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

namespace WindowsFormsApp2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void GenerateMatrix(string w, string k, Control box, int choice) //Stwórz pola typu Textbox dla macierzy
        {
            box.Controls.Clear();
            if (w == "" || k == "")
            {
                label11.Visible = true;
                label11.Text = "Błąd, nie wpisano wielkości macierzy";
                return;
            }
            box.Size = new Size(int.Parse(w)*60, int.Parse(k)*40);
            for (int i = 0; i < int.Parse(k); i++)
            {
                GroupBox row = new GroupBox();
                row.Margin = new Padding(0);
                row.Size = new Size(int.Parse(w)*60, 30);
                box.Controls.Add(row);
                for (int j = 0; j < int.Parse(w); j++)
                {
                    TextBox text = new TextBox();
                    text.Location = new Point(j*(50+10),0);
                    text.Size = new Size(50,30);
                    if (choice == 1)
                    {
                        Random rand = new Random();
                        for (int z = 0; z < 200000; z++)
                        {
                            text.Text = rand.Next(1, 10).ToString();
                        }
                    }
                    else if (choice == 2)
                    {
                        if (int.Parse(w) != int.Parse(k))
                        {
                            label11.Text = "Macierz nie jest kwadratowa";
                            label11.Visible = true;
                            return;
                        }
                        if (i==j){
                            text.Text = "1";
                        }else{
                            text.Text = "0";
                        }
                    }else{
                        text.Text = "";
                    }
                    row.Controls.Add(text);
                }
            }
        }
        private double[,] Read(Control box) //Czytaj wartości z pól
        {
            int i = 0;
            if (box.Controls.Count == 0)
            {
                double[,] tmpmat = new double[0,0];
                return tmpmat;
            }
            double[,] matrix = new double[box.Controls.Count, box.Controls
              [0].Controls.Count];
            foreach (Control control in box.Controls)
            {
                int j = 0;
                foreach (TextBox text in control.Controls)
                {
                    if (text.Text == "")
                    {
                        double[,] tmpmat = new double[0, 0];
                        return tmpmat;

                    }
                    matrix[i, j] = double.Parse(text.Text);
                    j++;
                }
                i++;
            }
            return matrix;
        }
        private void DisplayMatrix(double[,] matrix) //Wyświetl wartości w polach
        {
            GenerateMatrix(matrix.GetLength(1).ToString(), matrix.GetLength(0).ToString(), flowLayoutPanel3,0);
            int i = 0;

