ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и технологий (ИКНТ)

(наименование учебного подразделения)

Отчет о прохождении учебной **практики**

(вид и тип практики)

Стойкоски Никола

(Ф.И.О. обучающегося)

II курс, группа 23534/1

(номер курса обучения и учебной группы)

09.03.04 - Программная инженерия

(Направление подготовки (код и наименование)

Место прохождения практики: Высшая школа программной инженерии (ВШПИ)

(указывается наименование профильной организации или наименование структурного подразделения

ФГАОУ ВО «СПбПУ», фактический адрес)

Сроки практики: 25.06.18-21.07.18

Руководитель практики от ФГАОУ ВО «СПбПУ»:

Александрова Ольга Всеволодовна, Старший преподаватель ВШПИ

(Ф.И.О., уч.степень, должность)

Руководитель практики от профильной организации:

(Ф.И.О., должность)

Оценка:

Руководитель практики
от ФГАОУ ВО «СПбПУ»: /Ф.И.О./

Руководитель практики
от профильной организации:/Ф.И.О./

Обучающийся: /Ф.И.О./

Дата:

Оглавление

Техническое задание

Разработать программу «Судоку», где пользователь сможет вводить значения головоломки, а Ваша программа будет ее решать и выдавать результирующую таблицу.

Игровое поле представляет собой квадрат размером 9×9, разделённый на меньшие квадраты со стороной в 3 клетки. Таким образом, всё игровое поле состоит из 81 клетки. В них уже в начале игры стоят некоторые числа (от 1 до 9), называемые *подсказками*. Требуется заполнить свободные клетки цифрами от 1 до 9 так, чтобы в каждой строке, в каждом столбце и в каждом малом квадрате 3×3 каждая цифра встречалась бы только один раз.

Подход к решению

Нужно создать простой интерфейс где отображается сетка “судоку“ и пользователь может вводить либо изменять значения с помощью клавиатуры или мыши. С использованием мыши пользователь может щелкнуть любую из 81 ячейки. Выбранную ячейку будет визуально отображатся в программе. Можно будет задать значение нажимая соответствующей клавиши (1-9). Должна быть кнопка “решить“ при нажатие которой решается данная комбинация т.е. заполняются все оставщиеся пустые ячейки цифрами от 1 до 9 в соответствие с правилам игры. Так же должна быть и кнопка “сброс“ при нажатие которой сетка очищается, и заново можно вводить новые значения головоломки.

Интерфейсные элементы можно будет построить с использованием фреймворка Qt.

Чтобы дойти до решение головоломки можно использовать метод – бэктрекинг.

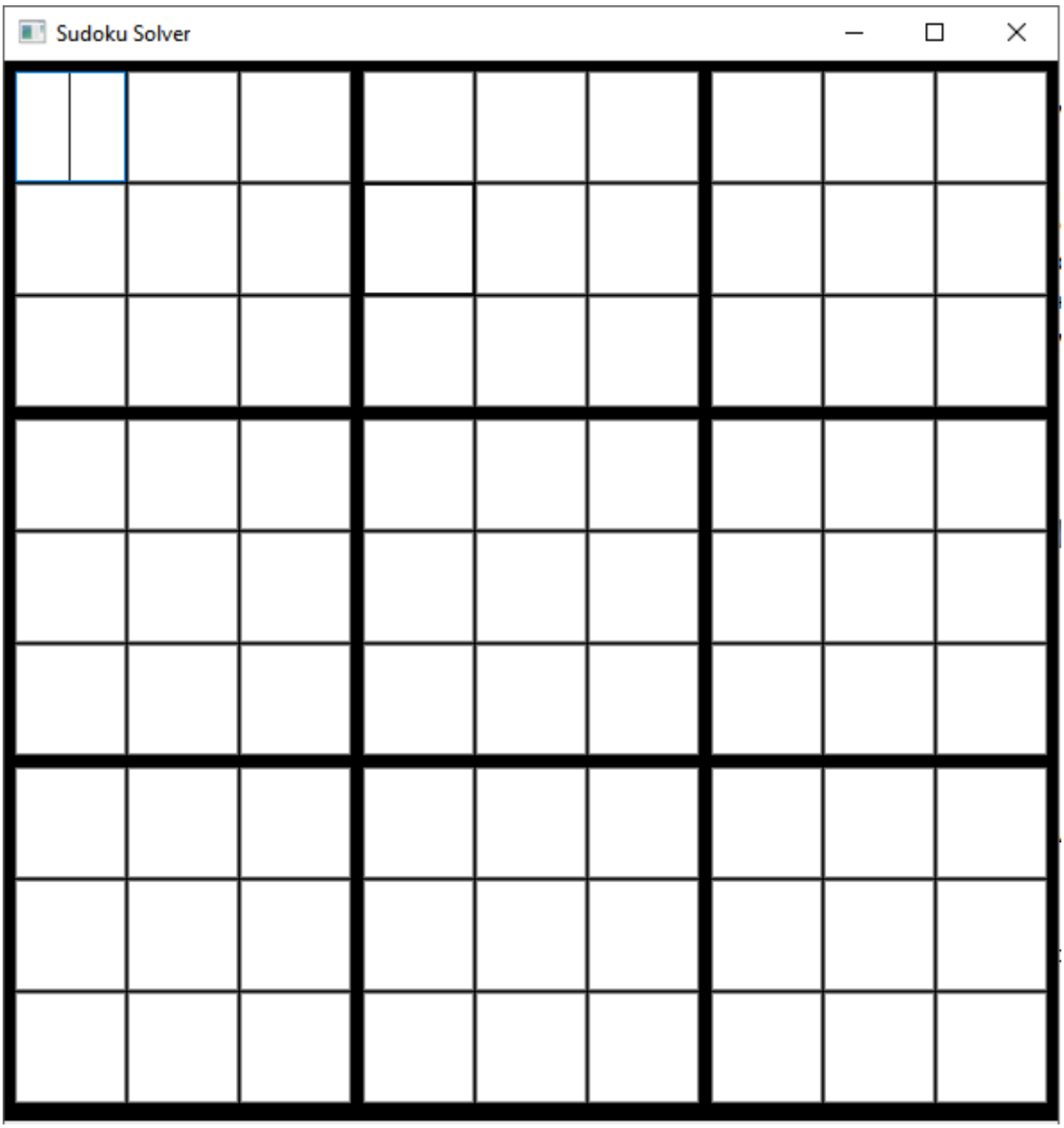
Поиск с возвратом, бэктрекинг (англ. *backtracking*) — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве M. Как правило позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Есть ли способ …», «Существует ли объект…» и т. п.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Данный метод нужно будет оптимизировать так как существует огромное количество комбинаций 81 цифр т.е. компютер долго будет искать решение. Исходя из правила игры можно будеть существенно уменьшить пространство поиска.

