MatrixWhiz - Temă proiect "Date structure"

Secțiunea "Educațional"

Nume Elev: Bobea Alin-Gabriel

Profesor Coordonator: Petre Florin

Școala: Colegiul Național "Nicolae Iorga", Vălenii de Munte

SECȚIUNEA 1. Descrierea proiectului

MatrixWhiz este o aplicație inovatoare și captivantă proiectată pentru a facilita învățarea și explorarea matricelor într-un mod interactiv și distractiv. Cu o interfață intuitivă și o gamă diversă de caracteristici, MatrixWhiz aduce lumea complexă a matricelor la îndemâna utilizatorilor de toate nivelele de experiență în programare și matematică.

SECȚIUNEA 2. Principii și idei folosite la crearea acestui program

Reflectând asupra programului, Bobea Alin-Gabriel a respectat toate principiile enunțate în regulamentul acestei competiții. Afirmăm că toate materialele sunt create de noi, cu mici excepții, anume:
-Ecuațiile pentru fractal.

-Această versiune de minesweeper este concepută și facută de noi, însă noi nu deținem jocul original.

SECȚIUNEA 3. Conținutul programului

SECȚIUNEA 3.1 Paginile principale/meniuri

1. Pagina de login

Acest cod gestionează autentificarea unui utilizator prin conectarea la o bază de date locală. Utilizatorul poate vizualiza sau ascunde parola bifând o casetă de selecție. La apăsarea butonului de autentificare, codul verifică emailul și parola în baza de date. Dacă datele sunt corecte, utilizatorul este autentificat și formularul principal se deschide. Dacă autentificarea eșuează, este afișat un mesaj de eroare. De asemenea, codul gestionează deschiderea și

închiderea formularului de înregistrare printr-un link.

```
public partial class login : Form
{
    SqlConnection con = new SqlConnection(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=" +
Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName +
@"\Database1.mdf;Integrated Security=True");
    SqlCommand com;
    SqlDataReader dr;
    acasa form;
    Form9 register;
    bool register_deschis = false;
    public static string utilizator_logat;
    public login()
    {
        InitializeComponent();
    }

    private void checkBox1_CheckedChanged(object sender, EventArgs e)
    {
        if (checkBox1.Checked)
        {
            textBox2.PasswordChar = '\0';
        }
        else
        {
            textBox2.PasswordChar = '*';
        }
    }

    private void button1_Click(object sender, EventArgs e)
```

```
    {
        con.Open();
        com = new SqlCommand("select * from utilizatori where email = @e and
parola = @p", con);
        com.Parameters.AddWithValue("@e", textBox1.Text);
        com.Parameters.AddWithValue("@p", textBox2.Text);
        dr = com.ExecuteReader();
        if (dr.HasRows)
        {
            while (dr.Read())
            {
                utilizator_logat = dr["prenume"].ToString();
                MessageBox.Show(utilizator_logat);
            }
            this.Hide();
            form = new acasa();
            form.Show();
        }
        else
        {
            label3.Visible = true;
            textBox1.Text = "";
            textBox2.Text = "";
        }
        con.Close();
    }

    private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
    {
```

```
        if (!register_deschis)
        {
            register = new Form9();
            register.Show();
            register_deschis = true;
        }
        else
        {
            register_deschis = false;
            register.Close();
        }
    }
}
```

2. Pagina de register

Acest cod gestionează înregistrarea unui utilizator în aplicație. Conectează aplicația la o bază de date locală pentru a verifica existența unui email introdus de utilizator. Dacă emailul există deja, afișează un mesaj de eroare. Dacă nu, înregistrează utilizatorul cu datele introduse și afișează un mesaj de confirmare. Codul permite, de asemenea, vizualizarea sau ascunderea caracterelor parolei în funcție de starea casetelor de selecție.

```
    public partial class Form9 : Form
    {
        SqlConnection con = new SqlConnection(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=" +
Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName +
@"\Database1.mdf;Integrated Security=True");
        SqlCommand com;
        SqlDataReader dr;
        acasa form;
        public static string utilizator_logat;
        public Form9()
        {
```

```
            InitializeComponent();
        }
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked)
            {
                textBox2.PasswordChar = '\0';
            }
            else
            {
                textBox2.PasswordChar = '*';
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            com = new SqlCommand("select * from utilizatori where email=@t",
con);
            com.Parameters.AddWithValue("@t", textBox1.Text);
            con.Open();
            dr = com.ExecuteReader();
            if (dr.HasRows)
            {
                dr.Close();
                label9.Visible = true;
            }
            else
            {
                dr.Close();
                com = new SqlCommand("insert into utilizatori (email, parola,
nume, prenume) values (@email, @parola, @nume, @prenume)", con);
```

```
                com.Parameters.AddWithValue("@email", textBox1.Text);
                com.Parameters.AddWithValue("@parola", textBox2.Text);
                com.Parameters.AddWithValue("@nume", textBox4.Text);
                com.Parameters.AddWithValue("@prenume", textBox5.Text);
                dr = com.ExecuteReader();
                MessageBox.Show("Cont inregistrat! Vă mulțumim!");
            }
            con.Close();
        }
        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox2.Checked)
            {
                textBox3.PasswordChar = '\0';
            }
            else
            {
                textBox3.PasswordChar = '*';
            }
        }
    }
```

