenja, potrebno je omogućiti scroll-anje kroz sva rješenja.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

David Haramustek

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Ivan Volarić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Viktor Sučić

IZJAVA

Ja, David Haramustek izjavljujem da je moj završni rad "Izrada aplikacije za rješavanje sudoku zagonetke" izrađen samostalno pod vodstvom mentora doc. dr. sc. Ivana Volarića prema članku 8. Pravilnika o završnom radu, završnom ispitu i završetku preddiplomskih sveučilišnih studija.

David Haramustek

Sadržaj:

1. UVOD

Sudoku je zagonetka temeljena na postavljanju različitih brojeva na prazna mjesta dane zagonetke. Riječ sudoku dolazi od izraza „Su-ji wa dokushin ni kagiru“ čiji prijevod glasi: brojevi moraju biti pojedinačni. Korijeni sudoku zagonetke su u Švicarskoj. Leonhard Euler je u 18. stoljeću stvorio "carré latin" koji je sličan Sudoku slagalici, ali se razlikuje u nekim detaljima. Prva originalna sudoku zagonetka objavljena je 1979. godine, a izmislio ju je Howard Garns. Sudoku slagalica se sastoji od 81 ćelije podijeljene u 9 redaka, stupaca i 3 puta 3 područja ćelija. Rješavanje zagonetke provodi se tako da se u prazne ćelije postavlja broj od 1 do 9 na način da se u svakom stupcu, retku i 3 puta 3 području svaki broj pojavljuje samo jednom. Sudoku zagonetka sastoji se od najmanje 17 zadanih brojeva, ali najčešće ih je zadano od 22 do 30. To je logička, a ne matematička zagonetka te za njeno rješavanje nije potrebno poznavati nikakvu matematiku. Postoji mnogo izvora sudoku slagalica kao što su: časopisi, novine, internet... Zadatak ovog završnog rada je realizacija aplikacije koja rješava sudoku zagonetku u Qt razvojnom okruženju koristeći programski jezik C++. Qt je multiplatformski softver koji se koristi za kreiranje grafičkih korisničkih sučelja (GUI) kao i čitavog niza aplikacija koje mogu raditi na raznim hardverskim i softverskim platformama kao što su Windows, Linux i macOs. Mogu se razvijati i programi koji nisu GUI, kao što su alati naredbenog retka i konzole za poslužitelje.

2. OBJAŠNJENJE PROGRAMSKOG KODA

2.1. Datoteke zaglavlja

2.1.1. CONSTDEF.H

U prvom dijelu koda pod nazivom „constdef.h“ slijedi popis zaglavlja koja su uključena pomoću #include "naziv-zaglavlja". Ova zaglavlja sadrže sve klase koje su potrebne za izvršenje programa. Također, definirana je veličina mreže 9x9 s početnim vrijednostima 0.

```
#pragma once
#include <QApplication>
#include <QObject>
#include <QResizeEvent>
#include <QShowEvent>
#include <QString>
#include <QVector>
#include <QMap>
#include <QList>
#include <QPair>
#include <QTime>
#include <QStringList>
#include <QDateTime>
#include <QTimer>
#include <QDate>
#include <QByteArray>
#include "QProcess"
#include <QDebug>
#include <QWindow>
#include "QWidget"
#include <QDialog>
#include <QFileDialog>
#include <QThread>
#include <QDir>
#include <QSharedMemory>
#include <QtEndian>
#include <QComboBox>
#include <QPushButton>
#include <QVariant>
#include <QByteArray>
#include <QMessageBox>
#include <QFontDatabase>
#include <QMenu>
#include <QAction>
#include <QPainter>
#include <QMutexLocker>
#include <QMutex>
#include <QWaitCondition>
```

```
#include <QByteArray>
#include <QLibrary>
#include <QButtonGroup>
#include <string>
#include <functional>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;
namespace SudokuData
{
#define side 9
       class Sudoku {
     public:
          int actualSolution = 0;

          int grid[side][side] = {
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0}
          };
          int previousGrid[side][side] = {
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0}
          };

        public:
               Sudoku();
        };

}
```

2.1.2. EXTRAWND.H

Klasa „extrawnd.h“ sadrži varijable koje definiraju neke od važnih dimenzija prozora: en_Margin definira koliko prostora treba biti između svake ćelije na mreži, en_CellSize definira koliko svaka pojedinačna ćelija treba biti velika, en_CellUnit postavlja koliko ćelija čini jednu jedinicu (npr. 9

ćelija jednako je 1 kvadratu) i na kraju, en_WndHeight izračunava se množenjem CellSize * CellUnit + 2 * Margin kako bi se dobila ukupna visina sadržaja prozora.

```
en_Margin = 10,

en_CellSize = 45,

en_CellUnit = 9,

en_WndHeight = en_CellSize * en_CellUnit + 2 * en_Margin,
```

Funkcija InitWnd() stvara i upravlja prozorima na zaslonu računala. To su pokazivači na QRect objekte (pravokutnik), QPen objekte (olovka) i QBrush objekte (kist).

```
 void InitWnd();

QRect m_rcAvailable;

QPen m_cluePen;

QBrush m_clueBrush;

QPen m_viewPen;

QBrush m_viewBrush;
```

2.1.3. INPUTCELL.H

U sljedećoj klasi funkcija InitCell() poziva se prilikom kreiranja ćelije i inicijalizacije njenog teksta. Funkcija SetPos() postavlja položaj ćelije u retku nRow i stupcu nCol u nadređenom prozoru, dok funkcija GetPos() vraća trenutni položaj ćelije kao int vrijednost. Funkcije SetClueMode() i SetViewMode() mijenjaju izgled ćelije ovisno o stanju ćelije. Uz to definirana je jedna funkcija SlotTextChanged() koja je ujedno i slot te jedan signal SignalSubmitData(). Pomoću prve funckije definirano je ponašanje ćelije s obzirom na pritiske tipaka, dok se signal emitira prilikom promjene sadržaja ćelije.

```
CInputCell(QWidget *parent);
 ~CInputCell();
void InitCell();
void SetPos(int nRow, int nCol);
void GetPos(int& nRow, int& nCol);
protected Q_SLOTS:
    void  SlotTextChanged(const QString &text);
```

```
Q_SIGNALS:
      void SignalSubmitData(int nRow,int nCol, const QString&
strText ,CInputCell* pCell);
```

2.1.4. SUDOKU.H

Funkcija Init() inicijalizira sve varijable koje koristi ovaj program, uključujući one za ćelije za unos i tipke u prozoru aplikacije, kao i one za rješavanje zagonetki. Funkcija InitCtrl() inicijalizira kontrole za ćelije unosa i tipke u prozoru aplikacije, dok funkcija InitVar() inicijalizira varijable koje koristi ovaj program. SetGUIInitHint() je funkcija koja inicijalizira podatke za korisničko sučelje.

```
void Init();

void InitCtrl();

void InitVar();

void SetGUIInitHint();
```

Funkcija SlotSubmitData() sadrži sav tekst iz bilo koje ćelije. Zatim slijedi deklaracija funkcija; SlotResetButClicked(), SolveClicked(), SlotPreviousButClicked() i SlotNextButClicked() čija zadaća odgovara imenu same funkcije.

```
void SlotSubmitData(int nRow, int nCol, const QString& strText,
CInputCell* pCell);

void SlotResetButClicked();

void SolveClicked();

void SlotPreviousButClicked();

void SlotNextButClicked();
```