Реализация

Интерфейс построен таким образом что есть основной QGridLayout размера 3x3, который содержит малые квадраты 3x3 где разрешаетсея каждая цифра только один раз. Эти квадраты также представлени в виде QGridLayout размера 3x3 каждый элемент которого является текстовое поле, где можно всавить/писать цифры. Основной модуль интерфейса, который отображает окно содержающее все элементы – это MainWindow, класс который унаследован от QWidget. Вставляя черный фон, меняя margin и spacing каждого QGridLayout получаем оригинальная сетка “судоку“.



Таким образом пользователю дается возможность вводить/менять значения головоломки. Единственно нужно сделать так что разрешалось вводить лишь один символ в ячейку, и разрешалось только цифры 1-9. Также хотелось бы иметь возможность переходить на соседные ячейки используя клавиши со стрелками. Для это унаследум класс QLineEdit и назовем его GridCellEdit , в сам конструктор задаем нужные параметры:

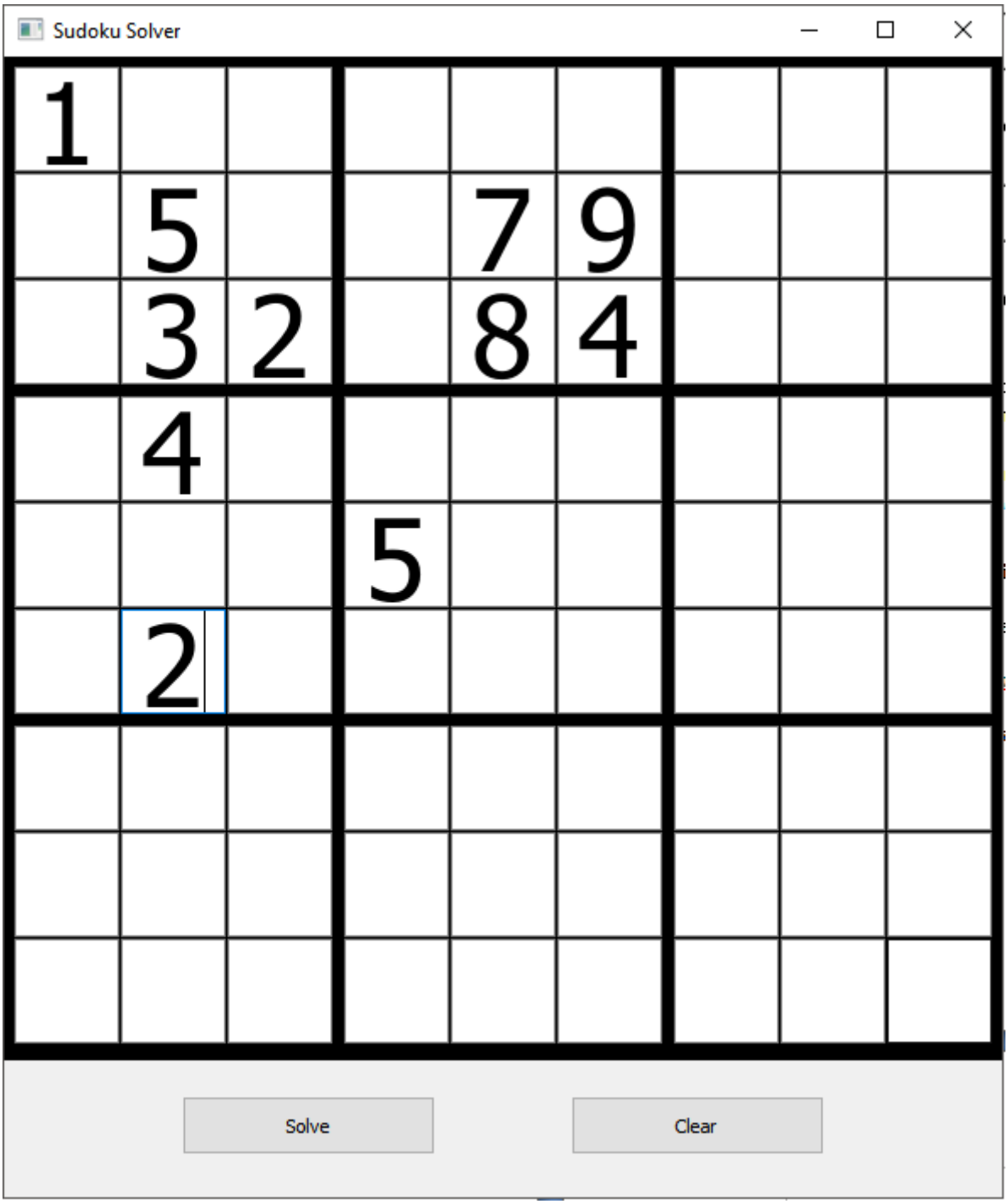
```
setMaxLength(1);
setValidator(new QIntValidator(1, 9, this));
```

И переопределяем метод захват события нажатие на клавиш:

```
virtual void keyPressEvent(QKeyEvent *event) override;
```

Имея ссылка на матрица которая содержит все 81 элементы GridCellEdit, используем setFocus() чтобы перейти на соседный элемент, в зависимости от нажатая клавиша со стрелками.

