ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

КОЛЛЕДЖ МНОГОУРОВНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ ОТДЕЛЕНИЕ ЭЛЕКТРИЧЕСКИХ СТАНЦИЙ И КОМПЬЮТЕРНЫХ СЕТЕЙ

Создание игры «Судоку».

Выполнила студентка

группы 22КС-20:

Смиркина Милена Михайловна

Проверил преподаватель:

Кукшева Байрта Анатольевна

Москва

2021

Содержание

Введение.

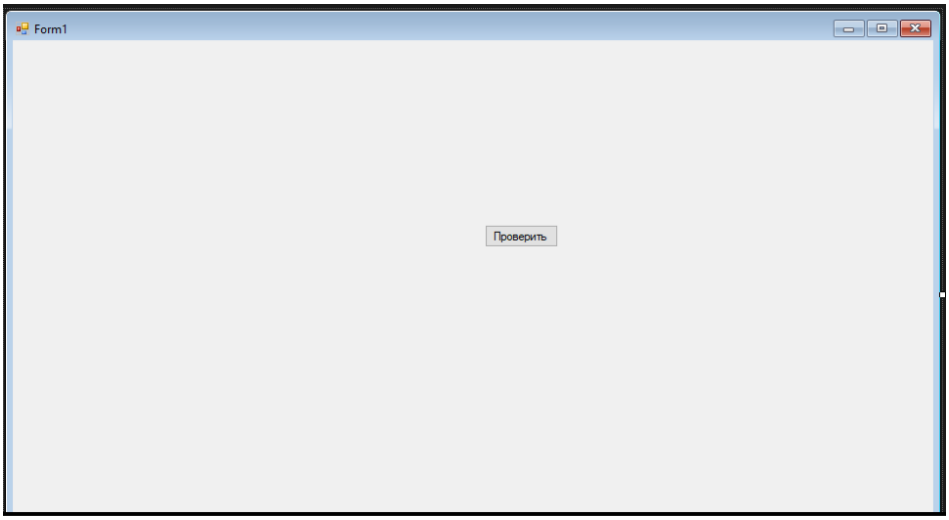
Классическое судоку - головоломка с цифрами. Судоку играется на игровом поле, состоящем из 9 на 9 клеток, всего 81 клетка. Внутри игрового поля находятся 9 "квадратов" (состоящих из 3 x 3 клеток). Каждая горизонтальная строка, вертикальный столбец и квадрат (9 клеток каждый) должны заполняться цифрами 1-9, не повторяя никаких чисел в строке, столбце или квадрате. Правила Судоку относительно несложные - но игра необычайно разнообразна, с миллионами возможных комбинаций чисел и широким диапазоном уровней сложности. Но все это основано на простых принципах использования чисел 1-9, заполнении пробелов на основе дедуктивного мышления и никогда не повторяющихся чисел в каждом квадрате, строке или столбце.

Компьютерная игра (иногда используется неоднозначный термин видеоигра) — компьютерная программа или часть компьютерной программы, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра. Компьютерные игры оказали столь существенное влияние на общество, что в информационных технологиях отмечена устойчивая тенденция к геймификации для неигрового прикладного программного обеспечения. Компьютерные игры также с 2011 года официально признаны правительством США и американским Национальным фондом отдельным видом искусства, наряду с театром, кино и т. п.

Так как в настоящее время дети и взрослые все больше проводят время за компьютером, мы решили совместить приятное с полезным и воспроизвести полюбившуюся многим игру на компьютере. Для выполнения поставленной задачи будет использоваться программа Microsoft Visual Studio.

Ход работы:

1. Создаем форму с кнопкой «Проверить», она понадобится нам для проверки ответа в будущем.