Sljedeći dio programskog koda je definicija klase Ui::SudokuClass. Član korisničkog sučelja (Ui) sadrži sve informacije o korisničkom sučelju, kao što su tipke koje su dostupne na korisničkom sučelju, koje se ćelije koriste itd. Varijabla solutions je QVector<QString> koja će sadržavati sva moguća rješenja unesene sudoku zagonetke te prilikom ispisa rješenja u prozoru aplikacije ona će se čitati iz ove varijable. Zatim slijedi deklaracija funkcija koje će poslužiti za rješavanje problematike sudokua, kao i pohranu rješenja.

```
Ui::SudokuClass ui;
QVector<QString> solutions;
bool isSafe_Cell(int row, int col, int num);
bool isSafe_3x3Box(int row, int col, int num);
bool isSafe(int i, int j, int num);
bool get_Cell(int &row, int &col);
void solve();
void transpileStringToGrid(const QString &);
void saveSolution();
void getSolution(int i);
void count_Solution(int &ans, int limit);
```

2.2. Izvorne datoteke

2.2.1. EXTRAWND.CPP

Funkcije u ovoj klasi služe za podešavanje vizualnog identiteta aplikacije i nisu posebno važne za samu funkcionalnost aplikacije, stoga nisu detaljno objašnjene.

```
void CExtraWnd::DrawViewData
```

```
void CExtraWnd::DrawClueData
```

2.2.2. INPUTCELL.CPP

Funkcija void keyPressEvent(), kao što joj i ime sugerira, poziva se prilikom pritiska tipke kada je određena ćelija u fokusu; odnosno prilikom pritiska na tipku enter miče fokus sa ćelije. Funkcija focusInEvent() poziva se kada određena funkcija dobije fokus, prvenstveno pritiskom tipke miša. Funkcija focusOutEvent() poziva se kada određena ćelija izgubi fokus. Sljedeće funkcije definiraju boju ćelije u različitim situacijama. Primjerice, kako će izgledati ćelija kad se pritisne View Result ili kako će izgledati kada korisnik unese pogrešan broj koji ne može biti dio ispravnog rješenja.

```
void CInputCell::SetClueMode(const QString& strText)
```

```
void CInputCell::SetClueModeError(const QString& strText)
```

```
void CInputCell::SetViewMode(const QString& strText)
```

```
void CInputCell::SetInputMode(const QString& strText)
```

2.2.3. MAIN.CPP

Kao što svaki program ima jednu glavnu funkciju, tako i ovaj. Aplikacija će se početi izvršavati od ove funkcije u kojoj je deklarirana varijabla w, tipka Sudoku. Unutar klase Sudoku je implementirana funkcionalnost ove aplikacije.

```
int main(int argc, char *argv[])
{
    QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
```

```
    QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);

    QApplication a(argc, argv);

    Sudoku w;

    w.show();

    return a.exec();

}
```

2.2.4. SUDOKU.CPP

Na početku klase sudoku definirana je funkcija InitCtrl() koja će inicijalizirati korisničko sučelje za cijelu mrežu sudoku slagalice, tj. sve elemente mreže koji se nalaze u 9 redova.

```
void Sudoku::InitCtrl()
{
    int nRow = 0;
    m_vectR0   <<   ui.lineEdit_r0_0   <<   ui.lineEdit_r0_1   <<
ui.lineEdit_r0_2    <<    ui.lineEdit_r0_3<<    ui.lineEdit_r0_4<<
ui.lineEdit_r0_5<<      ui.lineEdit_r0_6<<      ui.lineEdit_r0_7<<
ui.lineEdit_r0_8;
    for (int i=0;i < m_vectR0.size();i++)
    {
        m_vectR0[i]->SetPos(nRow, i);
    }
```

U sljedećem dijelu koda unutar navedene funkcije postavljaju se četiri tipke korisničkog sučelja: reset, prikaz rezultata, sljedeće i prethodno rješenje. Sve tipke imaju isti stil (strButStyle) te su pomoću signal-slot funkcionalnosti povezane s odgovarajućom funkcijom koja se poziva prilikom pritiska na tipku. Tipke za sljedeće i prethodno rješenje u inicijalnom stanju aplikacije nisu vidljive, a to je realizirano metodom setEnabled(false).

```
m_pResetBut = ui.pushButton_resetsudoku;
m_pResetBut->setStyleSheet(strButStyle);
QObject::connect(m_pResetBut,&QPushButton::clicked,this,&Sudoku:
:SlotResetButClicked);
m_pViewResult = ui.pushButton_viewresult;
```

```
m_pViewResult->setStyleSheet(strButStyle);
 m_pPreviousSolution = ui.previousSolutionButton;
 m_pNextSolution = ui.nextSolutionButton;
 m_pPreviousSolution->setEnabled(false);
 m_pNextSolution->setEnabled(false);
QObject::connect(m_pPreviousSolution,&QPushButton::clicked,this,
&Sudoku::SlotPreviousButClicked);
QObject::connect(m_pNextSolution,&QPushButton::clicked,this,&Sud
oku::SlotNextButClicked);
```

Funkcija SignalSubmitData() koristi se za slanje podataka u trenutnu ćeliju mreže. Funkcija prima četiri argumenta; broj retka i stupca ćelije, string s tekstom upisa te pokazivača na ćeliju. String se pretvara u integer te se pomoću funkcije isSafe() provjerava ima li grešaka s ulaznim podacima. Ukoliko je došlo do pogrešnog unosa, na ekranu će se pojaviti tekst „Incorrect Board“ što znači da je korisnik unio neki broj koji ne može biti dio točnog rješenja. Ako nema pogrešaka, tada se unesena vrijednost zapisuje u matricu te ispisuje unutar ćelije.

```
void Sudoku::SlotSubmitData(int nRow, int nCol, const QString&
strText, CInputCell* pCell)
{
     int nVal = strText.toInt();
     if(!isSafe(nRow, nCol, nVal)){
        m_lNumberOfSolution = ui.solutionNumberLabel;
        m_lNumberOfSolution->setText("Incorrect Board");
        m_lActualSolutionNumber = ui.actualSolutionLabel;
        m_lActualSolutionNumber->setText("Incorrect Board");
        pCell->SetClueModeError(QString::number(nVal));
        m_pViewResult->setEnabled(false);
    }else{
        m_sudoku.grid[nRow][nCol] = nVal;
        m_sudoku.previousGrid[nRow][nCol] = nVal;
        pCell->SetClueMode(QString::number(nVal));
    }
}
```

Sljedeće tri funkcije provjeravaju je li moguće unijeti broj u određenu ćeliju sudoku zagonetke. Primaju tri argumenta; redak i stupac ćelije te broj koji korisnik želi unijeti. Funkcija isSafe() poziva dvije funkcije; isSafe_Cell() i isSafe3x3Box().

```
bool Sudoku::isSafe(int i, int j, int num)
```

```
{
    return isSafe_Cell(i, j, num) && isSafe_3x3Box(i, j, num);
}
```

Funkcija isSafe_3x3Box() provjerava je li broj num moguće unijeti u pripadnu 3x3 grupu ćelija te vraća boolean vrijednost u skladu s imenom funkcije.

```
bool Sudoku::isSafe_3x3Box(int row, int col, int num)
{
    int boxx = row / 3;
    int boxy = col / 3;
    for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
    {
        if (m_sudoku.grid[boxx * 3 + i][boxy * 3 + j] == num)
            return false;
    }
    return true;
}
```