Добавляем еще кнопки “Solve“ и “Clear“ и интерфейс завершен.



В качество контейнер – матрица, используется:

```
template<typename T>
using Matrix = std::vector< std::vector< T >>;
```

Это упрощает создание аналоги функции std::transform, std::for_each только для матриц, вместо массивы.

```
template<typename T, typename U, class UnaryOperation>
void transform_2d(Matrix<T> &a, Matrix<U> &b,
UnaryOperation op)


template<typename T, class UnaryOperation>
void for_each_2d(Matrix<T> &m, UnaryOperation op)
```

Такие функции позволяют значительно сократить и упростить код программы.

При нажатие кнопки “Solve“ отправляем сетка “судоку“ в виде целочисленная матрица решателю – экземпляр класса SudokuSolver.

Перед начало решения, нужно сделать проверка – соответствует ли заданная пользователем матрица с правилам игры.

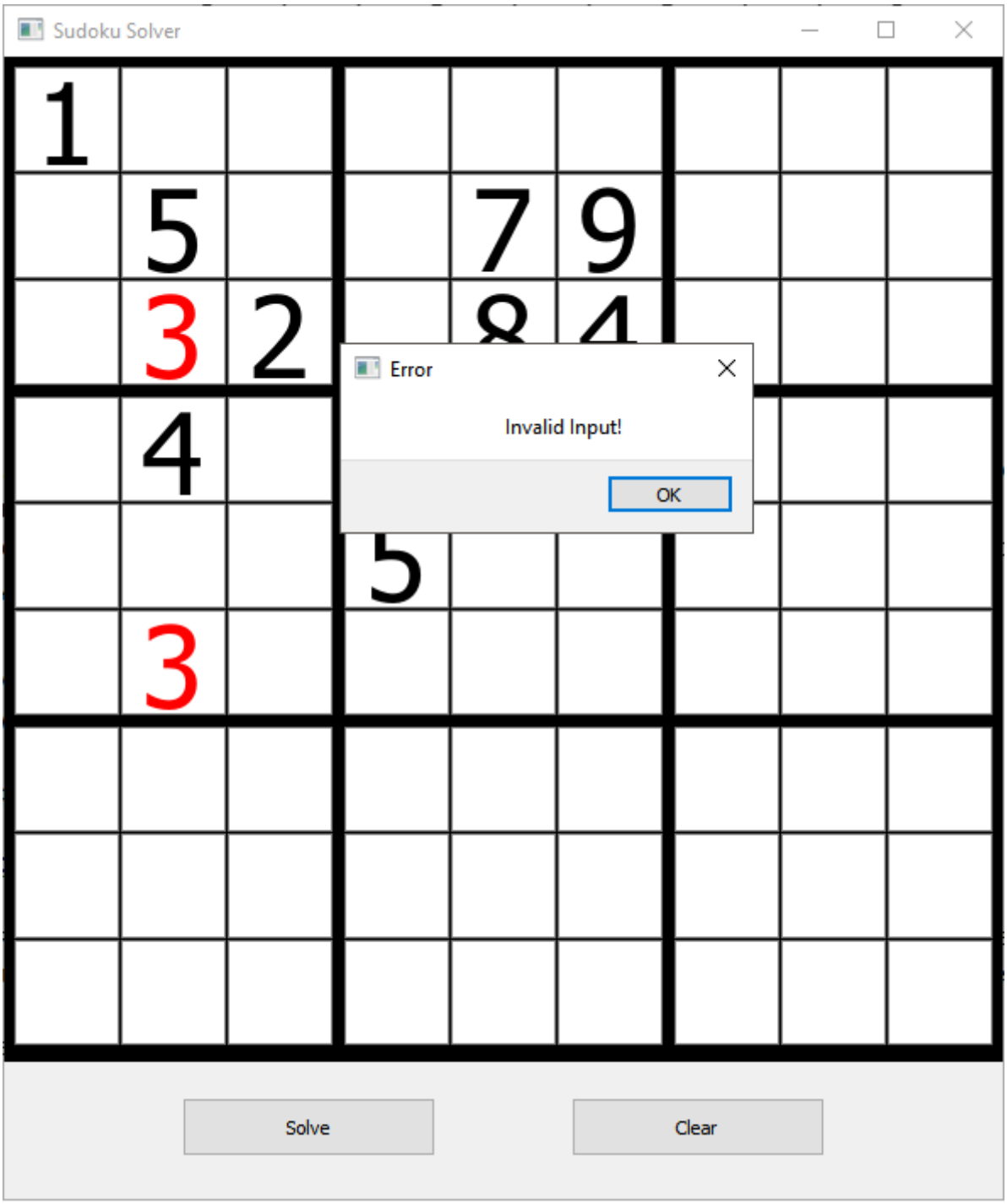
Создаем Матрицы int inCol[9][10], inRow[9][10], inRegion[9][10], делаем пробег по матрица на вход, и считам сколько из каждой цифры в строке, столбец и регион (3x3 квадрат). Области нумеруются таким образом, что верхний левый индексируется как 0, соседный на право – 1, нижный правый – 8.

Чтобы узнать в какой регион находится ячейка с индексы (i, j) в полная матрица 9x9, используем формула: (i/3)*3 + (j/3)

Если одна и та же цифра встречается более одного раза в строка, столбец либо регион – то это значит что заданная пользователем матрица не соответствует с правилам игры – её нужно исправить.

Легко узнать какие именно ячейки не соовмещаются – те ячейки, внутренная цыфра которых встречается больше чем раз, в соответствующая строка, столбец либо регион. В таких ячейках записываем “-1“ чтобы потом можноп было визуельно отобразить не соответствующие ячейки.