3. Pagina de acasă

Este meniul principal. De aici utilizatorul poate selecta mai multe meniuri pentru a le parcurge. Codul este simplu, toate butoanele folosesc un cod identic, singura diferență fiind numele variabilelor.

```
public partial class acasa : Form
{
    int form_deschis = 0;
    int form_deschis1 = 0;
```

```
int form_deschis2 = 0;
int form_deschis3 = 0;
int form_deschis4 = 0;
int form_deschis5 = 0;
int count = 0;
bool pauza = false;
informatii form2;
meniu_aplicatii form5;
meniu_chestionare form6;
meniu_diverse form7;
int r = 106;
int g = 90;
int b = 205;
private Size marime_formular;
private Rectangle gradient;
private int opacitate = 255;
Font font = new Font("Arial", 16);
private int form_x;
private int form_y;
string[] v = new string[1];
int k, i = 0, l;
int lungsubsir = 0;
private StreamReader str;
string s, s1;
System.Windows.Forms.Timer timer;
string citat;
public acasa()
{
    InitializeComponent();
```

```
            label6.Text = login.utilizator_logat;
            if (login.utilizator_logat == null)
            {
                label6.Text = "UTILIZATOR";
            }
            str = new StreamReader("citate.txt");
            timer1.Start();
            timer2.Interval = 40;
            timer3.Interval = 2000;
            CenterToScreen();
            marime_formular = this.Size;
            this.Paint += new PaintEventHandler(set_fundal);
        }
        private void set_fundal(Object sender, PaintEventArgs e)
        {
            Graphics graphics = e.Graphics;
            gradient = new Rectangle(0, 0, Width, Height);
            Brush a = new SolidBrush(Color.FromArgb(opacitate, 0, 0, 0));
            Brush b = new LinearGradientBrush(gradient, Color.FromArgb(57, 20, 127),
Color.FromArgb(60, 120, 210), 65f);
            graphics.FillRectangle(b, gradient);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (form_deschis == 0)
            {
                form2 = new informatii();
                form2.Show();
                form_deschis = 1;
            }
```

```
    else
    {
        form2.Close();
        form_deschis = 0;
    }
}
private void asteapta(int milliseconds)
{
    Task.Delay(milliseconds);
}
private void button2_Click(object sender, EventArgs e)
{
    if (form_deschis3 == 0)
    {
        form5 = new meniu_aplicatii();
        form5.Show();
        form_deschis3 = 1;
    }
    else
    {
        form5.Close();
        form_deschis3 = 0;
    }
}
private void button3_Click(object sender, EventArgs e)
{
    if (form_deschis4 == 0)
    {
        form6 = new meniu_chestionare();
```

```
            form6.Show();
            form_deschis4 = 1;
        }
        else
        {
            form6.Close();
            form_deschis4 = 0;
        }
    }
    private void button4_Click(object sender, EventArgs e)
    {
        if (form_deschis5 == 0)
        {
            form7 = new meniu_diverse();
            form7.Show();
            form_deschis5 = 1;
        }
        else
        {
            form6.Close();
            form_deschis5 = 0;
        }
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        if (str == null || str.EndOfStream)
        {
            str = new StreamReader("citate.txt");
        }
```

```
    citat = str.ReadLine();
    count = 0;
    timer2.Start();
    timer1.Stop();
}
private async Task scrie_citat()
{
    if (citat != null)
    {
        if (count < citat.Length)
        {
            label3.Text += citat[count];
            count++;
        }
        else
        {
            await Task.Delay(4000);
            label3.Text = "";
            timer2.Stop();
            timer1.Start();
        }
    }
    else
    {
        label3.Text = "NULL";
    }
}
private async void timer2_Tick(object sender, EventArgs e)
{
```

```
            await scrie_citat();
        }
    }
```

4. Meniul pentru informații

Meniul cu informații generale despre proiect, folosește un gradient brush pentru a da un efect frumos fundalului.

```
public partial class informatii : Form
{
    informatii form2;
    private Rectangle gradient;
    private int opacitate = 255;
    Font font = new Font("Arial", 16);
    public informatii()
    {
        InitializeComponent();
        this.Paint += new PaintEventHandler(set_fundal);
    }
    private void set_fundal(Object sender, PaintEventArgs e)
    {
        Graphics graphics = e.Graphics;
        gradient = new Rectangle(0, 0, Width, Height);
        Brush a = new SolidBrush(Color.FromArgb(opacitate, 0, 0, 0));
        Brush b = new LinearGradientBrush(gradient, Color.FromArgb(57, 20, 127),
Color.FromArgb(60, 120, 210), 65f);
        SizeF textSize = graphics.MeasureString("PROIECT (am uitat numele)",
font);
        float x = (Width - textSize.Width) / 2;
        float y = (Height - textSize.Height) / 4 - textSize.Height / 2;
        PointF puncte = new PointF(x, y);
        graphics.FillRectangle(b, gradient);
```

```
    }
}
```

5. Meniul pentru diverse

Conține toate lecțiile/parcurgeriile. Are aproape același cod în butoane ca meniul de acasă, excepția fiind numele variabilelor.

```
public partial class meniu_diverse : Form
{
    item1 item1;
    item2 item2;
    item3 item3;
    item4 item4;
    item5 item5;
    item6 item6;
    item7 item7;
    item8 item8;
    item9 item9;
    item10 item10;
    item11 item11;
    item12 item12;
    item13 item13;
    item14 item14;
    item15 item15;
    item16 item16;
    item17 item17;
    item18 item18;
    public bool formal = false;
    public bool non_formal = false;
    public bool informal = false;
    public meniu_diverse()
```

```
{
    InitializeComponent();
}

private void button1_Click(object sender, EventArgs e)
{
    if (!formal)
    {
        panel1.Show();
        panel2.Hide();
        formal = true;
        informal = false;
    }
    else
    {
        panel1.Hide();
        formal = false;
        informal = false;
    }
}

private void button2_Click(object sender, EventArgs e)
{
    item1 = new item1();
    item1.Show();
}

private void panel1_Paint(object sender, PaintEventArgs e)
{
```

```
}

private void button5_Click(object sender, EventArgs e)
{
    item2 = new item2();
    item2.Show();
}

private void button8_Click(object sender, EventArgs e)
{
    item5 = new item5();
    item5.Show();
}

private void button6_Click(object sender, EventArgs e)
{
    item3 = new item3();
    item3.Show();
}

private void button9_Click(object sender, EventArgs e)
{
    item4 = new item4();
    item4.Show();
}

private void button7_Click(object sender, EventArgs e)
{
```

```
    item13 = new item13();
    item13.Show();
}

private void button12_Click(object sender, EventArgs e)
{
    item14 = new item14();
    item14.Show();
}

private void button20_Click(object sender, EventArgs e)
{
    item6 = new item6();
    item6.Show();
}

private void button18_Click(object sender, EventArgs e)
{
    item7 = new item7();
    item7.Show();
}

private void button19_Click(object sender, EventArgs e)
{
    item12 = new item12();
}

private void button17_Click(object sender, EventArgs e)
{
```

```
    item16 = new item16();
    item16.Show();
}

private void button16_Click(object sender, EventArgs e)
{
    item17 = new item17();
    item17.Show();
}

private void button15_Click(object sender, EventArgs e)
{
    item11 = new item11();
    item11.Show();
}

private void button14_Click(object sender, EventArgs e)
{
    item10 = new item10();
    item10.Show();
}

private void button13_Click(object sender, EventArgs e)
{
    item9 = new item9();
    item9.Show();
}

private void button3_Click(object sender, EventArgs e)
```

```
{
    item8 = new item8();
    item8.Show();
}

private void button4_Click(object sender, EventArgs e)
{
    if (!informal)
    {
        panel1.Hide();
        panel2.Show();
        informal = true;
        formal = false;
    }
    else
    {
        panel2.Hide();
        informal = false;
        formal = false;
    }
}

private void button10_Click(object sender, EventArgs e)
{
    item15 = new item15();
    item15.Show();
}

private void button11_Click(object sender, EventArgs e)
```

```
    {
        item18 = new item18();
        item18.Show();
    }
}
```