Funkcija isSafeCell() provjerava nalazi li se već broj num u istom retku ili istom stupcu sudoku zagonetke te tako vraća boolean vrijednost u skladu s imenom funkcije.

```
bool Sudoku::isSafe_Cell(int row, int col, int num)
{
    for (int i = 0; i < N; i++)
    {
        if (m_sudoku.grid[row][i] == num)
                return false;
        if (m_sudoku.grid[i][col] == num)
                return false;
    }
    return true;
}
```

Pritiskom na tipku View Result poziva se funkcija koja broji broj mogućih rješenja sudoku slagalice, a zatim inicijalizira m_sudoku.actualSolution na 0, što znači da još nema rješenja. Zatim poziva rekurzivnu funkciju solve() koja pronalazi sva moguća rješenja sudoku zagonetke. Nakon toga na ekranu se ispisuje broj mogućih rješenja koji je limitiran na 600 pomoću konstante LIMIT. Nakon toga, prvo dobiveno rješenje se prikazuje u korisničkom sučelju.

```
void Sudoku::SolveClicked()
{
    QString message;
    m_sudoku.actualSolution = 0;
    solve();
    if(solutions.size()>LIMIT){
        message = "more than " + QString::number(LIMIT);
    }else{
        message = QString::number(solutions.size());
    }

    m_lNumberOfSolution = ui.solutionNumberLabel;
    m_lNumberOfSolution->setText(message);
    if(solutions.size()>0){
        getSolution(m_sudoku.actualSolution);
        if(solutions.size()>1){
            m_pNextSolution->setEnabled(true);
        }
        m_lActualSolutionNumber = ui.actualSolutionLabel;
        m_lActualSolutionNumber-
>setText(QString::number(m_sudoku.actualSolution+1));
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i       *       9       +       j]-
>SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
                    m_vectCell[i       *       9       +       j]-
>SetViewMode(QString::number(m_sudoku.grid[i][j]));
```

```
            }
        }
    }
}
```

Funkcija solve() sva moguća rješenja sprema u varijablu solutions te funkcija getSolutions() u kombinaciji s funkcijom transpileStringToGrid() dohvaća jedno od rješenja sudoku zagonetke i zapisuje u odgovarajuću matricu iz koje će se kasnije ispisati odgovarajuće rješenje u korisničkom sučelju.

```
void Sudoku::getSolution(int i){
    if(i>=0 && i<solutions.size()){
        transpileStringToGrid(solutions[i]);
    }
}
void Sudoku::transpileStringToGrid(const QString &s){
    for(int i = 0; i<N; i++)
        for(int j = 0; j <N; j++)
            m_sudoku.grid[i][j]= s[i * 9 + j].digitValue();
}
```

U sljedećem dijelu koda je navedena glavna, rekurzivna funkcija ovoga koda. Služi za rješavanje sudoku zagonetke, spremanje rješenja i brojanje mogućih rješenja. Naredba if (get_Cell(i, j)) provjerava ima li praznih ćelija u mreži. Ukoliko postoji, u praznu ćeliju se upisuje prvi broj za koji funkcija isSafe() kaže da se može upisati, na temelju postojećih brojeva, već upisanih u sudoku mrežu, kao što je već i prije objašnjeno. Postupak se ponavlja, sve dok se sve ćelije ne popune, tj. dok ne dođemo do validnog rješenja. Tada se ažurira brojač rješenja i to dobiveno rješenje se sprema u Qvector solutions pomoću funkcije saveSolution().

```
void Sudoku::count_Solution(int & ans, int limit)
{
    int i, j;
    if(ans>limit)
        return;
    if (get_Cell(i, j))
    {
        for (int num = 1; num <= N; num++)
```

```
        {
            if (isSafe(i, j, num))
            {
                m_sudoku.grid[i][j] = num;
                count_Solution(ans, limit);
                m_sudoku.grid[i][j] = 0;
            }
        }
    }
    else{
        ans++;
        saveSolution();
    }
}
bool Sudoku::get_Cell(int &row, int &col)
{
    for (row = 0; row < N; row++)
    for (col = 0; col < N; col++)
        if (m_sudoku.grid[row][col] == 0)
            return true;
    return false;
}
```

Funkcija saveSolution() sprema validno rješenje zagonetke u vektor sa svim rješenjima *QVector<string> solutions*.

```
void Sudoku::saveSolution(){
    QString s;
    for(int i = 0; i<N;i++)
        for(int j = 0; j<N; j++)
            s.append(QString::number(m_sudoku.grid[i][j]));
    solutions.push_back(s);
}
```

U modu pregleda rješenja uključuje se tipka Reset koja isključuje ostale tipke i postavlja prazne vrijednosti na broju mogućih rješenja i trenutno rješenje te resetira sve vrijednosti ćelija na prazne vrijednosti. Ta funkcionalnost je ostvarena u funkciji SlotResetButClicked().

```
void Sudoku::SlotResetButClicked()
{
   this->SetGUIInitHint();
        if (!m_pViewResult->isEnabled())
        {
                m_pViewResult->setEnabled(true);
        }
   m_pPreviousSolution->setEnabled(false);
   m_pNextSolution->setEnabled(false);
   m_lNumberOfSolution = 0;
   m_lActualSolutionNumber = 0;
   m_lNumberOfSolution = ui.solutionNumberLabel;
   m_lNumberOfSolution->setText("");
   m_lActualSolutionNumber = ui.actualSolutionLabel;
   m_lActualSolutionNumber->setText("");
}
```

Osim tipke Reset, u modu pregledavanja rješenja, potrebno je prikazati sva moguća rješenja sudoku zagonetke (ukoliko ih ima više). U tom slučaju pojavljuju se dvije tipke: Prevoius i Next Solution, čijim se pritiskom pozivaju funkcije SlotPreviousButClicked() i SlotNextButClicked().

```
void Sudoku::SlotPreviousButClicked()
{
    if(m_sudoku.actualSolution>0){
        m_sudoku.actualSolution--;
```

```
        getSolution(m_sudoku.actualSolution);
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i * 9 + j]->SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
                    m_vectCell[i * 9 + j]->SetViewMode(QString::number(m_sudoku.grid[i][j]));
            }
        }
        if(m_sudoku.actualSolution==0)
            m_pPreviousSolution->setEnabled(false);
    }
    m_lActualSolutionNumber = ui.actualSolutionLabel;
    m_lActualSolutionNumber->setText(QString::number(m_sudoku.actualSolution+1));
    m_pNextSolution->setEnabled(true);
}
void Sudoku::SlotNextButClicked()
{
    if(m_sudoku.actualSolution<solutions.size()-1){
        m_pNextSolution->setEnabled(true);
        m_sudoku.actualSolution++;
        getSolution(m_sudoku.actualSolution);
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i * 9 + j]->SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
                    m_vectCell[i * 9 + j]->SetViewMode(QString::number(m_sudoku.grid[i][j]));
            }
        }
```

```
    }
    m_lActualSolutionNumber = ui.actualSolutionLabel;
    m_lActualSolutionNumber-
>setText(QString::number(m_sudoku.actualSolution+1));
    if(m_sudoku.actualSolution==solutions.size()-1)
        m_pNextSolution->setEnabled(false);

    if(m_sudoku.actualSolution==LIMIT-1)
        m_pNextSolution->setEnabled(false);
    else {
        m_pPreviousSolution->setEnabled(true);
    }
}
```