2. Прописываем поле для игры, которое состоит из 81 ячейки. Объявляем константу n, которая будет отвечать за размер ячейки. Используем формулу n*n для построения карты 9*9. Чтобы играла выполнялась правильно, рассчитываем формулу для расстановки чисел и прописываем ее на карту.
   Объявим функцию, на которой будем выполнять карту

```
namespace Sudoku
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        const int n = 3;
        const int sizeButton = 50;
        public int[,] map = new int[n * n, n * n];
        public Button[,] buttons = new Button[n * n, n * n];
```

3. Чтобы играла выполнялась правильно, рассчитываем формулу для расстановки чисел и прописываем ее на карту.

```
Ссылок: 2
public void GenerateMap()
{
    for (int i = 0; i < n * n; i++)
    {
        for (int j = 0; j < n * n; j++)
        {
            map[i, j] = (i * n + i / n + j) % (n * n) + 1;
            buttons[i, j] = new Button();
        }
    }
```

4. Создаем новую кнопку, указываем размер, необходимое значение и расположение и добавляем на форму.

```
Ссылок: 1
public void CreateMap()
{
    for (int i = 0; i < n * n; i++)
    {
        for (int j = 0; j < n * n; j++)
        {
            Button button = new Button();
            buttons[i, j] = button;
            button.Size = new Size(sizeButton, sizeButton);
            button.Text = map[i, j].ToString();
            button.Click += OnCellPressed;
            button.Location = new Point(j * sizeButton, i * sizeButton);
            this.Controls.Add(button);
        }
    }
}
```

5. Через двойной цикл меняем местами строки и столбцы в транспонировании матрицы и добавляем на карту.

```
Ссылок: 2
public void MatrixTransposition()
{
    int[,] tMap = new int[n * n, n * n];
    for (int i = 0; i < n * n; i++)
    {
        for (int j = 0; j < n * n; j++)
        {
            tMap[i, j] = map[j, i];
        }
    }
    map = tMap;
}
```

6. Прописываем функцию смены строк в пределах одной переменной, делая расстановку значений рандомным и не повторяющимся.

```
Ссылок: 1
public void SwapRowsInBlock()
{
    Random r = new Random();
    var block = r.Next(0, n);
    var row1 = r.Next(0, n);
    var line1 = block * n + row1;
    var row2 = r.Next(0, n);
    while (row1 == row2)
        row2 = r.Next(0, n);
    var line2 = block * n + row2;
    for (int i = 0; i < n * n; i++)
    {
        var temp = map[line1, i];
        map[line1, i] = map[line2, i];
        map[line2, i] = temp;
    }
}
```

7. Повторяем операцию со столбцами.

```
Ссылок: 1
public void SwapColumnsInBlock()
{
    Random r = new Random();
    var block = r.Next(0, n);
    var row1 = r.Next(0, n);
    var line1 = block * n + row1;
    var row2 = r.Next(0, n);
    while (row1 == row2)
        row2 = r.Next(0, n);
    var line2 = block * n + row2;
    for (int i = 0; i < n * n; i++)
    {
        var temp = map[i, line1];
        map[i, line1] = map[i, line2];
        map[i, line2] = temp;
    }
}
```

8. Добавляем функцию, меняющую блоки в пределах строк по вертикали и горизонтали.

```
Ссылок: 1
public void SwapBlocksInRow()
{
    Random r = new Random();
    var block1 = r.Next(0, n);
    var block2 = r.Next(0, n);
    while (block1 == block2)
        block2 = r.Next(0, n);
    block1 *= n;
    block2 *= n;
    for (int i = 0; i < n * n; i++)
    {
        var k = block2;
        for (int j = block1; j < block1 + n; j++)
        {
            var temp = map[j, i];
            map[j, i] = map[k, i];
            map[k, i] = temp;
            k++;
        }
    }
}
```

```
Ссылок: 1
public void SwapBlocksInColumn()
{
    Random r = new Random();
    var block1 = r.Next(0, n);
    var block2 = r.Next(0, n);
    while (block1 == block2)
        block2 = r.Next(0, n);
    block1 *= n;
    block2 *= n;
    for (int i = 0; i < n * n; i++)
    {
        var k = block2;
        for (int j = block1; j < block1 + n; j++)
        {
            var temp = map[i, j];
            map[i, j] = map[i, k];
            map[i, k] = temp;
            k++;
        }
    }
}
```

