МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)
Факультет энергетический
Кафедра информатики, вычислительной техники и прикладной математики

КУРСОВАЯ РАБОТА

По дисциплине Теория автоматов

на тему: «Программная реализация конечных автоматов»

Выполнил студент гр. ВМК-21
Фролова А.И.

Проверил доцент кафедры ИВТ и ПМ
Коган Е.С.

Чита
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)

Факультет энергетический
Кафедра информатики, вычислительной техники и прикладной математики

ЗАДАНИЕ

на курсовую работу

По дисциплине: Теория автоматов

Студенту: Фроловой Анастасии Ивановне

Специальности: 09.03.01 Информатика и вычислительная техника

1 Тема курсовой работы: «Программная реализация конечных автоматов»

2 Срок подачи студентом законченной работы: 21 декабря 2024 г.

3 Исходные данные к работе: литературные источники, источники в сети интернет

4 Перечень подлежащих в курсовой работе вопросов:

а) Постановка и анализ задачи.
б) Анализ данных.
в) Программная реализация.
г) Техническое задание.
д) Руководство пользователя.
е) Пример работы программы.

5 Перечень графического материала (если имеется): -

Дата выдачи задания «7» сентября 2024г.

Руководитель курсовой работы_____/ Коган. Е.С. /

(подпись, расшифровка подписи)

Задание принял к исполнению

«8» сентября 2024г.

Подпись студента _____ / Фролова А.И./

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)
Факультет энергетический
Кафедра информатики, вычислительной техники и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе

по 09.03.01 Информатика и вычислительная техника

на тему: Программная реализация конечного автомата

Выполнил студент группы ВМК-21 Фролова Анастасия Ивановна

Руководитель работы: доцент кафедры ИВТ и ПМ Коган Евгения Семёновна

РЕФЕРАТ

Пояснительная записка – 29 страниц, иллюстраций – 11, приложения – 1.

КЛЕТОЧНЫЙ АВТОМАТ, СУДОКУ, КОНЕЧНЫЙ АВТОМАТ, C#

В работе описывается создание решателя Судоку как клеточного автомата. Реализация программы производится на высокоуровневом языке программирования C# с использованием платформы Windows Forms .NET.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Термин «автоматы» происходит от греческого слова «αὐτόματα», что означает «самодействующий». Автомат – это абстрактное самодостаточное вычислительное устройство, которое автоматически выполняет заданную последовательность операций. [4]

Для выполнения курсовой работы произведен анализ основных понятий теории автоматов, чтобы реализовать решатель Судоку. Необходимо решить, какие понятия нужны для реализации поставленной задачи.

1. Постановка и анализ задачи

1.1 Описание предметной области

Конечный автомат – это математическая модель, которая используется для описания систем, способных обрабатывать информацию и выполнять определенные действия в зависимости от входных данных. Программная реализация такого автомата позволяет создать систему, которая может обрабатывать информацию, выполнять определенные операции и принимать решения на основе заданных правил [1].

Судоку является по своей сути клеточным автоматом, основой которого является пространство из прилегающих друг к другу клеток (ячеек), образующих решётку. Каждая клетка может находиться в одном из конечного множества состояний (например, 1 и 0) [2].

По правилам игры в Судоку каждая клетка должна быть такое значение, что ни по горизонтали, ни по вертикали, ни в блоке три на три клетки не может быть клетки с таким же значением. Множество значений для клеток Судоку (а также состояний конечного автомата) будет {1, 2, 3, 4, 5, 6, 7, 8, 9, 12}, где 12 означает, что ячейка еще пустая и не была решена.

Реализованный код позволяет решить такую задачу, когда для каждой клетки существует лишь один претендент. Если на данном шаге претендентов несколько, то проверяются следующие пустые клетки. Программа работает до тех пор, пока не выяснится, что больше нет пустых ячеек (задача решена) либо во всех оставшихся пустых клетках есть несколько возможных решений. В таком случае для поиска ответа нужно использовать алгоритмы голых троек и четверок, указывающих пар и другие алгоритмы, не реализованные в курсовой работе.

1.2 Постановка задачи

Разработать программный инструмент (решатель Судоку), позволяющий генерировать случайную задачу по правилам игры в Судоку, проверять, существует ли однозначное решение такой задачи и производить вычисления, способствующие нахождению верного ответа.

1.3 Средства реализации

Для реализации выбрана интегрированная среда разработки программного обеспечения Microsoft Visual Studio с использование языка программирования высокого уровня C# и платформы Windows Forms .NET [3]. Внешний вид интерфейса указан на рисунке 1.