3. PROGRAMSKI KOD

CONSTDEF.H

```
#pragma once
#include <QApplication>
#include <QObject>
#include <QResizeEvent>
#include <QShowEvent>
#include <QString>
#include <QVector>
#include <QMap>
#include <QList>
#include <QPair>
#include <QTime>
#include <QStringList>
#include <QDateTime>
#include <QTimer>
#include <QDate>
#include <QByteArray>
#include "QProcess"
#include <QDebug>
#include <QWindow>
#include "QWidget"
#include <QDialog>
#include <QFileDialog>
#include <QThread>
#include <QDir>
#include <QSharedMemory>
#include <QtEndian>
#include <QComboBox>
#include <QPushButton>
#include <QVariant>
#include <QByteArray>
#include <QMessageBox>
```

```
#include <QFontDatabase>
#include <QMenu>
#include <QAction>
#include <QPainter>
#include <QMutexLocker>
#include <QMutex>
#include <QWaitCondition>
#include <QByteArray>
#include <QLibrary>
#include <QButtonGroup>
#include <string>
#include <functional>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;
namespace SudokuData
{
#define side 9

     class Sudoku {
    public:
        int actualSolution = 0;
        int grid[side][side] = {
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
           {0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
            {0, 0, 0, 0, 0, 0, 0, 0, 0}
        };
        int previousGrid[side][side] = {
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0}
        };
    public:
         Sudoku();
    };
}
EXTRAWND.H
#pragma once
#include "constdef.h"
#include <QWidget>
class CExtraWnd : public QWidget
{
    Q_OBJECT
public:
    CExtraWnd(QWidget *parent);
   ~CExtraWnd();
    enum
    {
         en_Margin = 10,
         en_CellSize = 45,
         en_CellUnit = 9,
         en_WndHeight = en_CellSize * en_CellUnit + 2 * en_Margin,
```

```
    };
protected:
    void InitWnd();
    void resizeEvent(QResizeEvent *event);
    void DrawClueData(QPainter& painter, int& nVal,QRect& rect);
    void DrawViewData(QPainter& painter, int& nVal, QRect& rect);
private:
    QRect m_rcAvailable;
    SudokuData::Sudoku m_sudoku;
    QPen m_cluePen;
    QBrush m_clueBrush;
    QPen m_viewPen;
    QBrush m_viewBrush;
};
```

INPUTCELL.H

```
#pragma once
#include "constdef.h"
#include <QLineEdit>
class CInputCell : public QLineEdit
{
    Q_OBJECT
public:
    CInputCell(QWidget *parent);
    ~CInputCell();
    void InitCell();
    void SetPos(int nRow, int nCol);
    void GetPos(int& nRow, int& nCol);
    void SetClueMode(const QString& strText);
    void SetViewMode(const QString& strText);
    void SetInputMode(const QString& strText);
   void SetClueModeError(const QString &strText);
protected:
    void keyPressEvent(QKeyEvent * event);
```

```
    void focusInEvent(QFocusEvent * event);
    void focusOutEvent(QFocusEvent * event);
protected Q_SLOTS:
    void  SlotTextChanged(const QString &text);
Q_SIGNALS:
    void  SignalSubmitData(int  nRow,int  nCol,  const  QString&
strText ,CInputCell* pCell);
private:
    int m_nRow;
    int m_nCol;
    QString m_strCurrentText;
};
```

SUDOKU.H

```
#pragma once
#include <QtWidgets>
#include <QtWidgets/QWidget>
#include "inputcell.h"
#include "ui_sudoku.h"
#include <QVector>
#define N 9
#define LIMIT 600
using namespace std;
class Sudoku : public QWidget
{
    Q_OBJECT
public:
    Sudoku(QWidget *parent = Q_NULLPTR);
    ~Sudoku();

protected:

     void Init();
     void InitCtrl();
     void InitVar();
```

```
      void showEvent(QShowEvent *event);
      void SetGUIInitHint();
protected Q_SLOTS:
      void SlotSubmitData(int nRow, int nCol, const QString&
strText, CInputCell* pCell);
      void SlotResetButClicked();
      void SolveClicked();
      void SlotPreviousButClicked();
    void SlotNextButClicked();
private:
    Ui::SudokuClass ui;
    QVector<QString> solutions;
     bool m_blFirstShow;
     QVector<CInputCell*> m_vectR0;
     QVector<CInputCell*> m_vectR1;
     QVector<CInputCell*> m_vectR2;
     QVector<CInputCell*> m_vectR3;
     QVector<CInputCell*> m_vectR4;
     QVector<CInputCell*> m_vectR5;
     QVector<CInputCell*> m_vectR6;
     QVector<CInputCell*> m_vectR7;
     QVector<CInputCell*> m_vectR8;
     QVector<CInputCell*> m_vectCell;
    QPushButton* m_pResetBut;
    QPushButton* m_pViewResult;
    QPushButton* m_pPreviousSolution;
    QPushButton* m_pNextSolution;
    QLabel* m_lNumberOfSolution = nullptr;
    QLabel* m_lActualSolutionNumber = nullptr;
    SudokuData::Sudoku m_sudoku;
    bool isSafe_Cell(int row, int col, int num);
    bool isSafe_3x3Box(int row, int col, int num);
    bool isSafe(int i, int j, int num);
    bool get_Cell(int &row, int &col);
```

```
    void solve();
    void transpileStringToGrid(const QString &);
    void saveSolution();
    void getSolution(int i);
    void count_Solution(int &ans, int limit);
};
```

EXTRAWND.CPP

```
#include "extrawnd.h"
CExtraWnd::CExtraWnd(QWidget *parent)
     : QWidget(parent)
{
     this->InitWnd();
}
CExtraWnd::~CExtraWnd()
{
}
void CExtraWnd::InitWnd()
{
     m_cluePen.setColor(Qt::white);
     m_clueBrush.setColor(Qt::gray);
     m_viewPen.setColor(Qt::black);
     m_viewBrush.setColor(QColor(255, 192, 203));
}
void  CExtraWnd::resizeEvent(QResizeEvent *event)
{
     QWidget::resizeEvent(event);
     QSize szClient = QWidget::size();
     if (szClient.height() == 0)
     {
          m_rcAvailable.setHeight(0);
          return;
     }
```

```
    m_rcAvailable = QRect( (szClient.width() - 2 * en_Margin -
en_CellUnit        *        en_CellSize)/2+10,        en_Margin,
en_CellUnit*en_CellSize, en_CellUnit*en_CellSize);
}
void CExtraWnd::DrawClueData(QPainter& painter, int& nVal, QRect&
rect)
{
    painter.save();
    QRect rc = rect.adjusted(1, 1, -1, -1);
    QFont ft = painter.font();
    painter.setPen(Qt::NoPen);
    painter.fillRect(rc, Qt::gray);
    painter.setPen(m_cluePen);
    ft.setPixelSize(18);
    ft.setBold(true);
    painter.setFont(ft);
    painter.drawText(rc,                          Qt::AlignCenter,
QString::number(nVal));
    painter.restore();
}
void CExtraWnd::DrawViewData(QPainter& painter, int& nVal, QRect&
rect)
{
    painter.save();
    QRect rc = rect.adjusted(1, 1, -1, -1);
    QFont ft = painter.font();
    painter.setPen(Qt::NoPen);
    painter.fillRect(rc, QColor(255, 192, 203));
    painter.setPen(m_viewPen);
    ft.setPixelSize(18);
    ft.setBold(true);
    painter.setFont(ft);
    painter.drawText(rc,                          Qt::AlignCenter,
QString::number(nVal));
    painter.restore();
}
```