6. Meniul pentru aplicații

Conține jocurile, identic din punct de vedere tehnic ca meniul de diverse.

```
public partial class meniu_aplicatii : Form
{
    puzzle form6;
    meniu_minesweeper form7;
    quiz1 quiz1;
    x_0 form8;
    int form_deschis1 = 0;
    int form_deschis2 = 0;
    int form_deschis3 = 0;
    int form_deschis4 = 0;
    public meniu_aplicatii()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (form_deschis1 == 0)
        {
            form6 = new puzzle();
            form6.Show();
```

```
        form_deschis1 = 1;
    }
    else
    {
        form6.Close();
        form_deschis1 = 0;
    }
}

private void button4_Click(object sender, EventArgs e)
{
    if (form_deschis2 == 0)
    {
        form7 = new meniu_minesweeper();
        form7.Show();
        form_deschis2 = 1;
    }
    else
    {
        form7.Close();
        form_deschis2 = 0;
    }
}

private void button2_Click(object sender, EventArgs e)
{
    if (form_deschis3 == 0)
    {
        form8 = new x_0();
```

```
            form8.Show();
            form_deschis3 = 1;
        }
        else
        {
            form8.Close();
            form_deschis3 = 0;
        }
    }

    private void button3_Click(object sender, EventArgs e)
    {
        if (form_deschis3 == 0)
        {
            quiz1 = new quiz1();
            quiz1.Show();
            form_deschis4 = 1;
        }
        else
        {
            quiz1.Close();
            form_deschis4 = 0;
        }
    }
}
```

7. Meniul pentru chestionare

Conține toate chestionarele. Identic din punct de vedere tehnic cu meniul de diverse.

```
public partial class meniu_chestionare : Form
```

```
{
    int form_deschis = 0;
    int form_deschis1 = 0;
    int form_deschis2 = 0;
    quiz1 q1;
    quiz2 q2;
    quiz3 q3;
    public meniu_chestionare()
    {
        InitializeComponent();
    }

    private void button3_Click(object sender, EventArgs e)
    {
        if (form_deschis == 0)
        {
            q1 = new quiz1();
            q1.Show();
            form_deschis = 1;
        }
        else
        {
            q1.Close();
            form_deschis = 0;
        }
    }

    private void button6_Click(object sender, EventArgs e)
    {
```

```
        if (form_deschis1 == 0)
        {
            q2 = new quiz2();
            q2.Show();
            form_deschis1 = 1;
        }
        else
        {
            q2.Close();
            form_deschis1 = 0;
        }
    }

    private void button5_Click(object sender, EventArgs e)
    {
        if (form_deschis2 == 0)
        {
            q3 = new quiz3();
            q3.Show();
            form_deschis2 = 1;
        }
        else
        {
            q3.Close();
            form_deschis2 = 0;
        }
    }
}
```