9. Прописываем функцию, которая перемешает все предыдущие функции.

```
Ссылок: 1
public void ShuffleMap(int i)
{
    switch (i)
    {
        case 0:
            MatrixTransposition();
            break;
        case 1:
            SwapRowsInBlock();
            break;
        case 2:
            SwapColumnsInBlock();
            break;
        case 3:
            SwapBlocksInRow();
            break;
        case 4:
            SwapBlocksInColumn();
            break;
        default:
            MatrixTransposition();
            break;
    }
}
```

10. С помощью функции “HideCells” скрываем некоторые клетки в рандомном порядке и делаем их неактивными.

```
Ссылок: 1
public void HideCells()
{
    int N = 40;
    Random r = new Random();
    while (N > 0)
    {
        for (int i = 0; i < n * n; i++)
        {
            for (int j = 0; j < n * n; j++)
            {
                if (!string.IsNullOrEmpty(buttons[i, j].Text))
                {
                    int a = r.Next(0, 3);
                    buttons[i, j].Text = a == 0 ? "" : buttons[i, j].Text;
                    buttons[i, j].Enabled = a == 0 ? true : false;

                    if (a == 0)
                        N--;
                    if (N <= 0)
                        break;
                }
            }
            if (N <= 0)
                break;
        }
    }
}
```

11. На рабочие клавиши прописываем код, позволяющий выбирать значение. При пустом тексте ставим 1, а с каждым нажатием увеличиваем значение на 1.

```
Ссылок: 1
public void OnCellPressed(object sender, EventArgs e)
{
    Button pressedButton = sender as Button;
    string buttonText = pressedButton.Text;
    if (string.IsNullOrEmpty(buttonText))
    {
        pressedButton.Text = "1";
    }
    else
    {
        int num = int.Parse(buttonText);
        num++;
        if (num == 10)
            num = 1;
        pressedButton.Text = num.ToString();
    }

}
```

12. Через двойной цикл формы пробегаем по кнопкам, сравнивая значения с значением, которое хранит состояние карты, проверяем на несоответствия, в случае их присутствия выводим «Неверно!», если все соответствует, выводим «Верно!».

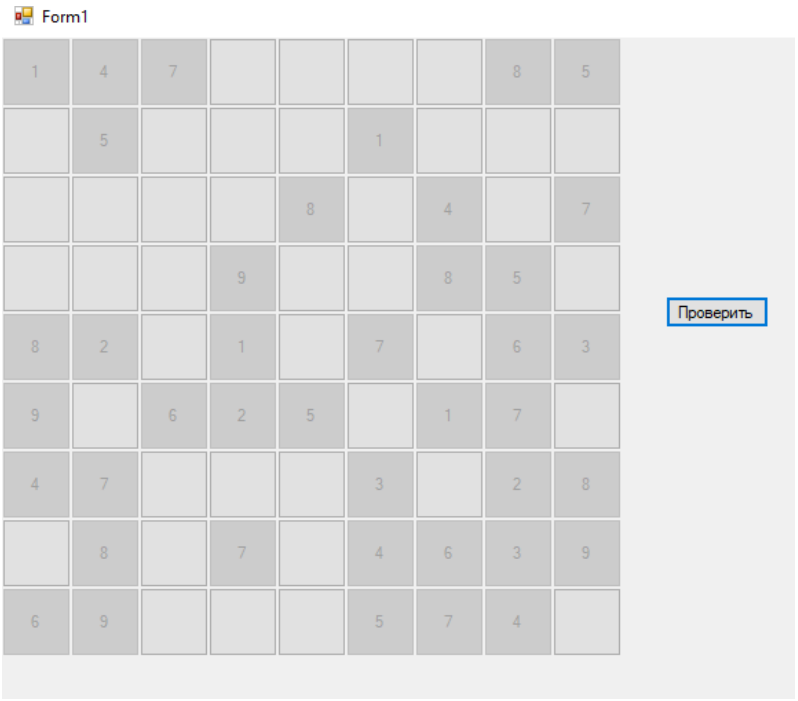
```
Ссылок: 1
private void button1_Click(object sender, EventArgs e)
{
    for (int i = 0; i < n * n; i++)
    {
        for (int j = 0; j < n * n; j++)
        {
            var btnText = buttons[i, j].Text;
            if (btnText != map[i, j].ToString())
            {
                MessageBox.Show("Неверно!");
                return;
            }
        }
    }
```