INPUTCELL.CPP

```
#include "inputcell.h"
CInputCell::CInputCell(QWidget *parent)
     : QLineEdit(parent)
{
}
CInputCell::~CInputCell()
{
}
void CInputCell::InitCell()
{
     QString strLineEditStyle = ""
          "QLineEdit     {     qproperty-alignment:AlignHCenter; font:bold  ;  font-size:  18px;     border:1px  solid  #B0B0B0; color:rgba(0,0,0,200); background-color:rgb(64,244,208); }"
          "QLineEdit:hover { border:1px solid #707070;background-color:#F7F7F7; }"
          "QLineEdit:focus       {       border:1px       solid #1274B7;color:rgba(0,0,0,200); }   "
          "QLineEdit:disabled      {      border:1px      solid lightgray;color:white;background-color:gray; }   "
          ;
     this->setStyleSheet(strLineEditStyle);
     QIntValidator* pIntV = new QIntValidator(1, 9, this);
     this->setValidator(pIntV);
     this->setMaxLength(1);
     QObject::connect(this,     &QLineEdit::textChanged,     this, &CInputCell::SlotTextChanged);
}
void  CInputCell::SlotTextChanged(const QString &text)
{
     if (!text.isEmpty() && text.toInt() == 0)
     {
          this->backspace();
     }
}
```

```
void   CInputCell::keyPressEvent(QKeyEvent * event)
{
     if  (event->key()  ==  Qt::Key_Return  ||  event->key()  ==
Qt::Key_Enter)
     {
          this->clearFocus();
     }
     QLineEdit::keyPressEvent(event);
}
void CInputCell::SetPos(int nRow, int nCol)
{
     m_nRow = nRow;
     m_nCol = nCol;
}
void CInputCell::GetPos(int& nRow, int& nCol)
{
     nRow = m_nRow;
     nCol = m_nCol;
}
void CInputCell::focusInEvent(QFocusEvent * event)
{
     m_strCurrentText = this->text();
     QLineEdit::focusInEvent(event);
}
void CInputCell::focusOutEvent(QFocusEvent * event)
{
     QString strNewText = this->text();
     if (strNewText != m_strCurrentText)
     {
          emit SignalSubmitData(m_nRow, m_nCol, strNewText, this);
     }
     QLineEdit::focusOutEvent(event);
}
void CInputCell::SetClueMode(const QString& strText)
```

```
{
     QString strLineEditStyle = ""
          "QLineEdit     {     qproperty-alignment:AlignHCenter;
font:bold  ;  font-size:  18px;    border:1px  solid  #B0B0B0;
color:rgba(0,0,0,200); background-color:rgb(64,244,208); }"
          "QLineEdit:hover { border:1px solid #707070;background-
color:#F7F7F7; }"
          "QLineEdit:focus       {       border:1px       solid
#1274B7;color:rgba(0,0,0,200); }   "
          "QLineEdit:disabled      {      border:1px      solid
lightgray;color:white;background-color:gray; }   "
          ;
     this->setStyleSheet(strLineEditStyle);
     this->setEnabled(false);
     QLineEdit::setText(strText);
}
void CInputCell::SetClueModeError(const QString& strText)
{
    QString strLineEditStyle = ""
        "QLineEdit { qproperty-alignment:AlignHCenter;  font:bold
;    font-size:    18px;        border:1px    solid    #B0B0B0;
color:rgba(0,0,0,200); background-color:rgb(64,244,208); }"
        "QLineEdit:hover  {  border:1px  solid  #707070;background-
color:#F7F7F7; }"
        "QLineEdit:focus        {        border:1px        solid
#1274B7;color:rgba(0,0,0,200); }   "
        "QLineEdit:disabled       {       border:1px       solid
lightgray;color:white;background-color:red; }   "
        ;
    this->setStyleSheet(strLineEditStyle);
    this->setEnabled(false);
    QLineEdit::setText(strText);
}
void CInputCell::SetViewMode(const QString& strText)
{
     QString strLineEditStyle = ""
```

```
          "QLineEdit     {     qproperty-alignment:AlignHCenter;
font:bold  ;  font-size:  18px;     border:1px  solid  #B0B0B0;
color:rgba(0,0,0,200); background-color:rgb(64,244,208); }"

          "QLineEdit:hover { border:1px solid #707070;background-
color:#F7F7F7; }"

          "QLineEdit:focus       {       border:1px       solid
#1274B7;color:rgba(0,0,0,200); }   "

          "QLineEdit:disabled      {      border:1px      solid
lightgray;color:black;background-color:rgb(255,192,203); }   "

          ;

     this->setStyleSheet(strLineEditStyle);

     this->setEnabled(false);

     QLineEdit::setText(strText);

}

void CInputCell::SetInputMode(const QString& strText)

{

     QString strLineEditStyle = ""

          "QLineEdit     {     qproperty-alignment:AlignHCenter;
font:bold  ;  font-size:  18px;     border:1px  solid  #B0B0B0;
color:rgba(0,0,0,200); background-color:rgb(64,244,208); }"

          "QLineEdit:hover { border:1px solid #707070;background-
color:#F7F7F7; }"

          "QLineEdit:focus       {       border:1px       solid
#1274B7;color:rgba(0,0,0,200); }   "

          "QLineEdit:disabled      {      border:1px      solid
lightgray;color:white;background-color:gray; }   "

          ;

     this->setStyleSheet(strLineEditStyle);

     this->setEnabled(true);

     QLineEdit::setText(strText);

}

MAIN.CPP

#include "sudoku.h"

#include <QObject>

#include <QApplication>

#include <QtGui>

#include <QtCore>
```

```
#include <QtWidgets>

int main(int argc, char *argv[])
{
     QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
     QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
     QApplication a(argc, argv);
     Sudoku w;
     w.show();
     return a.exec();
}
```