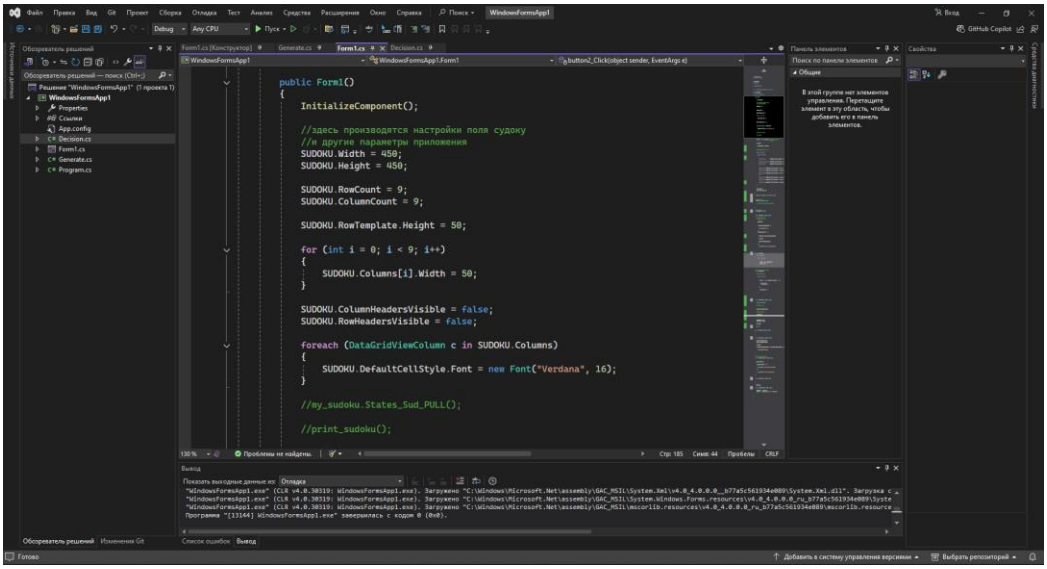


Рисунок 1 – интерфейс Microsoft Visual Studio

2 Анализ данных

Для работы программы можно выделить три типа данных:

a) входные данные, которые поступают в программу от пользователя;

b) промежуточные данные, которые используются в программе во время её выполнения;

c) выходные данные, которые программа выдает пользователю.

2.2 Входные данные

В качестве входных данных выступают:

a) количество известных ячеек в Судоку, если задача была сгенерированная программой;

b) головоломка, введенная пользователем с клавиатуры.

2.2 Выходные данные

В качестве выходных данных выступает головоломка, для которой было выявлено однозначное решение (если таковое имеется).

2.3 Промежуточные данные

Промежуточные данные используются во время решения задачи пользователю неизвестно. В список таких данных входит вспомогательная матрица для проверки решаемости головоломки, листы с флагами, нужные для поиска кандидатов на ячейку.

3 Программная реализация

3.1 Интерфейс программы.

В ходе разработки создан интерфейс конструктор сети Петри, в котором расположены следующие элементы:

1. Элемент *DataGrid*, благодаря которому происходит визуализация задачи.
2. Кнопка «*Сгенерировать*», по нажатию вызывается метод создания нового Судоку.
3. Элемент *trackBar*, позволяющий задать количество непустых ячеек Судоку.
4. Кнопка «*Очистить*» стирает задачу, которая была в поле *DataGrid*.
5. Кнопка «*Проверить решабельность*» позволяет узнать, есть ли у задачи однозначное решение в случае, если пользователь хочет сам попробовать решить задачу.
6. Кнопка «*Решить Судоку*» выводит пользователю решение, если таковое имеется.

Интерфейс решателя Судоку представлен на рисунке 2.

Код, выводящий графическое окно для взаимодействия с пользователем представлен в приложении А.