SECȚIUNEA 3.2 Aplicații și diverse

1. Parcurgerea unei matrici deasupra diagonalei principale

Acest cod implementează o parcurgere a unei matrici deasupra diagonalei principale într-o aplicație de ferestre. La început, sunt definite variabilele și inițializate elementele grafice necesare. Apoi, în funcția parcurgere1_Shown, se construiește și se desenează matricea, iar apoi se inițiază o parcurgere a elementelor acesteia printr-un timer. La fiecare tică al timerului, un punct de deasupra diagonalei principale este colorat.

```
public partial class item1 : Form
{
    int nr, nr1;
    Graphics g;
    Class1[] v = new Class1[1000];
    Rectangle[,] a = new Rectangle[100, 100];
    SolidBrush s = new SolidBrush(Color.Red);
    item2 form;
    public item1()
    {
        InitializeComponent();
    }

    private void button2_Click(object sender, EventArgs e)
    {

    }

    private async void timer1_Tick(object sender, EventArgs e)
    {
        if (nr1 == nr)
        {
            timer1.Stop();
```

```
    }
    else
    {
        nr1++;
        g.FillEllipse(s, a[v[nr1].x, v[nr1].y]);
    }
}

private void parcurgere1_Shown(object sender, EventArgs e)
{
    g = this.CreateGraphics();
    Pen p = new Pen(Color.Blue, 1);
    SolidBrush diag = new SolidBrush(Color.Blue);
    SolidBrush s = new SolidBrush(Color.Green);
    for (int i = 1; i <= 6; i++)
        for (int j = 1; j <= 6; j++)
        {
            a[i, j] = new Rectangle();
            a[i, j].Height = 50;
            a[i, j].Width = 50;
            a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);

            if (i == j)
            {
                g.DrawEllipse(p, a[i, j]);
                g.FillEllipse(diag, a[i, j]);
            }
            else
            {
```

```
                g.FillEllipse(s, a[i, j]);
            }
        }
    nr = 0;
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j <= i; j++)
        {
            nr++;
            v[nr] = new Class1();
            v[nr].x = 6 - i + j;
            v[nr].y = j;
        }

        nr1 = 0;
        timer1.Start();
    }
}

private async Task afiseaza_urmatorul()
{
    item2 nextForm = new item2();
    nextForm.Show();
    await Task.Delay(100);
    this.Close();
}
private void animatie()
{
    nr = 0;
```

```
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (i + j == 6 - 1)
            {
                nr++;
                v[nr] = new Class1();
                v[nr].x = i;
                v[nr].y = j;
            }
        }
    }
    nr1 = 0;
    timer1.Start();
}

private void item1_Load(object sender, EventArgs e)
{

}

private void incepe()
{
    g = this.CreateGraphics();
    Pen p = new Pen(Color.Blue, 1);
    SolidBrush diag = new SolidBrush(Color.Blue);
    SolidBrush s = new SolidBrush(Color.Green);
    for (int i = 1; i <= 6; i++)
```

```
    for (int j = 1; j <= 6; j++)
    {
        a[i, j] = new Rectangle();
        a[i, j].Height = 50;
        a[i, j].Width = 50;
        a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);

        if (i == j)
        {
            g.DrawEllipse(p, a[i, j]);
            g.FillEllipse(diag, a[i, j]);
        }
        else
        {
            g.FillEllipse(s, a[i, j]);
        }
    }
nr = 0;
for (int i = 0; i < 6; i++)
{
    for (int j = 0; j <= i; j++)
    {
        nr++;
        v[nr] = new Class1();
        v[nr].x = 6 - i + j;
        v[nr].y = j;
    }

    nr1 = 0;
```

```
            timer1.Start();
        }
    }
}
```

2. Parcurgerea unei matrici deasupra diagonalei secundare

Codul este aproape identic cu cel de mai sus, singura diferență fiind că cel de mai jos parcurge elementele unei matrici deasupra diagonalei secundare.

```
public partial class item2 : Form
{
    int nr, nr1;
    Graphics g;
    Class1[] v = new Class1[1000];
    Rectangle[,] a = new Rectangle[100, 100];
    SolidBrush s = new SolidBrush(Color.Red);
    SolidBrush blue = new SolidBrush(Color.Blue);
    SolidBrush green = new SolidBrush(Color.Green);
    public item2()
    {
        InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {

    }
    private async void timer1_Tick(object sender, EventArgs e)
    {
        if (nr1 == nr)
        {
            timer1.Stop();
            if (!timer1.Enabled)
```

```
        {
            await Task.Delay(1000);
            this.Close();
        }
    }
    else
    {
        nr1++;
        g.FillEllipse(s, a[v[nr1].x, v[nr1].y]);
    }
}
private void parcurgere2_Load(object sender, EventArgs e)
{

}
private void incepe()
{
    g = this.CreateGraphics();
    Pen p = new Pen(Color.Red, 1);
    Pen n = new Pen(Color.Black, 1);
    for (int i = 1; i <= 6; i++)
    {
        for (int j = 1; j <= 6; j++)
        {
            a[i, j] = new Rectangle();
            a[i, j].Height = 50;
            a[i, j].Width = 50;
            a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);
```

```
            g.DrawEllipse(n, a[i, j]);
            if ((i + j) == 7)
            {
                g.FillEllipse(blue, a[i, j]);
            }
            if ((i + j) > 7)
            {
                g.FillEllipse(green, a[i, j]);
            }
        }
    }
}

private void parcurgere2_Shown(object sender, EventArgs e)
{
    g = this.CreateGraphics();
    Pen p = new Pen(Color.Red, 1);
    Pen n = new Pen(Color.Black, 1);
    for (int i = 1; i <= 6; i++)
    {
        for (int j = 1; j <= 6; j++)
        {
            a[i, j] = new Rectangle();
            a[i, j].Height = 50;
            a[i, j].Width = 50;
            a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);

            g.DrawEllipse(n, a[i, j]);
            if ((i + j) == 7)
```

```
            {
                g.FillEllipse(blue, a[i, j]);
            }
            if ((i + j) > 7)
            {
                g.FillEllipse(green, a[i, j]);
            }
        }
    }
    nr = 0;
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j <= i; j++)
        {
            nr++;
            v[nr] = new Class1();
            v[nr].x = i - j + 1;
            v[nr].y = j;
        }
        nr1 = 0;
        timer1.Start();
    }
}

private void animatie()
{
    nr = 0;
    for (int i = 0; i < 6; i++)
    {
```

```
            for (int j = 0; j <= i; j++)
            {
                nr++;
                v[nr] = new Class1();
                v[nr].x = i - j + 1;
                v[nr].y = j;
            }
            nr1 = 0;
            timer1.Start();
        }
    }
}
```