SUDOKU.CPP

```
#include "sudoku.h"
Sudoku::Sudoku(QWidget *parent)
     : QWidget(parent), m_blFirstShow(false)
{
     ui.setupUi(this);
     this->setWindowFlags(this->windowFlags()                    |
Qt::MSWindowsFixedSizeDialogHint | Qt::CustomizeWindowHint);
     this->Init();
}
Sudoku::~Sudoku()
{

}
void Sudoku::Init()
{
    this->InitCtrl();
    this->InitVar();
}
void Sudoku::InitCtrl()
{
    int nRow = 0;
```

```
    m_vectR0   <<   ui.lineEdit_r0_0   <<   ui.lineEdit_r0_1   <<
ui.lineEdit_r0_2    <<    ui.lineEdit_r0_3<<    ui.lineEdit_r0_4<<
ui.lineEdit_r0_5<<      ui.lineEdit_r0_6<<      ui.lineEdit_r0_7<<
ui.lineEdit_r0_8;

    for (int i=0;i < m_vectR0.size();i++)

    {

        m_vectR0[i]->SetPos(nRow, i);

    }

    m_vectR1   <<   ui.lineEdit_r1_0   <<   ui.lineEdit_r1_1   <<
ui.lineEdit_r1_2  <<  ui.lineEdit_r1_3  <<  ui.lineEdit_r1_4  <<
ui.lineEdit_r1_5  <<  ui.lineEdit_r1_6  <<  ui.lineEdit_r1_7  <<
ui.lineEdit_r1_8;

    ++nRow;

    for (int i = 0; i < m_vectR1.size(); i++)

    {

        m_vectR1[i]->SetPos(nRow, i);

    }

    m_vectR2   <<   ui.lineEdit_r2_0   <<   ui.lineEdit_r2_1   <<
ui.lineEdit_r2_2  <<  ui.lineEdit_r2_3  <<  ui.lineEdit_r2_4  <<
ui.lineEdit_r2_5  <<  ui.lineEdit_r2_6  <<  ui.lineEdit_r2_7  <<
ui.lineEdit_r2_8;

    ++nRow;

    for (int i = 0; i < m_vectR2.size(); i++)

    {

        m_vectR2[i]->SetPos(nRow, i);

    }

    m_vectR3   <<   ui.lineEdit_r3_0   <<   ui.lineEdit_r3_1   <<
ui.lineEdit_r3_2  <<  ui.lineEdit_r3_3  <<  ui.lineEdit_r3_4  <<
ui.lineEdit_r3_5  <<  ui.lineEdit_r3_6  <<  ui.lineEdit_r3_7  <<
ui.lineEdit_r3_8;

    ++nRow;

    for (int i = 0; i < m_vectR3.size(); i++)

    {

        m_vectR3[i]->SetPos(nRow, i);

    }

    m_vectR4   <<   ui.lineEdit_r4_0   <<   ui.lineEdit_r4_1   <<
ui.lineEdit_r4_2  <<  ui.lineEdit_r4_3  <<  ui.lineEdit_r4_4  <<
ui.lineEdit_r4_5  <<  ui.lineEdit_r4_6  <<  ui.lineEdit_r4_7  <<
ui.lineEdit_r4_8;
```

```
    ++nRow;
    for (int i = 0; i < m_vectR4.size(); i++)
    {
        m_vectR4[i]->SetPos(nRow, i);
    }
    m_vectR5  <<  ui.lineEdit_r5_0  <<  ui.lineEdit_r5_1  <<
ui.lineEdit_r5_2  <<  ui.lineEdit_r5_3  <<  ui.lineEdit_r5_4  <<
ui.lineEdit_r5_5  <<  ui.lineEdit_r5_6  <<  ui.lineEdit_r5_7  <<
ui.lineEdit_r5_8;
    ++nRow;
    for (int i = 0; i < m_vectR5.size(); i++)
    {
        m_vectR5[i]->SetPos(nRow, i);
    }
    m_vectR6  <<  ui.lineEdit_r6_0  <<  ui.lineEdit_r6_1  <<
ui.lineEdit_r6_2  <<  ui.lineEdit_r6_3  <<  ui.lineEdit_r6_4  <<
ui.lineEdit_r6_5  <<  ui.lineEdit_r6_6  <<  ui.lineEdit_r6_7  <<
ui.lineEdit_r6_8;
    ++nRow;
    for (int i = 0; i < m_vectR6.size(); i++)
    {
        m_vectR6[i]->SetPos(nRow, i);
    }
    m_vectR7  <<  ui.lineEdit_r7_0  <<  ui.lineEdit_r7_1  <<
ui.lineEdit_r7_2  <<  ui.lineEdit_r7_3  <<  ui.lineEdit_r7_4  <<
ui.lineEdit_r7_5  <<  ui.lineEdit_r7_6  <<  ui.lineEdit_r7_7  <<
ui.lineEdit_r7_8;
    ++nRow;
    for (int i = 0; i < m_vectR7.size(); i++)
    {
        m_vectR7[i]->SetPos(nRow, i);
    }
    m_vectR8  <<  ui.lineEdit_r8_0  <<  ui.lineEdit_r8_1  <<
ui.lineEdit_r8_2  <<  ui.lineEdit_r8_3  <<  ui.lineEdit_r8_4  <<
ui.lineEdit_r8_5  <<  ui.lineEdit_r8_6  <<  ui.lineEdit_r8_7  <<
ui.lineEdit_r8_8;
    ++nRow;
    for (int i = 0; i < m_vectR8.size(); i++)
```

```
    {
        m_vectR8[i]->SetPos(nRow, i);
    }
    m_vectCell  << m_vectR0 << m_vectR1 << m_vectR2 << m_vectR3 << m_vectR4 << m_vectR5 << m_vectR6 << m_vectR7<<m_vectR8;
    for(auto var : m_vectCell)
    {
        var->InitCell();
        var->setText("");
        QObject::connect(var,      &CInputCell::SignalSubmitData, this, &Sudoku::SlotSubmitData);
    }
     QString strButStyle = QString(
          "QPushButton        { border-radius: 5px;  border: 2px solid  rgba(44,  61,  113,120);    color  :  #FFFFFF;  background: rgba(10,10,255,225); }"
          " QPushButton:focus{ padding: -1; } "
          "QPushButton:disabled    {background: gray;}"
          "QPushButton:hover  {border: none; background:  rgb(44, 61, 113) ;}"
          "QPushButton:pressed{border:none;}"
          "QPushButton:checked          {border: none;background: rgb(216,30,6);}"
          "QPushButton:checked:hover  {border: none; background: rgb(150,30,6) ; } "
          "QPushButton:checked:pressed{border: none;}"
     );
    m_pResetBut = ui.pushButton_resetsudoku;
     m_pResetBut->setStyleSheet(strButStyle);
     QObject::connect(m_pResetBut,&QPushButton::clicked,this,&Sudoku::SlotResetButClicked);
    m_pViewResult = ui.pushButton_viewresult;
     m_pViewResult->setStyleSheet(strButStyle);
    m_pPreviousSolution = ui.previousSolutionButton;
    m_pNextSolution = ui.nextSolutionButton;
    m_pPreviousSolution->setEnabled(false);
```

```
    m_pNextSolution->setEnabled(false);


QObject::connect(m_pPreviousSolution,&QPushButton::clicked,this,
&Sudoku::SlotPreviousButClicked);

QObject::connect(m_pNextSolution,&QPushButton::clicked,this,&Sud
oku::SlotNextButClicked);

    m_lNumberOfSolution = ui.solutionNumberLabel;

    m_lActualSolutionNumber = ui.actualSolutionLabel;

    QObject::connect(m_pViewResult, &QPushButton::clicked, this,
&Sudoku::SolveClicked);

     m_pViewResult->setEnabled(false);

     ui.scrollArea->setWidgetResizable(true);

}

void Sudoku::InitVar()

{

    this->SetGUIInitHint();

}

void Sudoku::showEvent(QShowEvent *event)

{

    if (!m_blFirstShow)

    {

        m_blFirstShow = true;

        m_pResetBut->click();

    }

    QWidget::showEvent(event);

}

void Sudoku::SlotSubmitData(int nRow, int nCol, const QString&
strText, CInputCell* pCell) //1

{

     int nVal = strText.toInt();

    if(!isSafe(nRow, nCol, nVal)){

        m_lNumberOfSolution = ui.solutionNumberLabel;

        m_lNumberOfSolution->setText("Incorrect Board");

        m_lActualSolutionNumber = ui.actualSolutionLabel;

        m_lActualSolutionNumber->setText("Incorrect Board");
```

```
            pCell->SetClueModeError(QString::number(nVal));
            m_pViewResult->setEnabled(false);
        }else{
            m_sudoku.grid[nRow][nCol] = nVal;
            m_sudoku.previousGrid[nRow][nCol] = nVal;
            pCell->SetClueMode(QString::number(nVal));
        }
}
void Sudoku::SlotResetButClicked()
{
    this->SetGUIInitHint();

     if (!m_pViewResult->isEnabled())
     {
          m_pViewResult->setEnabled(true);
     }
    m_pPreviousSolution->setEnabled(false);
    m_pNextSolution->setEnabled(false);
    m_lNumberOfSolution = 0;
    m_lActualSolutionNumber = 0;
    m_lNumberOfSolution = ui.solutionNumberLabel;
    m_lNumberOfSolution->setText("");
    m_lActualSolutionNumber = ui.actualSolutionLabel;
    m_lActualSolutionNumber->setText("");
}
void Sudoku::SetGUIInitHint()
{
    for(auto var : m_vectCell)
    {
        var->setText("");
        int nRow, nCol;
        var->GetPos(nRow, nCol);
        var->SetInputMode("");
```

```
    }
    for(int i = 0; i<N; i++)
        for(int j = 0; j <N; j++) {
            m_sudoku.previousGrid[i][j]= 0;
            m_sudoku.grid[i][j]= 0;
        }
    solutions.clear();
}
void Sudoku::SolveClicked()
{
    QString message;
    m_sudoku.actualSolution = 0;
    solve();
    if(solutions.size()>LIMIT){
        message = "more than " + QString::number(LIMIT);
    }else{
        message = QString::number(solutions.size());
    }
    m_lNumberOfSolution = ui.solutionNumberLabel;
    m_lNumberOfSolution->setText(message);
    if(solutions.size()>0){
        getSolution(m_sudoku.actualSolution);
        if(solutions.size()>1){
            m_pNextSolution->setEnabled(true);
        }
        m_lActualSolutionNumber = ui.actualSolutionLabel;
        m_lActualSolutionNumber-
>setText(QString::number(m_sudoku.actualSolution+1));
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i       *       9       +       j]-
>SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
```

```
                    m_vectCell[i       *       9       +       j]-
>SetViewMode(QString::number(m_sudoku.grid[i][j]));
            }
        }
    }
}
void Sudoku::SlotPreviousButClicked()
{
    if(m_sudoku.actualSolution>0){
        m_sudoku.actualSolution--;
        getSolution(m_sudoku.actualSolution);
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i       *       9       +       j]-
>SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
                    m_vectCell[i       *       9       +       j]-
>SetViewMode(QString::number(m_sudoku.grid[i][j]));
            }
        }
        if(m_sudoku.actualSolution==0)
            m_pPreviousSolution->setEnabled(false);
    }
    m_lActualSolutionNumber = ui.actualSolutionLabel;
    m_lActualSolutionNumber-
>setText(QString::number(m_sudoku.actualSolution+1));
    m_pNextSolution->setEnabled(true);
}
void Sudoku::SlotNextButClicked()
{
    if(m_sudoku.actualSolution<solutions.size()-1){
        m_pNextSolution->setEnabled(true);
        m_sudoku.actualSolution++;
        getSolution(m_sudoku.actualSolution);
```