13.Прописываем код для отчистки карты перед запуском.

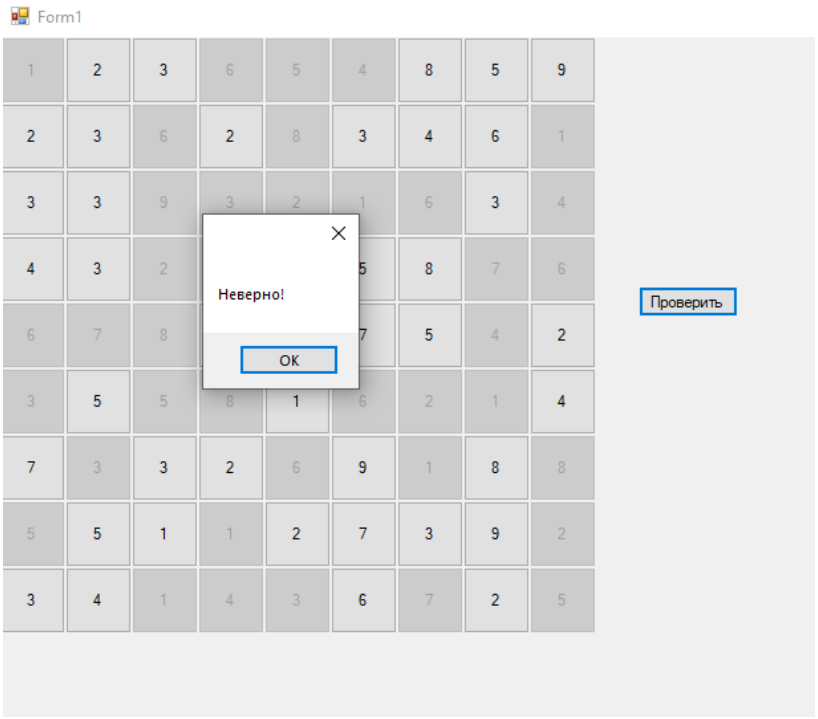
```
MessageBox.Show("Верно!");
for(int i = 0; i < n * n; i++)
{
    for (int j = 0; j < n * n; j++)
    {
        this.Controls.Remove(buttons[i, j]);
    }
}
GenerateMap();
```

Демонстрация результата.