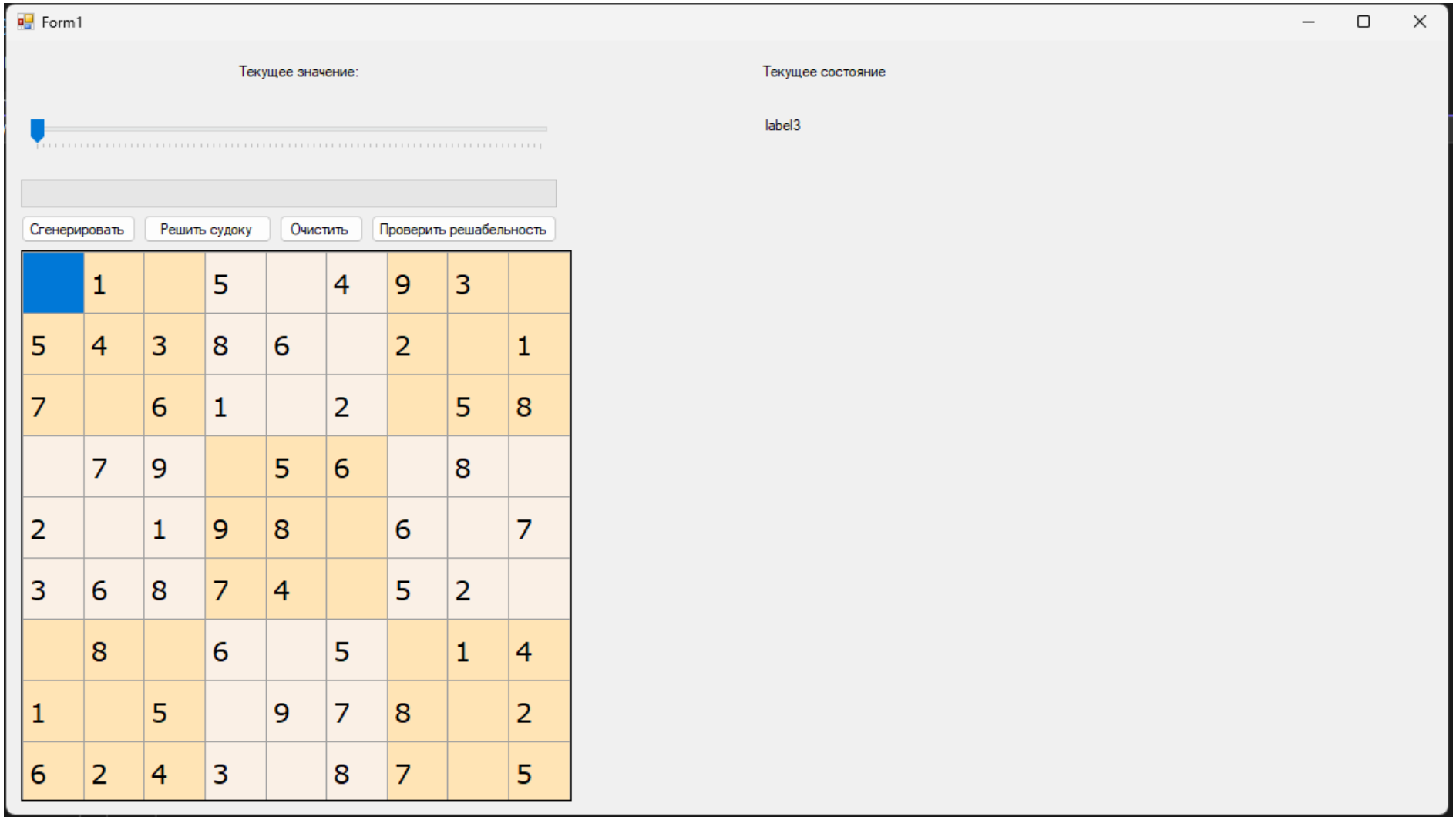


Рисунок 2 – Интерфейс программы

4 Техническое задание

4.1 Введение

Решатель Судоку. Условие задачи может задавать пользователь вводом с клавиатуры значений в поле формы, а также по нажатию кнопки «*Сгенерировать»*.

4.2 Требования к функциональным характеристикам

Приложение должно предоставлять:

a) возможность ввода условий задачи;

b) возможность генерации случайной задачи;

c) проверка и поиск однозначного решения Судоку;

d) поиск задачи с использованием различных методов решения Судоку

4.3 Требования к надежности

В программе должна присутствовать проверка входной информации при добавлении или редактировании Судоку. В случае возникновения ошибок предусмотреть возможность вывода пользователю сообщения об ошибки.

4.4 Требования к программной документации

Программная документация должна содержать руководство пользователя.

4.6 Требования к информационной и программной совместимости

Программа должна функционировать под управлением ОС семейства Windows не ниже версии 10.

5 Руководство пользователя

Программа решатель Судоку представляет из себя форму, на которой расположена таблица с значениями поля Судоку и кнопки, позволяющие взаимодействовать с задачей.

Для генерации Судоку нужно нажать кнопку “Сгенерировать”. Количество непустых элементов задачи будет равно тому значению, что выставил пользователь ползунком над полем задачи, что продемонстрировано на рисунке 3. Приведенная головоломка будет соответствовать правилам игры в Судоку.