3. Parcurgerea unei matrici sub diagonala principală

Codul este aproape identic cu cel de mai sus, singura diferență fiind că cel de mai jos parcurge elementele unei matrici sub diagonala principală.

```
public partial class item3 : Form
{
    int nr, nr1;
    Graphics g;
    Class1[] v = new Class1[1000];
    Rectangle[,] a = new Rectangle[100, 100];
    SolidBrush s = new SolidBrush(Color.Red);
    SolidBrush blue = new SolidBrush(Color.Blue);
    SolidBrush green = new SolidBrush(Color.Green);
    public item3()
    {
        InitializeComponent();
    }
    private void animatie()
    {
```

```
        g = this.CreateGraphics();
        Pen p = new Pen(Color.Red, 1);
        Pen n = new Pen(Color.Black, 1);
        for (int i = 1; i <= 6; i++)
            for (int j = 1; j <= 6; j++)
            {
                a[i, j] = new Rectangle();
                a[i, j].Height = 50;
                a[i, j].Width = 50;
                a[i, j].Location = new Point(5 + i * 56, 15 + j * 56); //5 + i *
56, 15 + j * 56

                g.DrawEllipse(n, a[i, j]);
                if (i == j)
                {
                    g.FillEllipse(blue, a[i, j]);
                }
                if ((i - j) > 0)
                {
                    g.FillEllipse(green, a[i, j]);
                }
            }
        nr = 0;
        for (int i = 0; i < 6; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                nr++;
                v[nr] = new Class1();
                v[nr].x = j;
```

```
                v[nr].y = 6 - i + j;
            }
            nr1 = 0;
            timer1.Start();
        }
    }
    private void parcurgere3_Shown(object sender, EventArgs e)
    {
        animatie();
    }

    private async void timer1_Tick(object sender, EventArgs e)
    {
        if (nr1 == nr)
        {
            timer1.Stop();
            if (!timer1.Enabled)
            {
                await Task.Delay(1000);
                this.Close();
            }
        }
        else
        {
            nr1++;
            g.FillEllipse(s, a[v[nr1].x, v[nr1].y]);
        }
    }
}
```

4. Parcurgerea unei matrici sub diagonala secundară

Codul este aproape identic cu cel de mai sus, singura diferență fiind că cel de mai jos parcurge elementele unei matrici sub diagonala secundară.

```
public partial class item4 : Form
{
    int nr, nr1;
    Graphics g;
    Class1[] v = new Class1[1000];
    Rectangle[,] a = new Rectangle[100, 100];
    SolidBrush s = new SolidBrush(Color.Red);
    SolidBrush blue = new SolidBrush(Color.Blue);
    SolidBrush green = new SolidBrush(Color.Green);
    public item4()
    {
        InitializeComponent();
    }

    private async void timer1_Tick(object sender, EventArgs e)
    {
        if (nr1 == nr) timer1.Stop();
        else
        {
            nr1++;

            g.FillEllipse(s, a[v[nr1].x, v[nr1].y]);

        }
    }

    private void parcurgere4_Shown(object sender, EventArgs e)
```

```
    {
        animatie();
    }

    private void item4_Load(object sender, EventArgs e)
    {

    }

    private void animatie()
    {
        g = this.CreateGraphics();
        Pen p = new Pen(Color.Red, 1);
        Pen n = new Pen(Color.Black, 1);
        for (int i = 1; i <= 6; i++)
            for (int j = 1; j <= 6; j++)
            {
                a[i, j] = new Rectangle();
                a[i, j].Height = 50;
                a[i, j].Width = 50;
                a[i, j].Location = new Point(5 + i * 56, 15 + j * 56); //5 + i *
56, 15 + j * 56

                g.DrawEllipse(n, a[i, j]);
                if ((i + j) == 7)
                {
                    g.FillEllipse(blue, a[i, j]);
                }
                if ((i + j) < 7)
                {
```

```
                    g.FillEllipse(green, a[i, j]);
                }
            }
        nr = 0;
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                nr++;
                v[nr] = new Class1();
                v[nr].x = 6 - j;
                v[nr].y = 6 - i + j;
            }
            nr1 = 0;
            timer1.Start();
        }
    }
}
```

5. Matrice pliabilă din colț

Acest cod definește o fereastră într-o aplicație care afișează o matrice care "se pliază" din colț. La început, sunt create și inițializate obiectele PictureBox pentru fiecare element din matrice. Aceste obiecte sunt adăugate pe fereastra curentă și inițial au dimensiuni zero și sunt colorate în roșu. Apoi, este pornit un timer care crește dimensiunile elementelor matricei astfel încât acestea să pară că "se pliază" din colț. După o anumită perioadă de timp, timerul se oprește și este pornit un alt timer care micșorează dimensiunile elementelor matricei, simulând "dezplierea" lor. Acest proces se repetă până când matricea este complet "dezpliată".

```
public partial class item5 : Form
{
```

```
PictureBox[,] a = new PictureBox[100, 100];
int k;
public item5()
{
    InitializeComponent();
}

private void item5_Shown(object sender, EventArgs e)
{
    for (int i = 0; i < 10; i++)
        for (int j = 0; j < 10; j++)
        {
            a[i, j] = new PictureBox();
            a[i, j].Height = 0;
            a[i, j].Width = 0;
            a[i, j].Location = new Point(5 + i * 56, 15 + j * 32);
            a[i, j].BackColor = Color.Red;
            this.Controls.Add(a[i, j]);
        }
    k = 0;
    timer1.Start();
}

private async void timer1_Tick(object sender, EventArgs e)
{
    if (k == 30)
    {
        timer1.Stop();
        await Task.Delay(1000);
```

```
            k = 0;
            timer2.Start();
        }
        else
        {
            k++;
            for (int i = 0; i < 10; i++)
                for (int j = 0; j < 10; j++)
                {
                    a[i, j].Width += 1;
                    a[i, j].Height += 1;
                }
        }
    }

    private void timer2_Tick(object sender, EventArgs e)
    {
        if (k == 30)
        {
            this.Close();
            timer2.Stop();
        }
        else
        {
            k++;
            for (int i = 0; i < 10; i++)
                for (int j = 0; j < 10; j++)
                {
                    a[i, j].Width -= 1;
```

```
                a[i, j].Height -= 1;
            }
        }
    }

    private void label4_Click(object sender, EventArgs e)
    {

    }
}
```

6. Animație de matrice în stil de ploaie

La inițializare, se stabilesc parametrii de bază, cum ar fi dimensiunile celulelor și numărul de rânduri și coloane ale matricei care va reprezenta ploaia. Se utilizează o matrice de booleane pentru a indica dacă picăturile de ploaie sunt prezente sau nu în fiecare celulă.