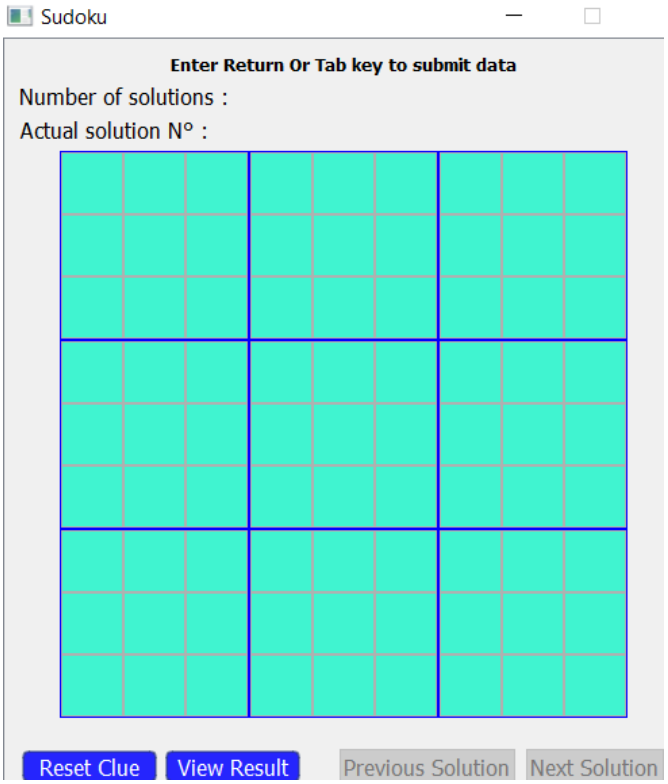
```
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if(m_sudoku.previousGrid[i][j]!=0)
                    m_vectCell[i * 9 + j]->SetClueMode(QString::number(m_sudoku.grid[i][j]));
                else
                    m_vectCell[i * 9 + j]->SetViewMode(QString::number(m_sudoku.grid[i][j]));
            }
        }
    }
    m_lActualSolutionNumber = ui.actualSolutionLabel;
    m_lActualSolutionNumber->setText(QString::number(m_sudoku.actualSolution+1));
    if(m_sudoku.actualSolution==solutions.size()-1)
        m_pNextSolution->setEnabled(false);
    if(m_sudoku.actualSolution==LIMIT-1)
        m_pNextSolution->setEnabled(false);
    else {
        m_pPreviousSolution->setEnabled(true);
    }
}
void Sudoku::transpileStringToGrid(const QString &s){
    for(int i = 0; i<N; i++)
        for(int j = 0; j <N; j++)
            m_sudoku.grid[i][j]= s[i * 9 + j].digitValue();
}
void Sudoku::saveSolution(){
    QString s;
    for(int i = 0; i<N;i++)
        for(int j = 0; j<N; j++)
            s.append(QString::number(m_sudoku.grid[i][j]));
    solutions.push_back(s);
}
```

```
void Sudoku::getSolution(int i){
    if(i>=0 && i<solutions.size()){
        transpileStringToGrid(solutions[i]);
    }
}
bool Sudoku::isSafe_Cell(int row, int col, int num)
{
    for (int i = 0; i < N; i++)
    {
        if (m_sudoku.grid[row][i] == num)
                return false;
        if (m_sudoku.grid[i][col] == num)
                return false;
    }
    return true;
}
bool Sudoku::isSafe_3x3Box(int row, int col, int num)
{
    int boxx = row / 3;
    int boxy = col / 3;

    for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
    {
        if (m_sudoku.grid[boxx * 3 + i][boxy * 3 + j] == num)
            return false;
    }
    return true;
}
bool Sudoku::isSafe(int i, int j, int num)
{
    return isSafe_Cell(i, j, num) && isSafe_3x3Box(i, j, num);
}
```