Если такие неправильности найдены – программа выдает дополнительное окно с сообщением “Invalid Input“, а также меняет цвет несоответствующих цифр красный. Это выгледит так:



Алгоритм

Создаем Матрицы для соблюдение правила игры bool inCol[9][10], inRow[9][10], inRegion[9][10], где левый индекс означает номер строка/столбец/регион, а правый индекс – цифра т.е. эти матрици отвечают на вопрос от типа “стоит ли цифра 5 в строка/столбец/регион 3?“. Эти матрици легко первоначально заполнить, делается аналогично тому как это делалось в проверка правильности матрицы.

Для простоты используются следующие вспомогательные функции:

1. Поставить цифра ‘num’ в ячейка с индексы i, j и запомнить что в i-строка, j-столбец и соответствующий регоион есть цифра ‘num’

```
void SudokuSolver::setCell(int i, int j, int num)
{
  grid_[i][j] = num;
  inRow[i][num] = true;
  inCol[j][num] = true;
  inRegion[(i/3)*3 + (j/3)][num] = true;
}
```

2. Удалить содержимое ячейки с индексами i, j и запомнить что в i-строка, j-столбец и соответствующий регоион больше такая цифра нет.

```
void SudokuSolver::resetCell(int i, int j)
{
  int num = grid_[i][j];
  inRow[i][num] = false;
  inCol[j][num] = false;
  inRegion[(i/3)*3 + (j/3)][num] = false;
  grid_[i][j] = 0;
}
```

3. Отвечает на вопрос “Можно ли поставить цифра ‘num’ в ячейка с индексы i, j без нарушения правил игры?“

```
bool SudokuSolver::isValidToPlace(int i, int j, int num)
{
  if(inRow[i][num] || inCol[j][num] || inRegion[(i/3)*3 +
(j/3)][num])
    return false;
  return true;
}
```

Първоначално заполнение матрицы содержимого строк/столбцов/регионов:

```
for(int i=0;i<9;i++)
  for(int j=0;j<9;j++)
    if(grid_[i][j] != 0) // 0 – пустая ячейка
      setCell(i, j, grid_[i][j]);
```

Решение основного проблема - с начало в верхняя левая ячейка, ищем пустая (не заполненная) ячейка, пробегая по матрицу на право в строку. Если дойдем до коцу строки, тогда продолжаем на следующую строку. Когда пустая ячейка найдена - ищем цифра которая можно там поставить (пробуем цифры 1-9), не ломая соответствие с правилам игры. Это проверяем с использование функцию **`isValidToPlace(int i, int j, int num)`**. Когда такая цифра найдена, её там зафиксируем используя **`setCell(int i, int j, int num)`** и продолжаем искать следующая пустая (не заполненная) ячейка в то же самое направление. Если в данная пустая ячейка не сможем зафиксировать никакая цифра, это значит что не правильно выбрана одна или несколько цифр в предыдущих ячеек. В такая ситуация делаем “бэктрекинг“ т.е. возвращаемся к предыдущей ячейке, где была зафиксированна некоторая цифра, удаляем содержимое с использованием функции **`resetCell(int i, int j)`**, и пытаемся зафиксировать другая цифра. Если такая цифра нет, то опять возвращаемся к предыдущему выбору. Если успеем зафиксировать цифра в нижняя правая ячейка – тогад решение головоломки найдено. По такому принципу работает метод “бэктрекинг“.

Сама функция решения выгледит так:

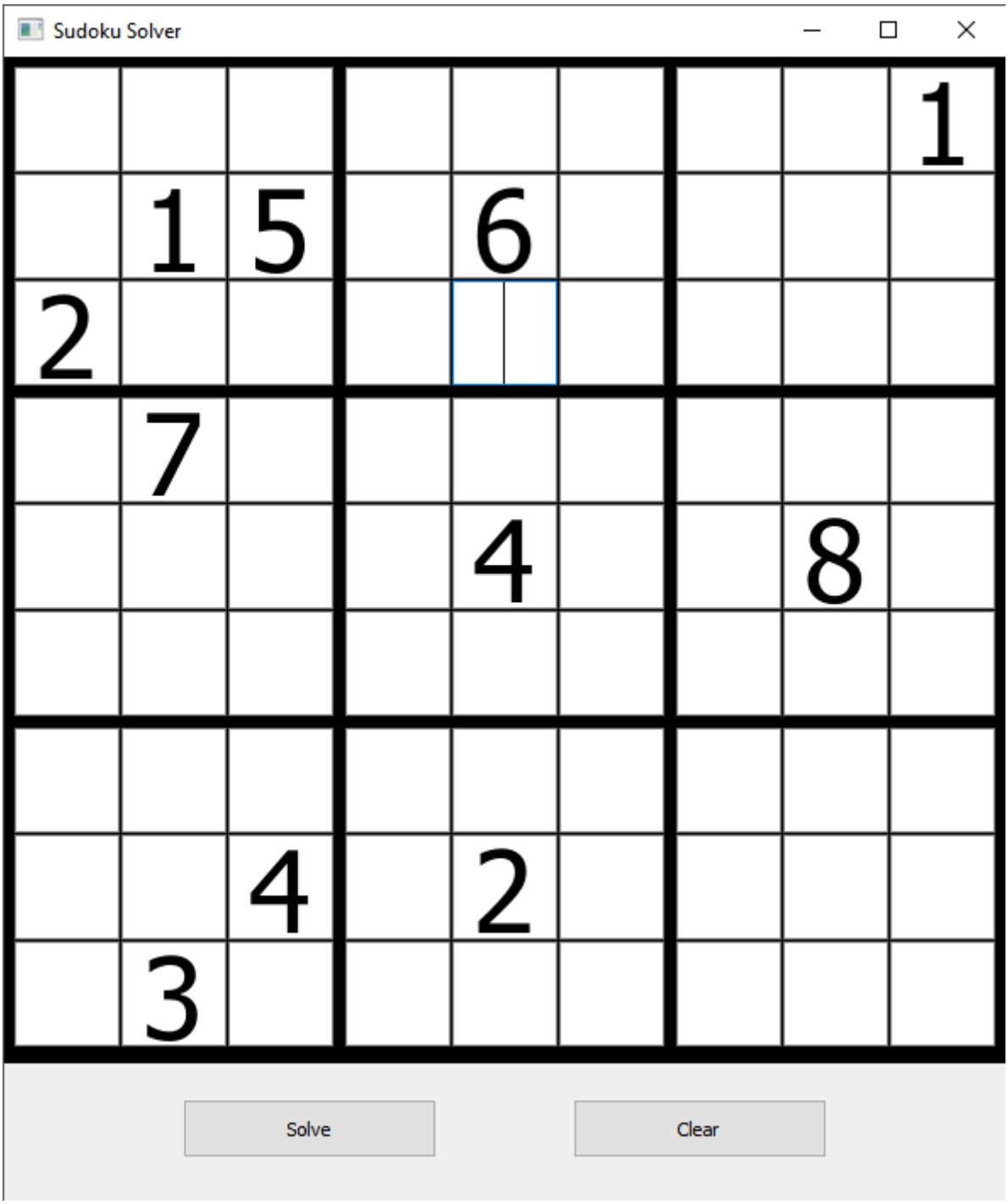
```
bool SudokuSolver::searchSolution(int i, int j) //i,j–текущая ячейка
{
  if(i > 8)
    return true; //решение найдено, нижняя правая ячейка пройдена
  if(j > 8)
    return searchSolution(i+1, 0) //переход к следующую строку
  if(grid_[i][j] != 0) //если ячейка не пустая
    return searchSolution(i, j+1); //переход к следующая ячейка

  for(int k=1; k<=9; k++) //пробуем цифры 1-9
  {
    if(isValidToPlace(i, j, k)) //можно ли сдесь поставить цифра ‘k’
    {
      setCell(i, j, k); //сдесь зафиксируем цифра ‘k’
```