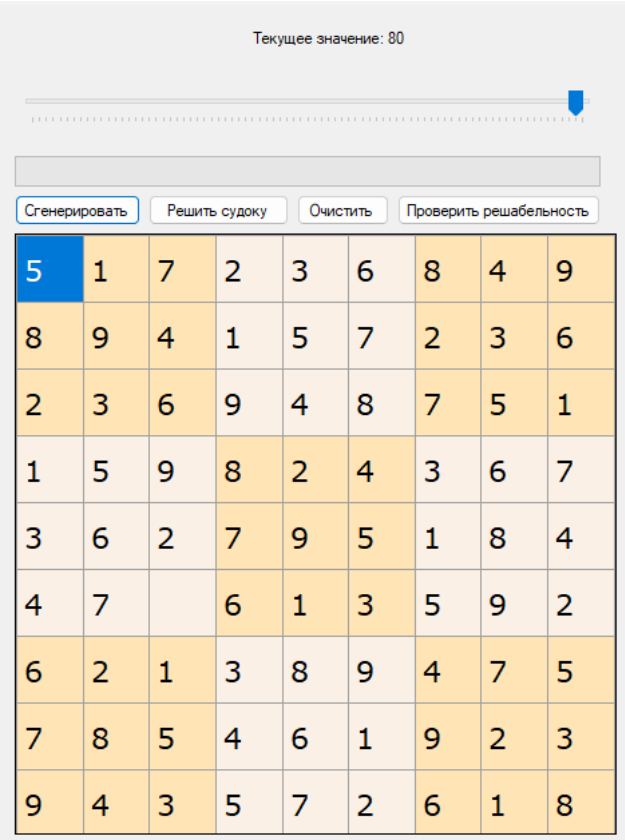


Рисунок 3 – Сгенерированная задача

Для очистки поля головоломки нужно нажать кнопку “Очистить”. Пользователь может сделать это когда ему понадобится ввести свою головоломку чтобы узнать решение или возможность его существования. Поле будет успешно очищено, а пользователь получит соответствующее уведомление (рисунок 4).

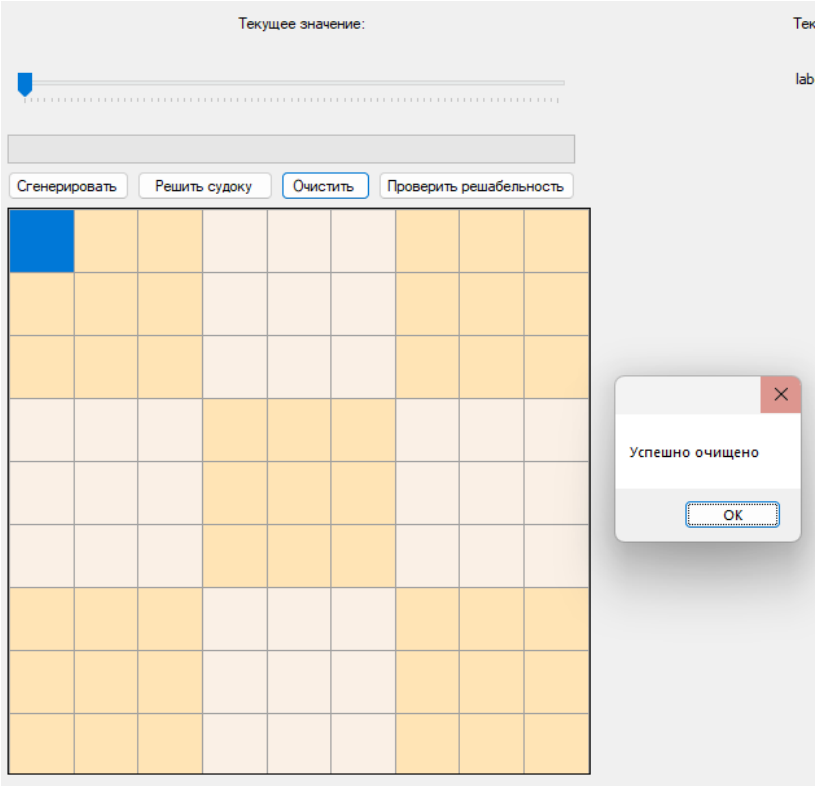


Рисунок 4 – Очистка поля Судоку

Для проверки решаемости задачи пользователю нужно нажать кнопку “Проверить решабельность”. В случае удачи на экране появится сообщение о том, что головоломка имеет решение (рисунок 5), иначе пользователь узнает, что решение не было найдено (рисунок 6).

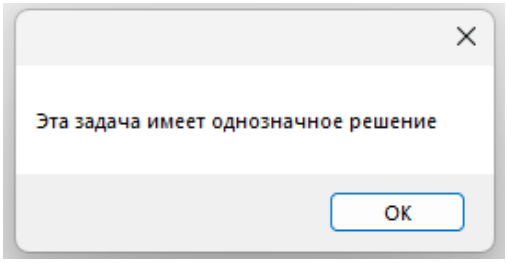


Рисунок 5 – Решение найдено

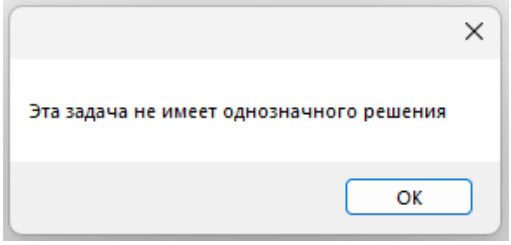


Рисунок 6 – Решение не найдено

6 Пример работы программы

По нажатию кнопки «*Сгенерировать*» задаются случайные условия задачи. Посмотрим результат генерации на рисунке 7.