În funcția incepe_ploaia(), se stabilește inițializarea picăturilor de ploaie pe primul rând, iar în funcția picura(), picăturile sunt deplasate în jos și sunt simulate prin schimbarea stării lor în matricea de picături. Astfel, picăturile de ploaie sunt create în mod aleatoriu și călătoresc în jos până când ajung la partea de jos a ferestrei, moment în care dispar și pot fi create altele noi. Această simulare a ploii este actualizată și desenată în mod repetat la fiecare intervale regulate de timp folosind un timer.

```
public partial class item6 : Form
{
    private const int CellSize = 10;
    private const int Rows = 20;
    private const int Columns = 40;
    private bool[,] rainDrops = new bool[Rows, Columns];
    private Random random = new Random();
```

```
private Timer timer = new Timer();
private Brush brush = Brushes.Blue;
public item6()
{
    InitializeComponent();
    incepe_ploaia();
    timer1.Interval = 50;
    timer1.Start();
}

private void timer1_Tick(object sender, EventArgs e)
{
    picura();
    Invalidate();
}
private void incepe_ploaia()
{
    for (int j = 0; j < Columns; j++)
    {
        rainDrops[0, j] = random.Next(0, 10) == 0;
    }
}

private void picura()
{
    for (int i = Rows - 1; i >= 0; i--)
    {
        for (int j = 0; j < Columns; j++)
        {
```

```
                if (rainDrops[i, j])
                {
                    rainDrops[i, j] = false;
                    if (i < Rows - 1)
                    {
                        rainDrops[i + 1, j] = true;
                    }
                }
            }
        }
        for (int j = 0; j < Columns; j++)
        {
            rainDrops[0, j] = random.Next(0, 10) == 0;
        }
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                if (rainDrops[i, j])
                {
                    e.Graphics.FillRectangle(brush, j * CellSize, i * CellSize,
CellSize, CellSize);
                }
            }
        }
```

```
    }


}
```

7. Animație de matrice în stil de scântei

Acest cod creează o animație de scântei care se deplasează în sus pe ecran. Sunt create 100 de scântei, fiecare cu o poziție și o dimensiune aleatorii. Acestea sunt reprezentate sub formă de elipse colorate cu o nuanță de galben către roșu, utilizând un gradient linear. La fiecare cadru de animație, scânteile se mișcă în sus cu o viteză aleatoare. Când o scânteie ajunge la partea de sus a ecranului, își resetează poziția și dimensiunea pentru a începe din nou.

```
public partial class item7 : Form

{

    private readonly Random aleator = new Random();

    private const int NumarScantei = 100;

    private readonly Scanteie[] scantei = new Scanteie[NumarScantei];

    private readonly Brush[] pensule = { Brushes.Yellow, Brushes.Orange,
Brushes.Red };

    private const int DimensiuneScanteieMin = 3;

    private const int DimensiuneScanteieMax = 12;

    public item7()

    {

        InitializeComponent();

        InitializareScantei();

        StartArderi();

    }


    private void timer1_Tick(object sender, EventArgs e)

    {


    }
```

```
    private void InitializareScantei()
    {
        for (int i = 0; i < NumarScantei; i++)
        {
            scantei[i] = new Scanteie
            {
                X = aleator.Next(ClientSize.Width / 2 + 80),
                Y = aleator.Next(ClientSize.Height),
                Dimensiune = aleator.Next(DimensiuneScanteieMin,
DimensiuneScanteieMax + 1)
            };
        }
    }

    private async void StartArderi()
    {
        while (true)
        {
            await Task.Delay(25);

            for (int i = 0; i < NumarScantei; i++)
            {
                scantei[i].Y -= aleator.Next(1, 4);
                if (scantei[i].Y < 0)
                {
                    scantei[i].Y = ClientSize.Height;
                    scantei[i].X = aleator.Next(ClientSize.Width / 2 + 80);
                    scantei[i].Dimensiune = aleator.Next(DimensiuneScanteieMin,
DimensiuneScanteieMax + 1);
                }
```

```
            }

            Invalidate();

        }

    }




    protected override void OnPaint(PaintEventArgs e)

    {

        foreach (var scanteie in scantei)

        {

            using (var gradienta = new LinearGradientBrush(new Point(0,
scanteie.Y), new Point(0, scanteie.Y + scanteie.Dimensiune), Color.Yellow,
Color.Red))

            {

                e.Graphics.FillEllipse(gradienta, scanteie.X, scanteie.Y,
scanteie.Dimensiune, scanteie.Dimensiune);

            }

        }

    }



    private class Scanteie

    {

        public int X { get; set; }

        public int Y { get; set; }

        public int Dimensiune { get; set; }

    }



    private void item7_Paint(object sender, PaintEventArgs e)

    {
```

```
    }
}
```

8. Simularea unor "licurici" folosind o matrice

La inițializare, sunt stabilite dimensiunile matricei și numărul de particule de licurici care vor fi simulate. Particulele sunt plasate inițial în mod aleatoriu în matrice.

Funcția timer1_Tick este responsabilă pentru actualizarea pozițiilor particulelor la fiecare interval de timp. Aceasta este realizată prin apelul funcției MoveParticles, care determină noile poziții ale particulelor conform unor reguli de mișcare aleatorii. Particulele se deplasează în toate direcțiile înconjurătoare în mod aleatoriu.