            foreach (Control control in flowLayoutPanel3.Controls)
            {
                int j = 0;
                foreach (TextBox text in control.Controls)
                {
                    text.Text = matrix[i, j].ToString();
                    j++;
                }
            i++;
            }
        }
        private void MatrixMultiplication(double[,] matrixA, double[,] matrixB) //Mnożenie macierzy
        {
            if (matrixA.GetLength(1) != matrixB.GetLength(0)) //Sprawdz czy dzialanie jest mozliwe
            {
                label11.Visible = true;
                label11.Text = "Blad, dzialanie niemozliwe, liczba kolumn macierzy A nie jest rowna liczbie wierszy macierzy B";
                return;
            }
            double[,] matrix = new double[matrixA.GetLength(0), matrixB.GetLength(1)];
            for (int k = 0; k < matrix.GetLength(1); k++)
            {
                for (int i = 0; i < matrixA.GetLength(0); i++)
                {
                    double sum = 0;
                    for (int j = 0; j < matrixA.GetLength(1); j++)
                    {
                        sum += matrixA[i, j] * matrixB[j,k];
                    }
                    matrix[i, k] = sum;
                }
            }
            DisplayMatrix(matrix);
        }
        private void DivideMatrix(double[,] matrixA, double[,] matrixB) //Dzielenie macierzy
        {
            double[,] InvMatrixB = InvertedMatrix(matrixB);
            MatrixMultiplication(matrixA, InvMatrixB);
        }
        private void AddMatrix(double[,] arr1, double[,] arr2) //Dodawanie macierzy
        {
            if (arr1.GetLength(0) != arr2.GetLength(0) || arr1.GetLength(1) != arr2.GetLength(1))
            {
                label11.Visible = true;
                label11.Text="Nie mozna dodac macierzy o roznych wymiarach";
                return;
            }
            for (int i = 0; i < arr1.GetLength(0); i++)
            {
                for (int j = 0; j < arr2.GetLength(1); j++)
                {
                    arr1[i, j] += arr2[i, j];
                }
            }
            DisplayMatrix(arr1);
        }
        private void SubtractMatrix(double[,] arr1, double[,] arr2) // Odejmowanie macierzy
        {
            if (arr1.GetLength(0) != arr2.GetLength(0) || arr1.GetLength(1) != arr2.GetLength(1))
            {
                label11.Visible = true;
                label11.Text="Nie mozna odejmowac macierzy o roznych wymiarach";
                return;
            }
            for (int i = 0; i < arr1.GetLength(0); i++)
            {
                for (int j = 0; j < arr2.GetLength(1); j++)
                {
                    arr1[i, j] -= arr2[i, j];
                }
            }
            DisplayMatrix(arr1);
        }
        private int MatrixDeterminant(double[,] matrix) //Wyznacznik macierzy
        {
            if (matrix.GetLength(0)!=matrix.GetLength(1)) return 0;
            int matrixsize = matrix.GetLength(1);
            for (int j = 0; j < matrixsize; j++)
            {
                double x = matrix[j, j]; //element listy na przekątnej
                if (x == 0) return 0; //jeśli element na przękątnej jest równy zero wyznacznik jest równy 0, można więc przerwać obliczenia
                for (int i = j + 1; i < matrixsize; i++)
                {
                    double y = matrix[i, j] / x; //liczba przez jaką trzeba pomnożyć wartość z wiersza j do wyzerowania wartości z wierszy
                    for (int k = 0; k < matrixsize; k++) //... element w wierszu "i" i kolumnie "j"
                    {
                        matrix[i, k] = matrix[i, k] - (matrix[j, k] * y); // odejmowanie wartości z wiersza i wartości z wiersza j pomnożonego przez wartość y
                    }
                }
            }
            double determinant = 1;
            for (int a = 0; a < matrixsize; a++)
            {
                determinant *= matrix[a, a];
            }
            //return (int)determinant;
            return (int)Math.Round(determinant);
        }
        private double[,] InvertedMatrix(double[,] matrix) //Macierz odwrotna
        {
            double[,] matrixClone = new double[matrix.GetLength(0), matrix.GetLength(1)];
            matrixClone = (double[,])matrix.Clone();
            int determinant = MatrixDeterminant(matrixClone);
            if (determinant != 0)
            {
                int matrixsize = matrix.GetLength(1);
                double[,] identitymatrix = new double[matrixsize, matrixsize]; //stworzenie macierzy jednostkowej
                for (int i = 0; i < matrixsize; i++)
                {
                    identitymatrix[i, i] = 1.0; //wypełnienie jej jedynkami na przekątnej
                }
                for (int j = 0; j < matrixsize; j++)
                {
                    double x = matrix[j, j];
                    for (int i = 0; i < matrixsize; i++) //zerowanie kolumn pod przekątną i odjęcie x od reszty wartości w wierszach
                    {
                        if (i == j) continue;
                        double y = matrix[i, j] / x;
                        for (int k = 0; k < matrixsize; k++)
                        {
                            identitymatrix[i, k] = identitymatrix[i, k] -
```

```
                         (identitymatrix[j, k] * y);
                                matrix[i, k] = matrix[i, k] - (matrix[j, k] * y);
                            }
                        }
                        for (int i = 0; i < matrixsize; i++) //uzyskanie 1 na 
                         przekątnej
                        {
                            identitymatrix[j, i] = (identitymatrix[j, i] / x);
                            matrix[j, i] = (matrix[j, i] / x);
                        }
                    }
                    for (int i = 0; i < matrixsize; i++)
                    {
                        for (int j = 0; j < matrixsize; j++)
                        {
                            identitymatrix[i, j] = Math.Round(identitymatrix[i, j], 
                         3); //zaokrąglanie wyników
                        }
                    }
                    return identitymatrix;
                }
                else
                {
                    label11.Visible = true;
                    label11.Text = "Macierz nie posiada macierzy odwrotnej";
                    return matrix;
                }
            }
            private void TransposeMatrix(double[,] matrix) //Macierz transponowana
            {
                double[,] TranMatrix = new double[matrix.GetLength(1), 
                  matrix.GetLength(0)];
                for (int i = 0; i < matrix.GetLength(0); i++)
                {
                    for (int j = 0; j < matrix.GetLength(1); j++)
                    {
                        TranMatrix[j, i] = matrix[i, j];
                    }
                }
                DisplayMatrix(TranMatrix);
            }
            private void Check(Control text)
            {
                if (text.Text != "")
                {
                    if (int.Parse(text.Text) > 8)
                    {
                        text.Text = "8";
                    }