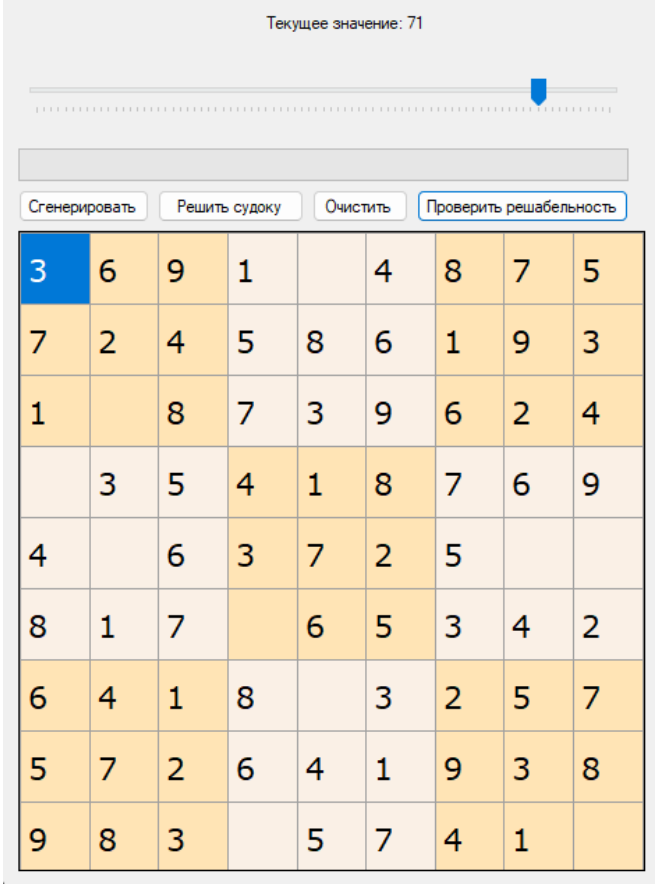


Рисунок 7 – Случайное Судоку

Как видим, число непустых клеток равно тому, что было установлено пользователем на ползунке. При попытке проверить, имеет ли задача решение, получим следующий результат:

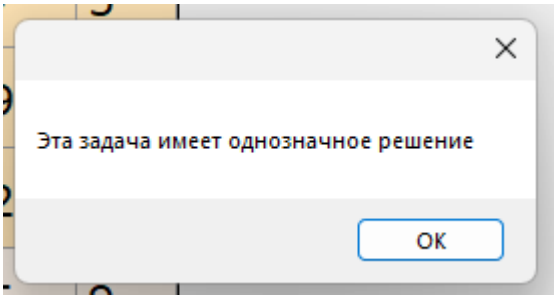
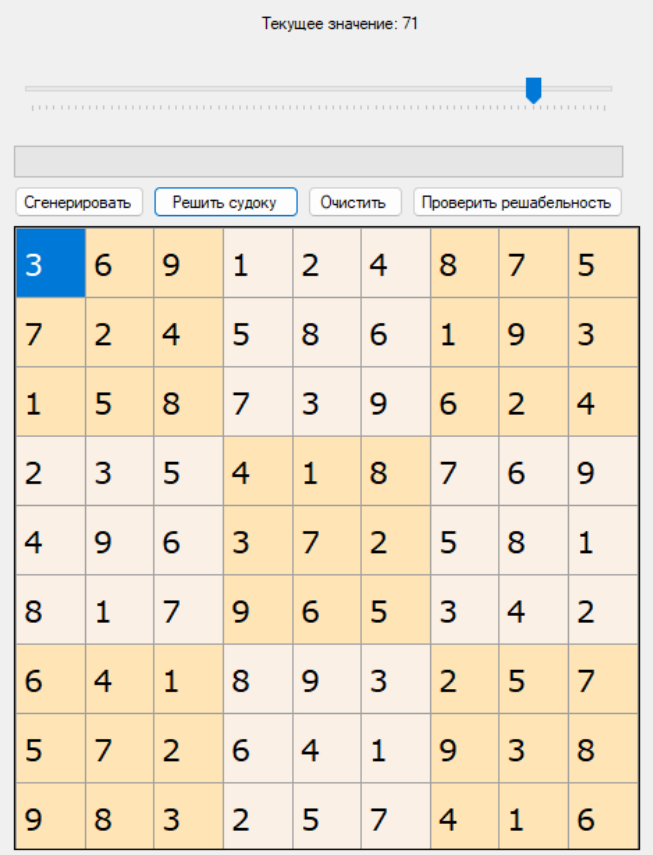


Рисунок 8. Задача имеет решение

По нажатию кнопки “Решить судоку” на поле выводится готовое решение (рисунок 9).



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

Рисунок 9 – Решенная задача

Ниже приведена часть кода, отвечающего за обработку нажатия на кнопку “Решить судоку”.

```
Ссылок: 1
private void Button_Solve(object sender, EventArgs e)
{
    //берем цифры записаные в поле
    //if (my_sudoku_flag == false)
    {
        NewSudokuPrint();

        my_sudoku.CanYouSolve();
        if (my_sudoku.isEMPTY(my_sudoku.MySud) == 81)
        {
            MessageBox.Show("Поле пустое!");
        }
        else
        if (my_sudoku.CanYouSolve() == true)
        {
            // проверяем глупые единицы
            my_sudoku.MySud = my_sudoku.Stupid_Unit(my_sudoku.MySud);

            print_sudoku();

            my_sudoku.ClearSUD(my_sudoku.minisud);
        }
        else
        {
            MessageBox.Show("Задача не имеет однозначного решения");
        }
    }
}
```

Рисунок 10 – Обработчик нажатия на кнопку “Решить судоку”

Если при генерации Судоку была сгенерирована задача с несколькими возможными решениями, об этом программа сообщит пользователю (рисунок 11).