În funcția OnPaint, matricea este desenată pe fereastra cu licurici, iar particulele sunt reprezentate prin dreptunghiuri albe pe fundalul negru, indicând prezența lor în matricea bidimensională.

```
public partial class item9 : Form
{
    private const int Width = 50;
    private const int Height = 50;
    private const int NumParticles = 50;
    private bool[,] matrix = new bool[Width, Height];
    private Random rand = new Random();
    public item9()
    {
        InitializeComponent();
        InitializeParticles();
        timer1.Interval = 50;
```

```
    timer1.Start();
}
private void InitializeParticles()
{
    for (int i = 0; i < NumParticles; i++)
    {
        int x = rand.Next(Width);
        int y = rand.Next(Height);
        matrix[x, y] = true;
    }
}
private void timer1_Tick(object sender, EventArgs e)
{
    MoveParticles();
    Invalidate();
}
private void MoveParticles()
{
    bool[,] newMatrix = new bool[Width, Height];
    for (int x = 0; x < Width; x++)
    {
        for (int y = 0; y < Height; y++)
        {
            if (matrix[x, y])
            {
                matrix[x, y] = false;
                int newX = (x + rand.Next(-1, 2) + Width) % Width;
                int newY = (y + rand.Next(-1, 2) + Height) % Height;
                newMatrix[newX, newY] = true;
```

```
                }
            }
        }
        matrix = newMatrix;
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics g = e.Graphics;
        g.Clear(Color.Black);
        float cellWidth = (float)ClientSize.Width / 2 / Width;
        float cellHeight = (float)ClientSize.Height / Height;
        for (int x = 0; x < Width; x++)
        {
            for (int y = 0; y < Height; y++)
            {
                if (matrix[x, y])
                {
                    RectangleF rect = new RectangleF(x * cellWidth, y *
cellHeight, cellWidth, cellHeight);
                    g.FillRectangle(Brushes.White, rect);
                }
            }
        }
    }

    private void label4_Click(object sender, EventArgs e)
    {

    }
```

```
}
```

9. Simulare expansiune

La inițializare, zona este inițializată cu un anumit procentaj de celule aprinse, iar la fiecare ticăit al timerului, focul se extinde în zonele vecine ale celulelor deja aprinse. Noua poziție a focului este determinată aleatoriu între pozițiile disponibile din jurul unei celule aprinse.

În funcția OnPaint, zona este desenată pe fereastră, iar celulele aprinse sunt reprezentate printr-un dreptunghi portocaliu.

```
public partial class item10 : Form

{

    private const int Width = 50;

    private const int Height = 25;

    private const int marime_celula = 10;

    private const double probabilitate = 0.15;

    private bool[,] grid = new bool[Width, Height];

    private Random random = new Random();

    public item10()

    {

        InitializeComponent();

        fa_panou();

        timer1.Start();

    }


    private void item10_Load(object sender, EventArgs e)

    {


    }


    private void timer1_Tick(object sender, EventArgs e)
```

```
{
    extinde_foc();
    Invalidate();

}
private void fa_panou()
{
    for (int x = 0; x < Width; x++)
    {
        for (int y = 0; y < Height; y++)
        {
            grid[x, y] = random.NextDouble() < probabilitate;
        }
    }
}
private void extinde_foc()
{
    bool[,] newGrid = new bool[Width, Height];
    for (int x = 0; x < Width; x++)
    {
        for (int y = 0; y < Height; y++)
        {
            if (grid[x, y])
            {
                newGrid[x, y] = true;
                extinde(newGrid, x, y);
            }
            else
            {
```

```
                    newGrid[x, y] = grid[x, y];
                }
            }
        }
        grid = newGrid;
    }
    private void extinde(bool[,] panou_nou, int x, int y)
    {
        for (int dx = -1; dx <= 1; dx++)
        {
            for (int dy = -1; dy <= 1; dy++)
            {
                if ((dx != 0 || dy != 0) && e_panou(x + dx, y + dy) && !grid[x +
dx, y + dy])
                {
                    if (random.NextDouble() < 0.5)
                    {
                        panou_nou[x + dx, y + dy] = true;
                    }
                }
            }
        }
    }
    private bool e_panou(int x, int y)
    {
        return x >= 0 && x < Width && y >= 0 && y < Height;
    }

    protected override void OnPaint(PaintEventArgs e)
    {
```

```
        base.OnPaint(e);

        Graphics g = e.Graphics;

        g.Clear(Color.Black);


        for (int x = 0; x < Width; x++)

        {

            for (int y = 0; y < Height; y++)

            {

                if (grid[x, y])

                {

                    Rectangle rect = new Rectangle(x * marime_celula, y *
marime_celula, marime_celula, marime_celula);

                    g.FillRectangle(Brushes.OrangeRed, rect);

                }

            }

        }

    }

}
```

10. Steluțe de gheață

La inițializare, sunt create și desenate steluțe de gheață, fiecare formată din linii care se întâlnesc în centrul unui cerc imaginar.

În funcția item11_Shown, sunt create și desenate steluțele de gheață în locațiile specificate. Desenarea este realizată prin intermediul funcției desen, care primește coordonatele de start ale steluței, dimensiunea acesteia și realizează desenarea efectivă.

```
public partial class item11 : Form

{

    Rectangle[,] a = new Rectangle[100, 100];

    Class1[] v = new Class1[1000];
```

```
int nr = 0, nr1;
Graphics g;
SolidBrush s = new SolidBrush(Color.Red);
public item11()
{
    InitializeComponent();
}

private void item11_Shown(object sender, EventArgs e)
{
    g = this.CreateGraphics();
    Pen p = new Pen(Color.Red, 1);
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 6; j++)
        {
            a[i, j] = new Rectangle();
            a[i, j].Height = 50;
            a[i, j].Width = 50;
            a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);
            desen(5 + i * 56, 15 + j * 56, 50);
        }

    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 6; j++)
        {
            if (i > j)
            {

                v[nr] = new Class1();
```

```
                v[nr].x = i;
                v[nr].y = j; nr++;
            }
        }
    nr1 = 0;
    timer1.Start();
}
void desen(int x, int y, int d)
{
    Graphics g = this.CreateGraphics();
    Pen blackPen = new Pen(Color.Black, 1);
    Point start = new Point(0, 0);
    Point end = new Point(0, d / 2);
    g.TranslateTransform(x + d / 2, y + d / 2);
    for (int i = 1; i <= 12; i++)
    {
        g.DrawLine(blackPen, start, end);
        g.RotateTransform(30.0F);
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (nr1 == nr)
    {
        timer1.Stop();

    }
    else
```

```
        {
            nr1++;
        }
    }

    private void item11_Load(object sender, EventArgs e)
    {

    }
}
```

11. Fractal Mandelbrot

Acest fractal este generat prin iterarea unui set de ecuații și colorarea pixelilor în funcție de numărul de iterații necesare pentru ca valoarea să depășească un anumit prag.