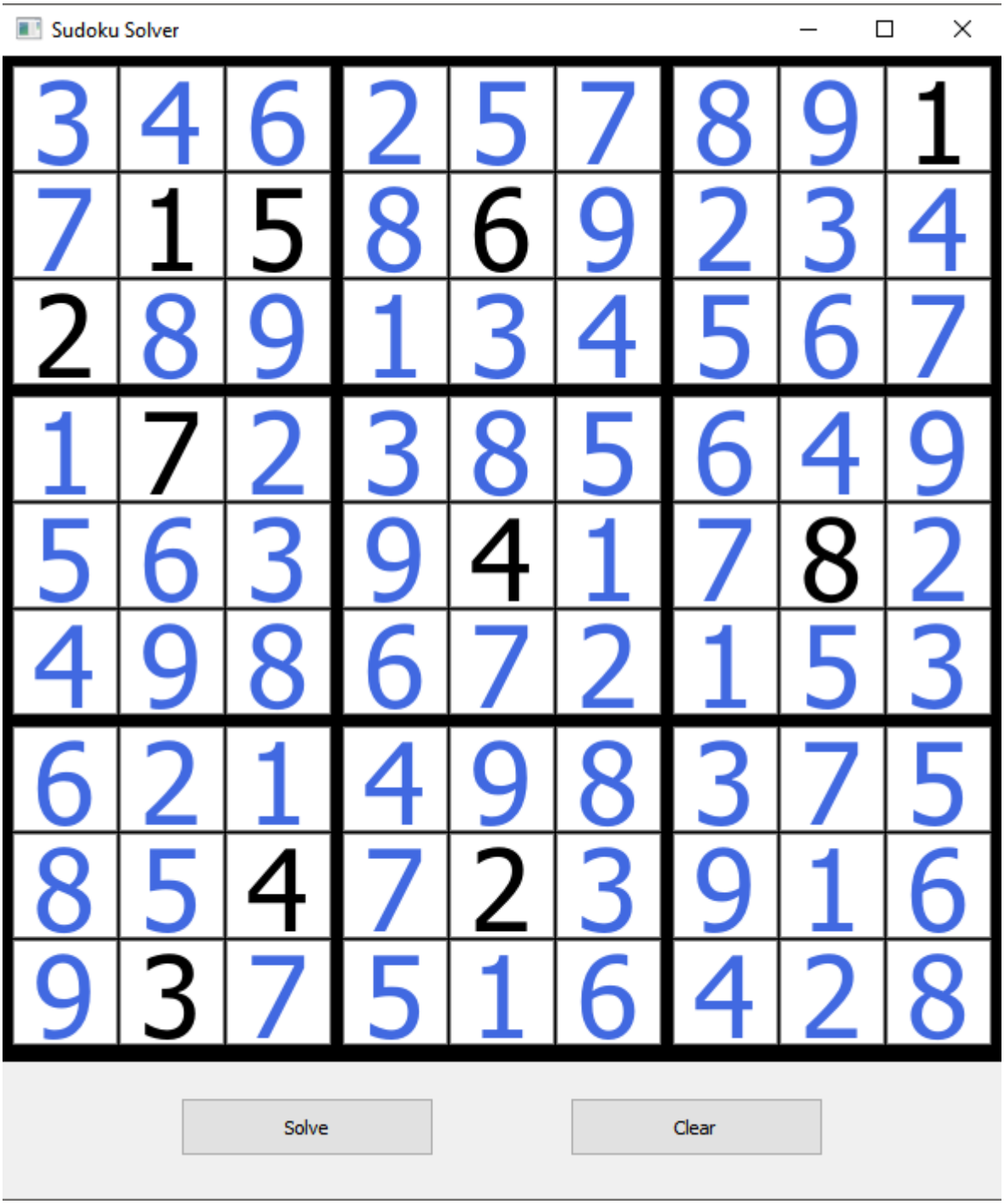
```
      if(searchSolution(i, j+1)) //переход к следующая ячейка
        return true; // если решение найдено, то выход из функции с
                     // результатом ‘true’
      resetCell(i, j); //удаляем зафиксированна цифра
    }
  }
  return false; //бэктрек
}
```

Результаты

После запуска программы отркывается окно решателя головоломки “судоку“. Отображается сетка из 9x9 квадратов, где пользователь может вводит значения.