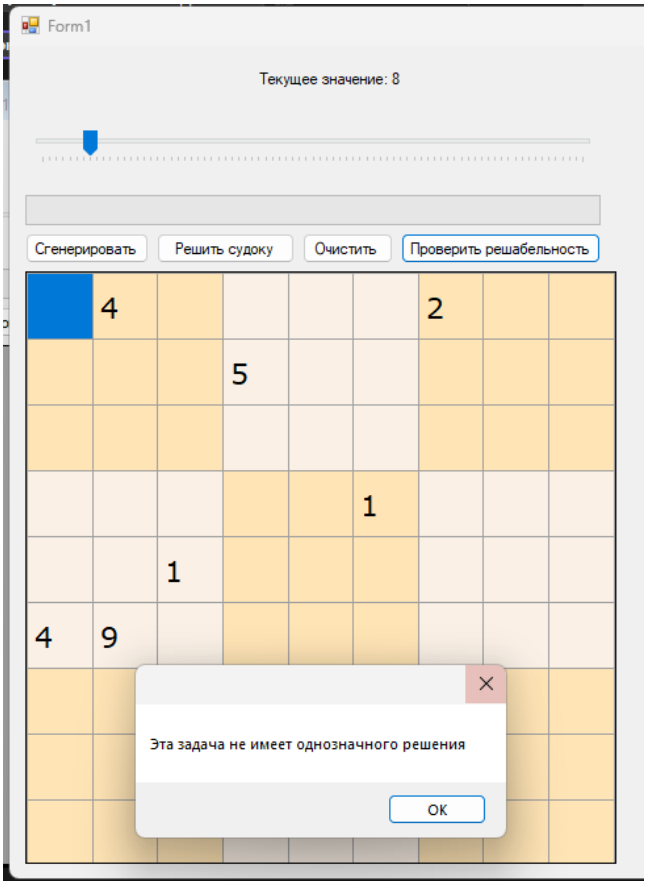


Рисунок 11 – Недостаточные условия

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была реализована программа - решатель Судоку, позволяющая находить решение для предложенной задачи, выяснять, возможно ли решить ее по правилам игры в Судоку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конечный автомат [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Конечный_автомат (дата обращения 19.10.2024)

2. Конечный автомат [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Клеточный_автомат (дата обращения 22.10.2024)

3. Краткий обзор языка C# [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (дата обращения 12.11.2024)

4. Теория автоматов. Краткое руководство [Электронный ресурс] Режим доступа: https://www.tutorialspoint.com/automata_theory/automata_theory_quick_guide.htm