                }
            }
            private void button1_Click(object sender, EventArgs e)
            {
                if (radioButton1.Checked == true)
                {
                    GenerateMatrix(textBox1.Text, textBox2.Text,
                      flowLayoutPanel1,1);
                }
                else if(radioButton2.Checked == true)
                {
                    GenerateMatrix(textBox1.Text, textBox2.Text,
                      flowLayoutPanel1,2);
                }
                else
                {
                    GenerateMatrix(textBox1.Text, textBox2.Text,
                      flowLayoutPanel1,0);
                }
            }
            private void textBox1_TextChanged(object sender, EventArgs e)
            {
                Check(textBox1);
            }
            private void button2_Click(object sender, EventArgs e)
            {
                if (radioButton4.Checked == true)
                {
                    GenerateMatrix(textBox4.Text, textBox3.Text, flowLayoutPanel2,
                      1);
                }
                else if (radioButton3.Checked == true)
                {
                    GenerateMatrix(textBox4.Text, textBox3.Text, flowLayoutPanel2,
                      2);
                }
                else
                {
                    GenerateMatrix(textBox4.Text, textBox3.Text, flowLayoutPanel2,
                      0);
                }
            }
            private void textBox2_TextChanged(object sender, EventArgs e)
            {
                Check(textBox2);
            }
            private void textBox4_TextChanged(object sender, EventArgs e)
            {
                Check(textBox4);
            }
            private void textBox3_TextChanged(object sender, EventArgs e)
            {
                Check(textBox3);
            }

            private void checkBox1_CheckedChanged(object sender, EventArgs e)
            {
                if (checkBox1.Checked)
                {
                    groupBox4.Enabled = true;
                    groupBox4.Visible = true;
                    groupBox1.Enabled = false;
                    groupBox5.Enabled = true;
                }
                else
                {
                    groupBox4.Enabled = false;
                    groupBox4.Visible= false;
                    groupBox1.Enabled = true;
                    groupBox5.Enabled = false;
                }
            }

            private void button3_Click(object sender, EventArgs e)
            {
                label11.Text = "";
                groupBox8.Visible = false;
                if (checkBox1.Checked == false)
                {
                    double[,] matrix = Read(flowLayoutPanel1);
                    if (radioButton10.Checked == true)
                    {
                        TransposeMatrix(matrix);

                    }else if (radioButton9.Checked == true)
                    {
                        double [,] invertedMatrix = InvertedMatrix(matrix);
                        DisplayMatrix(invertedMatrix);
                    }else if (radioButton11.Checked == true)
                    {
                        int determinant = MatrixDeterminant(matrix);
                        label14.Text = determinant.ToString();
                        groupBox8.Visible = true;
                    }
                    else
                    {
                        label11.Visible = true;
                        label11.Text = "Nie wybrano żadnego działania";
                    }
                }
                else
                {
                    double[,] matrix1 = Read(flowLayoutPanel1);
                    double[,] matrix2 = Read(flowLayoutPanel2);
                    if (radioButton5.Checked == true)
                    {
                        AddMatrix(matrix1, matrix2);
                    }else if (radioButton6.Checked == true)
                    {
                        SubtractMatrix(matrix1, matrix2);
                    }
                    else if(radioButton8.Checked == true)
                    {
                       MatrixMultiplication(matrix1, matrix2);
                    }
                    else if (radioButton7.Checked == true)
                    {
                        DivideMatrix(matrix2, matrix1);
                    }
                    else
                    {
                        label11.Visible = true;
                        label11.Text = "Nie wybrano żadnego działania";
                    }
                }
            }
        }

}
```