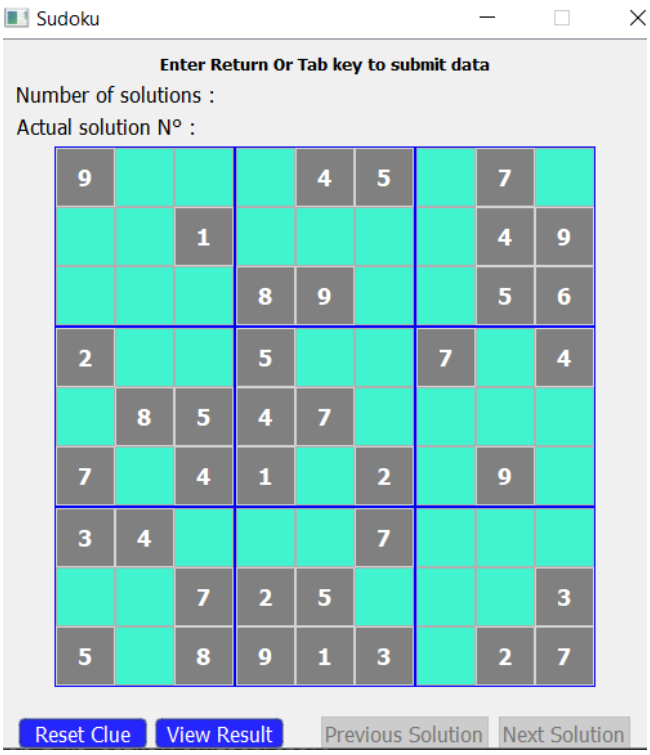
```
bool Sudoku::get_Cell(int &row, int &col)
{
    for (row = 0; row < N; row++)
    for (col = 0; col < N; col++)
        if (m_sudoku.grid[row][col] == 0)
            return true;
    return false;
}
void Sudoku::count_Solution(int & ans, int limit)
{
    int i, j;
    if(ans>limit)
        return;
    if (get_Cell(i, j))
    {
        for (int num = 1; num <= N; num++)
        {
            if (isSafe(i, j, num))
            {
                m_sudoku.grid[i][j] = num;
                count_Solution(ans, limit);
                m_sudoku.grid[i][j] = 0;
            }
        }
    }
    else{
        ans++;
        saveSolution();
    }
}
void Sudoku::solve()
{
    int a = 0;
```

```
    count_Solution(a, LIMIT);
}
```



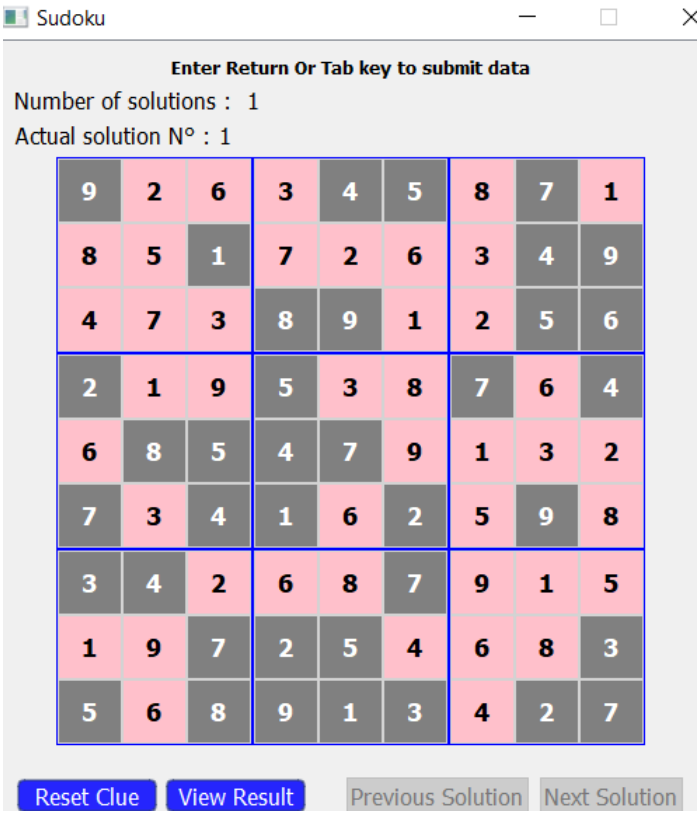
Slika 1. Izlazni rezultati aplikacije

Na slici iznad (*Slika 1.*) prikazana je prazna slagalica do koje se dolazi izvršavanjem programskog koda prikazanog u gornjem dijelu rada, u koju korisnik unosi ulazne podatke, tj. brojeve koji predstavljaju polazišnu točku od koje kreće rješavanje sudoku slagalice.



Slika 2. Slagalica s ulaznim podacima

Na slici iznad (*Slika 2.*) vidljiva je sudoku slagalica popunjena ulaznim vrijednostima koje je korosnik sam unio, a na slici ispod *(Slika 3.*) prikazano je rješenje sudoku zagonetke na temelju zadanih ulaznih podataka.



Slika 3. Slagalica s rješenjem

Pritiskom na tipku View Result aplikacija rješava odgovarajuću sudoku slagalicu i ispisuje rješenja. Ukoliko se zadani sudoku sastoji od više od jednog mogućeg rješenja, klikom na tipke za navigaciju (Previous Solution i Next Solution) aplikacija prikazuje i sva ostala moguća rješenja.

4. ZAKLJUČAK

U današnje se vrijeme neprestano razvijaju razne nove aplikacije kojima se nastoje ubrzati, automataizirati i olakšati procesi koje smo navikli raditi ručno. Sudoku zagonetke, koje smo navikli viđati u papirnatim novinama i slično, također je moguće automatizirati pa je cilj ovog rada bio izraditi aplikaciju za rješavanje sudoku zagonetke uz pomoć nekoliko kratkih klikova na računalu. Aplikacija je implementirana u programskom okruženju Qt, u programskom jeziku C++. Sudoku je izvrstan primjer rekurzivne funkcije jer je koncipiran tako da se isprobavaju sva moguća rješenja sve dok se ne dođe do jednog ili više ispravnih rješenja. Izvršavanjem programskog koda dolazi se do inicijaliziranja tablice u koju korisnik može unijeti početne, proizvoljne vrijednosti pri čemu se mora držati odgovarajućih pravila koja su ista kao i kod samog rješavanja sudokua. Na temelju upisanih ulaznih podataka korisnik jednim klikom dolazi do rješenja do kojih je nekada vrlo izazovno doći ako se sudoku rješava na papiru.

Popis literature

1. Qt | Cross-platform software development for embedded devices. Dostupno na: https://www.qt.io/ [25.3.2022.]
2. Stack Overflow. Dostupno na: https://stackoverflow.com/ [17.5.2022.]
3. GitHub: Where the world builds software. Dostupno na: https://github.com/ [17.5.2022.]
4. Play Free Sudoku online - solve web sudoku puzzles. Dostupno na: https://sudoku.com/ [20.5.2022.]
5. What is Sudoku? Dostupno na: http://www.sudoku-space.com/sudoku.php [5.9.2022.]

Popis slika

5. SAŽETAK I KLJUČNE RIJEČI

Tema ovoga rada je izrada aplikacije za rješavanje sudoku zagonetke. U prvom poglavlju rada prikazano je objašnjenje programskog koda. U drugom poglavlju je kopija cijelog programskog koda od kojeg se sastoji aplikacija. Nakon toga slijedi vizualni prikaz izgleda same aplikacije s postojećim primjerom slagalice i njenim rješenjem.

Ključne riječi: sudoku, Qt.

6. SUMMARY AND KEY WORDS

The topic of this bachelors thesis is the realization of application for solving sudoku puzzles. In the first chapter is presented explanation of the program code. In the second chapter, the entire application program code is given. This is followed by a visual representation of how the application looks with an existing puzzle example and it's solution.

Key words: sudoku, Qt.