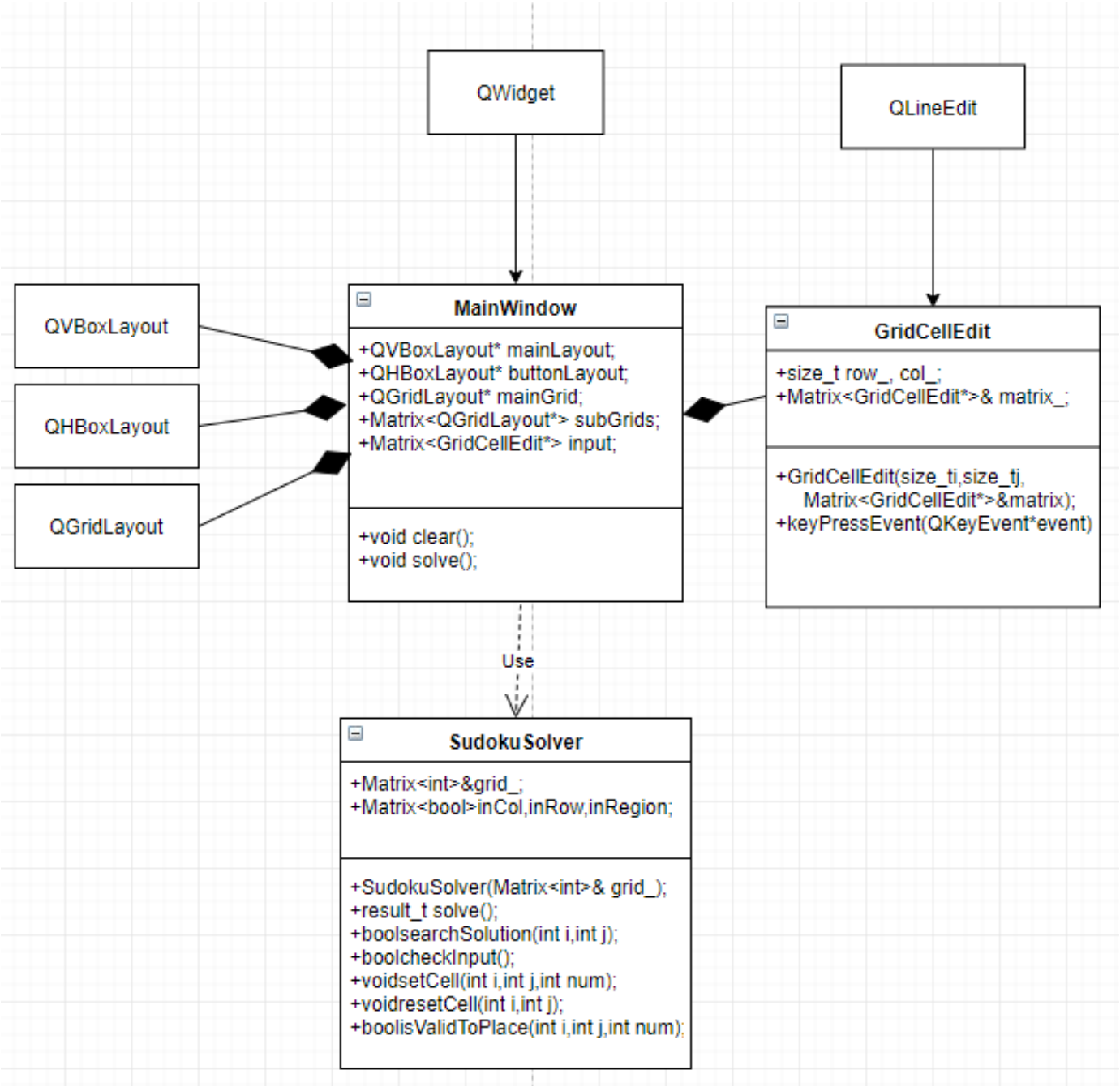


При нажатие кнопку “Solve“, программа находит решение заданного проблема и правильно заполняет пустые ячейки.



При нажатие кнопку “Clear“, сетка очищается, затем заново можно вводить новые значения.

Диаграмма классов



Вывод

В результате проведенной работы была разработана программа «Судоку», где пользователь может вводить значения головоломки, а программа ее решает и выдает результирующую таблицу. Были использованны методы объектно-ориентированного программирования, с применением библиотеки Qt. Был создан простой интерфейс где отображается сетка «Судоку» и пользователь может вводить либо изменять значения с помощью клавиатуры или мыши. Предоставлена возможность при нажатие на кнопку “решить“ получить решение для заданная комбинация, а так же кнопка “сброс“ при нажатие которой сетка очищается, и заново можно вводить новые значения головоломки. Так же реализована проверка на правильность ввода – неправильности визуельно отображаются. Само решение головоломки получается с использованием метода - поиск с возвратом, бэктрекинг.