Приложение А

```
using System;
using System.Collections.Generic;
using System.ComponentModel.Design;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms.VisualStyles;

namespace WindowsFormsApp1
{
    //класс решения судоку
    internal class Decision
    {
        public Decision() { }

        //поле судоку размером 9х9
        //нужен только для проверки решабельности
        public int[,] minisud =  {
        { 12, 5, 7, 12, 2, 3, 12, 1, 6 },
        { 1, 4, 12, 7, 12, 6, 8, 12, 2 },
        { 12, 3, 6, 9, 1, 8, 12, 5, 4 },

        { 5, 8, 1, 12, 9, 7, 6, 12, 3 },
        { 12, 6, 12, 5, 8, 4, 12, 9, 12 },
        { 4, 12, 2, 6, 12, 12, 5, 7, 8 },

        { 9, 2, 12, 12, 6, 5, 12, 8, 7 },
        { 3, 12, 5, 8, 12, 2, 1, 12, 12 },
        { 12, 1, 8, 12, 7, 12, 4, 2, 5 }
        };

        //то что задействовано в проге
        public int[,] MySud = {
        { 12, 5, 7, 12, 2, 3, 12, 1, 6 },
        { 1, 4, 12, 7, 12, 6, 8, 12, 2 },
        { 12, 3, 6, 9, 1, 8, 12, 5, 4 },

        { 5, 8, 1, 12, 9, 7, 6, 12, 3 },
        { 12, 6, 12, 5, 8, 4, 12, 9, 12 },
        { 4, 12, 2, 6, 12, 12, 5, 7, 8 },

        { 9, 2, 12, 12, 6, 5, 12, 8, 7 },
        { 3, 12, 5, 8, 12, 2, 1, 12, 12 },
        { 12, 1, 8, 12, 7, 12, 4, 2, 5 }
        };
```

```
        //обнуляем все судоку
        public void ClearSUD(int[,] sud)
        {
            for (int i = 0; i < 9; i++)
            {
                for (int j = 0; j < 9; j++)
                {
                    sud[i, j] = 12;
                }
            }
        }


        //листы флагов, позволяющие вычислить одно возможное число
        //int horizontal, vertical, square;
        List<int> flags = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9
};

        List<int> flags_h = new List<int>();
        List<int> flags_v = new List<int>();
        List<int> flags_s = new List<int>();

        //листы для попарного сравнения
        List<int> l1 = new List<int>();
        List<int> l2 = new List<int>();
        List<int> l3 = new List<int>();

        List<int> l1v = new List<int>();
        List<int> l1s = new List<int>();
        List<int> l2h = new List<int>();
        List<int> l2s = new List<int>();
        List<int> l3h = new List<int>();
        List<int> l3v = new List<int>();

        //лист с однозначными значениями
        List<int> l_res = new List<int>();

        //проверка на то не пустое ли поле
        public int isEMPTY(int[,] array )
        {
            int c = 0;
            for (int i = 0; i < 9; i++)
            {
                for (int j = 0; j < 9; j++)
                {
                    if (array[i, j] == 12)
                        c++;
                }
            }

            return c;
```

```
        }
        public bool CanYouSolve()
        {
            bool canSolve = true;
            // то се 5 10
            for (int i = 0; i < 9; i++)
            {
                for (int j = 0; j < 9; j++)
                {
                    minisud[i,j] = MySud[i,j];
                }
            }

            if ((isEMPTY(minisud) == 81)&(isEMPTY(MySud) == 81))
            {
                { canSolve = false; }
            }
            else
            {
                minisud = Stupid_Unit(MySud);

                //по факту тот же алгоритм что в глупых единицах
                for (int i = 0; i < 9; i++)
                {
                    for (int j = 0; j < 9; j++)
                    {
                        //на всякий пожарный
                        if (minisud[i, j] == 12)
                            { canSolve = false;}
                        //проверка вертикали
                        for (int c = 0; c < 9; c++)
                        {
                            //flags_v.Add(MySud[j, c]);
                            if ((minisud[i, j] == minisud[c, j]) & (i
!= c))
                            { canSolve = false;}
                        }

                        //проверка горизонтали
                        for (int c = 0; c < 9; c++)
                        {
                            //flags_h.Add(MySud[c, i]);
                            if ((minisud[i, j] == minisud[i, c]) & (j
!= c))
                            { canSolve = false; }
                        }

                        //проверка квадратика
                        //int a = id_square(i, j);
                        int k = 0, l = 0;
```

```
                        if ((j < 3) & (i < 3)) { k = 3; l = 3; }
                        else
                        if ((j < 6) & (i < 3)) { k = 6; l = 3; }
                        else
                        if ((j < 9) & (i < 3)) { k = 9; l = 3; }
                        else

                        if ((j < 3) & (i < 6)) { k = 3; l = 6; }
                        else
                        if ((j < 6) & (i < 6)) { k = 6; l = 6; }
                        else
                        if ((j < 9) & (i < 6)) { k = 9; l = 6; }
                        else

                        if ((j < 3) & (i < 9)) { k = 3; l = 9; }
                        else
                        if ((j < 6) & (i < 9)) { k = 6; l = 9; }
                        else
                        if ((j < 9) & (i < 9)) { k = 9; l = 9; }

                        //смотрим на конкретный квадратик где
обозревается ячейка

                        for (int ll = l - 3; ll < l; ll++)
                        {
                            for (int kk = k - 3; kk < k; kk++)
                            {
                                //flags_s.Add(MySud[kk, ll]);
                                if ((minisud[i, j] == minisud[ll, kk])
& (i != ll) & (j != kk))
                                { canSolve = false;}
                            }
                        }
                    }
                }
            }
            return canSolve;
        }

        public List<int> LIST_ENABLED(List<int> l1, List<int> l2)
        {
            List<int> l3 = new List<int>();

            int c = 0;

            for (int i = 0; i < l1.Count; i++)
            {
                for (int j = 0; j < l2.Count; j++)
                {
                    if (l1[i] == l2[j])
```

```
                        c++;
                    //если вышло так что на последнем шаге есть
единственный уникальный элемент
                    if ((j == l2.Count - 1) & (c == 0))
                    {
                        l3.Add(l1[i]);
                    }
                }
                c = 0;
            }
            return l3;
        }

        //отработал алгоритм глупой единицы
        //голые тройки
        bool trese = false;
        //голые четверки
        bool quatro = false;

        //алгоритм который я назвала глупым юнитом
        //есть ли в судоку такая клетка, в которую можно однозначно
записать лишь одно число?
        public int[,] Stupid_Unit( int[,] thissud)
        {
            //if (canSolve == true)
            {
                //проверяем каждую клеточку, чтобы совпало три условия
                for (int i = 0; i < 9; i++)
                {
                    for (int j = 0; j < 9; j++)
                    {
                        //j и i местами не менять!!!!
                        //если сейчас клетка пуста
                        if (thissud[j, i] == 12)
                        {
                            //проверка вертикали
                            for (int c = 0; c < 9; c++)