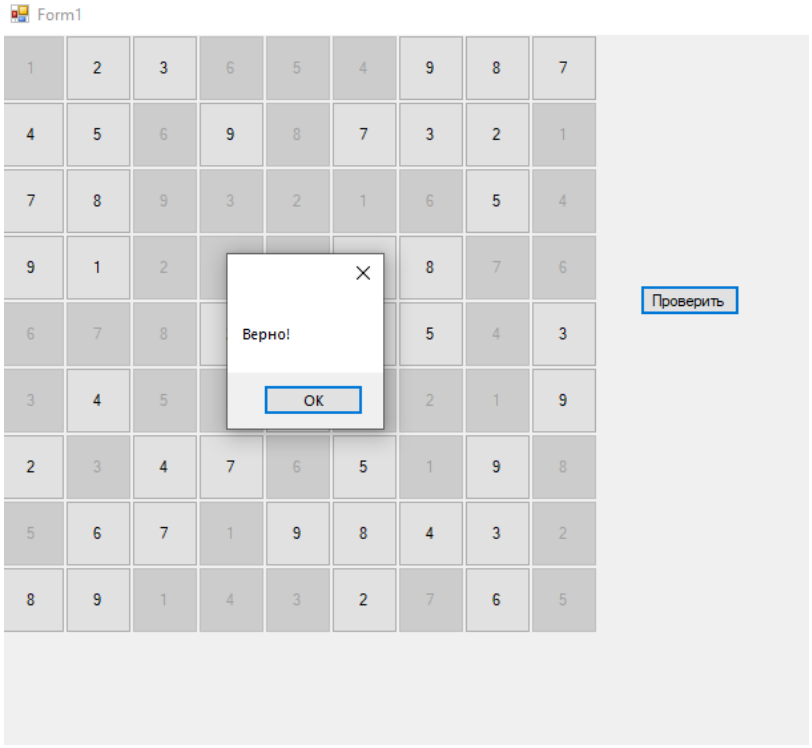
1. После написания программ, можно проверить ее работоспособность, для этого скомпилируем ее, нажав кнопку пуск в Visual Studio. У нас откроется наша форма.



2. Введем различные значения и нажмем кнопку «Проверить».



3. Как видим, мы ввели неправильные числа, поэтому программа выдала нам «Неверно!». Попробуем ввести правильное решение.



4. Программа показала, что мы верно решили головоломку. Все отлично работает!

Заключение.

Перед началом работы мы поставили себе задачу – создать популярную игру «Судоку» на компьютере, чтобы у людей была возможность отвлечься от работы во время перерыва или заниматься своим развитием в игровой форме в любой другой момент, используя для этого только монитор, клавиатуру и мышку. Для выполнения работы мы прописали необходимый код и проверили результат. Так как игра исправно работает и выполняет свои основные функции, было принято считать задачу выполненной. В будущем хочется исправить дизайн карты, чтобы он выглядел более презинтабельным и сделать игру пригодной для мобильных устройств.

Form1.cs:

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

namespace Sudoku
{
   public partial class Form1 : Form
   {
      const int n = 3;
      const int sizeButton = 50;
      public int[,] map = new int[n * n, n * n];
      public Button[,] buttons = new Button[n * n, n * n];
      public Form1()
      {
         InitializeComponent();
         GenerateMap();
      }

      public void GenerateMap()
      {
         for(int i = 0; i < n * n; i++)
         {
            for(int j = 0; j < n * n; j++)
            {
               map[i, j] = (i * n + i / n + j) % (n * n) + 1;
               buttons[i, j] = new Button();
            }
         }
         //MatrixTransposition();
         //SwapRowsInBlock();
         //SwapColumnsInBlock();
         //SwapBlocksInRow();
         //SwapBlocksInColumn();
         Random r = new Random();
         for(int i = 0; i < 40; i++)
         {
```

```
            ShuffleMap(r.Next(0, 5));
        }

        CreateMap();
        HideCells();
    }

    public void HideCells()
    {
        int N = 40;
        Random r = new Random();
        while (N > 0)
        {
            for (int i = 0; i < n * n; i++)
            {
                for (int j = 0; j < n * n; j++)
                {
                    if (!string.IsNullOrEmpty(buttons[i, j].Text)){
                        int a = r.Next(0, 3);
                        buttons[i, j].Text = a == 0 ? "" : buttons[i, j].Text;
                        buttons[i, j].Enabled = a == 0 ? true : false;