Приложение (текст программы)

GridCellEdit.h

```
#ifndef GRIDCELLEDIT_H
#define GRIDCELLEDIT_H

#include <QLineEdit>
#include <QKeyEvent>
#include <vector>
#include "matrix.h"

class GridCellEdit : public QLineEdit
{
public:

  virtual ~GridCellEdit() = default;

  GridCellEdit(size_t i, size_t j, Matrix<GridCellEdit*> &matrix);

protected:
  virtual void keyPressEvent(QKeyEvent *event) override;

private:
  size_t row_, col_;
  Matrix<GridCellEdit*> &matrix_;
};

#endif // GRIDCELLEDIT_H
```

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QGridLayout>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <vector>
#include "gridcelledit.h"
#include "matrix.h"

class MainWindow : public QWidget
{
  Q_OBJECT

public:
  MainWindow();
  virtual ~MainWindow();

  QVBoxLayout *mainLayout;
  QHBoxLayout *buttonLayout;
  QGridLayout *mainGrid;

  Matrix<QGridLayout*> subGrids;
  Matrix<GridCellEdit*> input;

public slots:
  void clear();
```

```
  void solve();

private:
  void setupGrid();

};

#endif // MAINWINDOW_H
```

SudokuSolver.h

```
#ifndef SUDOKUSOLVER_H
#define SUDOKUSOLVER_H

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

class SudokuSolver
{
public:

  enum result_t{INVALID_INPUT, NO_SOLUTION, SOLVED};

  SudokuSolver(Matrix<int> &grid_);
  virtual ~SudokuSolver() = default;

  result_t solve();

private:
  Matrix<int> &grid_;
  Matrix<bool> inCol, inRow, inRegion;

  bool searchSolution(int i, int j);
  bool checkInput();
  void setCell(int i, int j, int num);
  void resetCell(int i, int j);
  bool isValidToPlace(int i, int j, int num);

};

#endif // SUDOKUSOLVER_H
```

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <vector>
#include <algorithm>
#include <stdexcept>

template<typename T>
using Matrix = std::vector< std::vector< T >>;

template<typename T, typename U, class UnaryOperation>
void transform_2d(Matrix<T> &a, Matrix<U> &b, UnaryOperation op)
{
  if(a.size() != b.size())
    throw std::invalid_argument("Matrix size does not match");
```

```
  for(int i=0; i<a.size(); i++)
  {
    std::transform(a[i].begin(), a[i].end(), b[i].begin(), op);
  }
}

template<typename T, class UnaryOperation>
void for_each_2d(Matrix<T> &m, UnaryOperation op)
{
  for(int i=0; i<m.size(); i++)
  {
    std::for_each(m[i].begin(), m[i].end(), op);
  }
}


#endif // MATRIX_H
```

GridCellEdit.cpp

```
#include "gridcelledit.h"
#include <QIntValidator>

GridCellEdit::GridCellEdit(size_t i, size_t j, Matrix<GridCellEdit*>
&matrix):
  matrix_(matrix),
  row_(i),
  col_(j)
{

  static const int width = 62;
  setFixedSize(width,width);
  setAlignment(Qt::AlignCenter);
  setMaxLength(1);
  setValidator(new QIntValidator(1, 9, this));

  static QFont font;
  font.setPointSize(50);
  setFont(font);

}

void GridCellEdit::keyPressEvent(QKeyEvent *event)
{

  switch(event->key())
  {
    case Qt::Key_Up:
      if(row_ > 0)
      {
        clearFocus();
        matrix_[row_-1][col_]->setFocus();
        matrix_[row_-1][col_]->setSelection(0,1);
      }
      break;
    case Qt::Key_Down:
      if(row_ < 8)
      {
        clearFocus();
        matrix_[row_+1][col_]->setFocus();
        matrix_[row_+1][col_]->setSelection(0,1);
      }
      break;
```

```
    case Qt::Key_Left:
      if(col_ > 0)
      {
        clearFocus();
        matrix_[row_][col_-1]->setFocus();
        matrix_[row_][col_-1]->setSelection(0,1);
      }
      break;
    case Qt::Key_Right:
      if(col_ < 8)
      {
        clearFocus();
        matrix_[row_][col_+1]->setFocus();
        matrix_[row_][col_+1]->setSelection(0,1);
      }
      break;
    default:
      QLineEdit::keyPressEvent(event);
      break;
  }
}
```