                            {
                                flags_v.Add(thissud[j, c]);
                            }

                            //проверка горизонтали
                            for (int c = 0; c < 9; c++)
                            {
                                flags_h.Add(thissud[c, i]);
                            }

                            //проверка квадратика
                            //int a = id_square(i, j);
                            int k = 0, l = 0;
```

```
                            if ((j < 3) & (i < 3)) { k = 3; l = 3; }
                            else
                            if ((j < 6) & (i < 3)) { k = 6; l = 3; }
                            else
                            if ((j < 9) & (i < 3)) { k = 9; l = 3; }
                            else

                            if ((j < 3) & (i < 6)) { k = 3; l = 6; }
                            else
                            if ((j < 6) & (i < 6)) { k = 6; l = 6; }
                            else
                            if ((j < 9) & (i < 6)) { k = 9; l = 6; }
                            else

                            if ((j < 3) & (i < 9)) { k = 3; l = 9; }
                            else
                            if ((j < 6) & (i < 9)) { k = 6; l = 9; }
                            else
                            if ((j < 9) & (i < 9)) { k = 9; l = 9; }

                            //смотрим на конкретный квадратик где
обозревается ячейка

                            for (int ll = l - 3; ll < l; ll++)
                            {
                                for (int kk = k - 3; kk < k; kk++)
                                {
                                    flags_s.Add(thissud[kk, ll]);
                                }
                            }

                            //теперь сравниваем листы чтобы получить
элементы которых нет в общем списке
                            //здесь проверяются цифры которых нет по
верт, горизонт и в квардрате
                            l1 = LIST_ENABLED(flags, flags_h);
                            l2 = LIST_ENABLED(flags, flags_v);
                            l3 = LIST_ENABLED(flags, flags_s);

                            //здесь нужно менять алгоритм
                            //попарно сравниваем места где также могут
быть повторы
                            l1v = LIST_ENABLED(l1, flags_v);
                            l1s = LIST_ENABLED(l1, flags_s);

                            l2h = LIST_ENABLED(l2, flags_h);
                            l2s = LIST_ENABLED(l2, flags_s);

                            l3h = LIST_ENABLED(l3, flags_h);
                            l3v = LIST_ENABLED(l3, flags_v);
```

```
                                int counter = 0;

                                //выявляем то самое особое число
                                for (int b = 0; b < l1v.Count(); b++)
                                {
                                    for (int c = 0; c < l1s.Count(); c++)
                                    {
                                        if (l1v[b] == l1s[c])
                                        {
                                            counter++;
                                        }
                                            //если это число встретилось во
всех пяти списках, значит оно нам надо
                                        if ((counter == 5))
                                        {
                                            l_res.Add(l1v[b]);
                                        }

                                    }

                                    for (int c = 0; c < l2h.Count(); c++)
                                    {
                                        if (l1v[b] == l2h[c])
                                        {
                                            counter++;
                                        }

                                        if (counter == 5)
                                        {
                                            l_res.Add(l1v[b]);
                                        }
                                    }

                                    for (int c = 0; c < l2s.Count(); c++)
                                    {
                                        if (l1v[b] == l2s[c])
                                        {
                                            counter++;
                                        }

                                        if (counter == 5)
                                        {
                                            l_res.Add(l1v[b]);
                                        }

                                    }
                                    for (int c = 0; c < l3h.Count(); c++)
                                    {
                                        if (l1v[b] == l3h[c])
                                        {
                                            counter++;
```

```
                                    }

                                    if (counter == 5)
                                    {
                                        l_res.Add(l1v[b]);
                                    }

                                }

                                for (int c = 0; c < l3v.Count(); c++)
                                {
                                    if (l1v[b] == l3v[c])
                                    {
                                        counter++;
                                    }
                                    if (counter == 5)
                                    {
                                        l_res.Add(l1v[b]);
                                    }
                                }

                            }
                            //элементы которые нигде не повторяются
                            //if (coouter_big > 0 | l_res[1] == 0 |
MySud[i, j] != 12)
                            //if (coouter_big > 0)
                            //if (proverka == false)
                            //if ((counter == 5)&(l_res[0] != 0))
                            if (l_res.Count != 0)
                            {
                                thissud[j, i] = l_res[0];
                                //SUDOKU.Rows[i].Cells[j].Value =
my_sudoku.translate_to_int(my_sudoku.array_states[j, i]);

//SUDOKU.Rows[i].Cells[j].Style.BackColor = Color.White;
                                //закончился проход по судоку
                                //break;

                                //stup_un = true; пригодится в будущем

                                flags_h.Clear();
                                flags_v.Clear();
                                flags_s.Clear();

                                l1.Clear();
                                l2.Clear();
                                l3.Clear();

                                l1v.Clear();
                                l1s.Clear();
                                l2h.Clear();
```

```
                                l2s.Clear();
                                l3h.Clear();
                                l3v.Clear();

                                l_res.Clear();
                            }
                            else break;

                        }
                    }
                }

            }

            return thissud;
        }
    }

}
```