                        if (a == 0)
                            N--;
                        if (N <= 0)
                            break;
                    }
                }
                if (N <= 0)
                    break;
            }
        }
    }

    public void ShuffleMap(int i)
    {
        switch (i)
        {
            case 0:
                MatrixTransposition();
                break;
            case 1:
                SwapRowsInBlock();
                break;
            case 2:
                SwapColumnsInBlock();
                break;
            case 3:
                SwapBlocksInRow();
                break;
            case 4:
                SwapBlocksInColumn();
                break;
            default:
                MatrixTransposition();
                break;
        }
    }

    public void SwapBlocksInColumn()
    {
        Random r = new Random();
```

```
    var block1 = r.Next(0, n);
    var block2 = r.Next(0, n);
    while (block1 == block2)
      block2 = r.Next(0, n);
    block1 *= n;
    block2 *= n;
    for (int i = 0; i < n * n; i++)
    {
      var k = block2;
      for (int j = block1; j < block1 + n; j++)
      {
        var temp = map[i,j];
        map[i,j] = map[i,k];
        map[i,k] = temp;
        k++;
      }
    }
  }

  public void SwapBlocksInRow()
  {
    Random r = new Random();
    var block1 = r.Next(0, n);
    var block2 = r.Next(0, n);
    while (block1 == block2)
      block2 = r.Next(0, n);
    block1 *= n;
    block2 *= n;
    for(int i = 0; i < n * n; i++)
    {
      var k = block2;
      for(int j = block1; j < block1 + n; j++)
      {
        var temp = map[j, i];
        map[j, i] = map[k, i];
        map[k, i] = temp;
        k++;
      }
    }
  }

  public void SwapRowsInBlock()
  {
    Random r = new Random();
    var block = r.Next(0, n);
    var row1 = r.Next(0, n);
    var line1 = block * n + row1;
    var row2 = r.Next(0, n);
    while (row1 == row2)
      row2 = r.Next(0, n);
    var line2 = block * n + row2;
    for(int i = 0; i < n * n; i++)
    {
      var temp = map[line1, i];
      map[line1, i] = map[line2, i];
      map[line2, i] = temp;
    }
  }

  public void SwapColumnsInBlock()
  {
    Random r = new Random();
```

```
    var block = r.Next(0, n);
    var row1 = r.Next(0, n);
    var line1 = block * n + row1;
    var row2 = r.Next(0, n);
    while (row1 == row2)
        row2 = r.Next(0, n);
    var line2 = block * n + row2;
    for (int i = 0; i < n * n; i++)
    {
        var temp = map[i,line1];
        map[ i, line1] = map[i,line2];
        map[i,line2] = temp;
    }
}

public void MatrixTransposition()
{
    int[,] tMap = new int[n * n, n * n];
    for(int i = 0; i < n * n; i++)
    {
        for(int j = 0; j < n * n; j++)
        {
            tMap[i, j] = map[j, i];
        }
    }
    map = tMap;
}

public void CreateMap()
{
    for (int i = 0; i < n * n; i++)
    {
        for (int j = 0; j < n * n; j++)
        {
            Button button = new Button();
            buttons[i, j] = button;
            button.Size = new Size(sizeButton, sizeButton);
            button.Text = map[i, j].ToString();
            button.Click += OnCellPressed;
            button.Location = new Point(j * sizeButton, i * sizeButton);
            this.Controls.Add(button);
        }
    }
}

public void OnCellPressed(object sender, EventArgs e)
{
    Button pressedButton = sender as Button;
    string buttonText = pressedButton.Text;
    if (string.IsNullOrEmpty(buttonText))
    {
        pressedButton.Text = "1";
    }
    else
    {
        int num = int.Parse(buttonText);
        num++;
        if (num == 10)
            num = 1;
        pressedButton.Text = num.ToString();
    }
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            for(int i = 0; i < n * n; i++)
            {
                for(int j = 0; j < n * n; j++)
                {
                    var btnText = buttons[i, j].Text;
                    if(btnText != map[i, j].ToString())
                    {
                        MessageBox.Show("Неверно!");
                        return;
                    }
                }
            }
            MessageBox.Show("Верно!");
            for(int i = 0; i < n * n; i++)
            {
                for (int j = 0; j < n * n; j++)
                {
                    this.Controls.Remove(buttons[i, j]);
                }
            }
            GenerateMap();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```