MainWindow.cpp

```
#include "mainwindow.h"
#include <QIntValidator>
#include <QPalette>
#include <QColor>
#include <QMessageBox>
#include "sudokusolver.h"
#include "matrix.h"

MainWindow::MainWindow()
{

  setWindowTitle("Sudoku Solver");

  QPixmap bkgnd(":/images/background.png");
  QPalette palette;
  palette.setBrush(QPalette::Background, bkgnd);
  setPalette(palette);

  setupGrid();

  buttonLayout = new QHBoxLayout();
  buttonLayout->setMargin(25);

  QPushButton *buttonSolve = new QPushButton("Solve");
  buttonSolve->setFixedSize(150,35);
  connect(buttonSolve, SIGNAL(released()), this, SLOT(solve()));
  buttonLayout->addWidget(buttonSolve);

  QPushButton *buttonClear = new QPushButton("Clear");
  buttonClear->setFixedSize(150,35);
  connect(buttonClear, SIGNAL(released()), this, SLOT(clear()));
  buttonLayout->addWidget(buttonClear);

  mainLayout = new QVBoxLayout();
  mainLayout->addLayout(mainGrid);
  mainLayout->addLayout(buttonLayout);
  mainLayout->setMargin(0);
  mainLayout->setSpacing(0);
  mainLayout->setSizeConstraint(QLayout::SetFixedSize);
```

```
  mainLayout->setAlignment(mainLayout, Qt::AlignLeft);
  setLayout(mainLayout);

}

void MainWindow::setupGrid()
{

  mainGrid = new QGridLayout();
  mainGrid->setMargin(5);
  mainGrid->setSpacing(5);

  subGrids.resize(3, std::vector<QGridLayout*>(3));
  input.resize(9, std::vector<GridCellEdit*>(9));

  for(int i=0; i<3; i++)
  {
    for(int j=0; j<3; j++)
    {

      subGrids[i][j] = new QGridLayout();
      subGrids[i][j]->setMargin(1);
      subGrids[i][j]->setSpacing(1);

      for(int k=0; k<3; k++)
      {
        for(int z=0; z<3; z++)
        {
          int curRow = i*3 + k;
          int curCol = j*3 + z;

          input[curRow][curCol] = new GridCellEdit(curRow, curCol, input);

          subGrids[i][j]->addWidget(input[curRow][curCol], k, z);

        }
      }
      mainGrid->addLayout(subGrids[i][j], i, j);
    }
  }
}

MainWindow::~MainWindow()
{

}

void MainWindow::clear()
{
  QPalette palette;
  palette.setColor(QPalette::ColorRole::Text, Qt::black);

  for_each_2d(input, [&](GridCellEdit* cell){
    cell->clear();
    cell->setPalette(palette);
  });

}

void MainWindow::solve()
{
  Matrix<int> grid(9, std::vector<int>(9));
  Matrix<bool> fromUser(9, std::vector<bool>(9, false));
```

```
  transform_2d(input, grid, [](GridCellEdit* cell){ return cell-
>text().toInt();});
  transform_2d(grid, fromUser, [](int num){return (num != 0 ? true :
false);});

  SudokuSolver solver(grid);
  SudokuSolver::result_t result = solver.solve();

  if(result == SudokuSolver::INVALID_INPUT)
  {
    QPalette palette;
    palette.setColor(QPalette::ColorRole::Text, Qt::red);

    for(int i=0;i<9;i++)
      for(int j=0;j<9;j++)
        if(grid[i][j] == -1)
          input[i][j]->setPalette(palette);

    QMessageBox messageBox;
    messageBox.setWindowTitle("Error");
    messageBox.setText("<p align='center'>Invalid Input!</p>");
    messageBox.setStyleSheet("QLabel{min-width: 200px;}");
    messageBox.exec();

    palette.setColor(QPalette::ColorRole::Text, Qt::black);

    for(int i=0;i<9;i++)
      for(int j=0;j<9;j++)
        if(grid[i][j] == -1)
          input[i][j]->setPalette(palette);

  }
  else if(result == SudokuSolver::NO_SOLUTION)
  {
    QMessageBox messageBox;
    messageBox.setWindowTitle("Error");
    messageBox.setText("<p align='center'>No solution could be
found!</p>");
    messageBox.setStyleSheet("QLabel{min-width: 200px;}");
    messageBox.exec();
  }
  else
  {
    QPalette palette;
    palette.setColor(QPalette::ColorRole::Text, QColor(65,105,225));

    for(int i=0;i<9;i++)
    {
      for(int j=0;j<9;j++)
      {
        input[i][j]->setText(QString::number(grid[i][j]));
        if(!fromUser[i][j])
          input[i][j]->setPalette(palette);
      }
    }
  }
}
```

SudokuSolver.cpp

```
#include "sudokusolver.h"
#include <iostream>
#include <algorithm>
#include <cstring>
```

```
SudokuSolver::result_t SudokuSolver::solve()
{

  if(!checkInput())
    return INVALID_INPUT;

  inCol.resize(9, std::vector<bool>(10, false));
  inRow.resize(9, std::vector<bool>(10, false));
  inRegion.resize(9, std::vector<bool>(10, false));

  for(int i=0;i<9;i++)
    for(int j=0;j<9;j++)
      if(grid_[i][j] != 0)
        setCell(i, j, grid_[i][j]);

  if(searchSolution(0,0))
    return SOLVED;
  else
    return NO_SOLUTION;

}

SudokuSolver::SudokuSolver(Matrix<int> &grid):
  grid_(grid)
{
}

bool SudokuSolver::searchSolution(int i, int j)
{
  if(i > 8)
    return true;

  if(j > 8)
    return searchSolution(i+1, 0);

  if(grid_[i][j] != 0)
    return searchSolution(i, j+1);

  for(int k=1; k<=9; k++)
  {
    if(isValidToPlace(i, j, k))
    {
      setCell(i, j, k);

      if(searchSolution(i, j+1))
        return true;

      resetCell(i, j);
    }
  }
  return false;

}

void SudokuSolver::setCell(int i, int j, int num)
{
  grid_[i][j] = num;
  inRow[i][num] = true;
  inCol[j][num] = true;
  inRegion[(i/3)*3 + (j/3)][num] = true;
}

void SudokuSolver::resetCell(int i, int j)
```

```
{
  int num = grid_[i][j];
  inRow[i][num] = false;
  inCol[j][num] = false;
  inRegion[(i/3)*3 + (j/3)][num] = false;
  grid_[i][j] = 0;
}

bool SudokuSolver::isValidToPlace(int i, int j, int num)
{
  if(inRow[i][num] || inCol[j][num] || inRegion[(i/3)*3 + (j/3)][num])
    return false;
  return true;
}
bool SudokuSolver::checkInput()
{
  Matrix<int> inCol(9, std::vector<int>(10, 0));
  Matrix<int> inRow(9, std::vector<int>(10, 0));
  Matrix<int> inRegion(9, std::vector<int>(10, 0));

  for(int i=0;i<9;i++)
  {
    for(int j=0;j<9;j++)
    {
      if(grid_[i][j] != 0)
      {
        inRow[i][grid_[i][j]]++;
        inCol[j][grid_[i][j]]++;
        inRegion[(i/3)*3 + (j/3)][grid_[i][j]]++;
      }
    }
  }

  bool isValid = true;
  for(int i=0; i<9; i++)
  {
    for(int j=0; j<9; j++)
    {
      int num = grid_[i][j];
      if((inRow[i][num] > 1) || (inCol[j][num] > 1)
         || (inRegion[(i/3)*3 + (j/3)][num] > 1))
      {
        isValid = false;
        grid_[i][j] = -1;
      }
    }
  }
  return isValid;
}
```

Main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  MainWindow w;
  w.show();

  return a.exec();
}
```