În cadrul funcției item12_Shown, se definește dimensiunea imaginii fractalei și se creează o imagine goală. Apoi, pentru fiecare pixel din imagine, se calculează coordonatele corespunzătoare în planul complex și se aplică formula specifică fractalului Mandelbrot pentru a determina culoarea pixelului.

Culoarea pixelilor este determinată în funcție de numărul de iterații necesare pentru a depăși un anumit prag. Dacă numărul de iterații depășește pragul maxim, pixelul este colorat în negru; altfel, culoarea este determinată în funcție de numărul de iterații și este atribuită pixelului în funcție de acesta.

După ce toți pixelii sunt colorați, imaginea fractală este afișată într-un control PictureBox.

```
public partial class item12 : Form
{

    public item12()
    {
```

```
    InitializeComponent();
}

private void item12_Load(object sender, EventArgs e)
{

}

private void item12_Shown(object sender, EventArgs e)
{
    int width = 800;
    int height = 600;

    Bitmap fractalImage = new Bitmap(width, height);

    double xMin = -2.5;
    double xMax = 1.0;
    double yMin = -1.0;
    double yMax = 1.0;

    for (int xPixel = 0; xPixel < width; xPixel++)
    {
        for (int yPixel = 0; yPixel < height; yPixel++)
        {
            double x0 = xMin + (xMax - xMin) * xPixel / (double)width;
            double y0 = yMin + (yMax - yMin) * yPixel / (double)height;

            double x = 0, y = 0;
            int iterations = 0;
```

```
                int maxIterations = 1000;


                while (x * x + y * y <= 5 && iterations < maxIterations)
                {
                    double xTemp = x * x - y * y + x0;
                    y = 2 * x * y + y0;
                    x = xTemp;
                    iterations++;
                }


                Color color;
                if (iterations == maxIterations)
                {
                    color = Color.Black;
                }
                else
                {
                    int colorValue = iterations % 256;
                    color = Color.FromArgb(colorValue, colorValue, colorValue);
                }


                fractalImage.SetPixel(xPixel, yPixel, color);
            }
            pictureBox1.Image = fractalImage;
        }
    }
}
```

12. Matrice pliantă orizontal

La inițializare, sunt create controale PictureBox cu lățimea zero și înălțimea specificată, apoi acestea sunt adăugate pe

fereastră. Un timer este pornit pentru a gestiona extinderea matricei de controale PictureBox în mod progresiv în timp.

În funcția item13_Shown, sunt create controale PictureBox și sunt plasate pe fereastră într-o matrice de dimensiune 10x10, fiecare control având o înălțime fixă și o lățime inițială de zero. Aceste controale sunt colorate în roșu și sunt adăugate pe fereastră.

Apoi, în funcția timer1_Tick, lățimea controalelor PictureBox este crescută treptat la fiecare ticăit al timerului, până când atinge valoarea de 40. La acest punct, timerul se oprește.

```
public partial class item13 : Form
{
    PictureBox[,] a = new PictureBox[100, 100];
    int k;
    public item13()
    {
        InitializeComponent();
        timer1.Interval = 50;
    }

    private void item13_Shown(object sender, EventArgs e)
    {
        for (int i = 0; i < 10; i++)
            for (int j = 0; j < 10; j++)
            {
                a[i, j] = new PictureBox();
                a[i, j].Height = 40;
                a[i, j].Width = 0;
                a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);
                a[i, j].BackColor = Color.Red;
                this.Controls.Add(a[i, j]);
```

```
            }
        k = 0;
        timer1.Start();
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        if (k == 40) timer1.Stop();
        else
        {
            k++;
            for (int i = 0; i < 10; i++)
                for (int j = 0; j < 10; j++)
                    a[i, j].Width += 1;
        }

    }
}
```

13. Matrice pliantă vertical

La inițializare, sunt create controale PictureBox cu înălțimea zero și lățimea specificată, apoi acestea sunt adăugate pe fereastră. Un timer este pornit pentru a gestiona extinderea matricei de controale PictureBox în mod progresiv în timp.

În funcția item14_Shown, sunt create controale PictureBox și sunt plasate pe fereastră într-o matrice de dimensiune 10x10, fiecare control având o lățime fixă și o înălțime inițială de zero. Aceste controale sunt colorate în roșu și sunt adăugate pe fereastră.

Apoi, în funcția timer1_Tick, înălțimea controalelor PictureBox este crescută treptat la fiecare ticăit al timerului, până când atinge valoarea de 30. La acest punct, timerul se oprește.

```
public partial class item14 : Form
{
    PictureBox[,] a = new PictureBox[100, 100];
    int k;
    public item14()
    {
        InitializeComponent();
    }

    private void item14_Shown(object sender, EventArgs e)
    {
        for (int i = 0; i < 10; i++)
            for (int j = 0; j < 10; j++)
            {
                a[i, j] = new PictureBox();
                a[i, j].Height = 0;
                a[i, j].Width = 30;
                a[i, j].Location = new Point(5 + i * 56, 15 + j * 56);
                a[i, j].BackColor = Color.Red;
                this.Controls.Add(a[i, j]);
            }
        k = 0;
        timer1.Start();
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        if (k == 30) timer1.Stop();
        else
        {
```

```
            k++;
            for (int i = 0; i < 10; i++)
                for (int j = 0; j < 10; j++)
                    a[i, j].Height += 1;
        }
    }
}
```

14. Simulare program C#

În cadrul acestei simulări, se utilizează un fișier text pentru a citi și afișa linii de cod, iar apoi se simulează execuția acestor linii de cod prin schimbarea culorii textului din albastru în roșu, pe măsură ce programul "rulează".

În funcția item15_Shown, se încarcă și se afișează conținutul unui fișier text care conține liniile de cod ale programului de simulat. Fiecare linie este afișată printr-un control Label pe fereastră.

Funcția suma_matrice simulează execuția programului C#. Acesta construiește o matrice, calculează suma elementelor pare și înregistrează secvența de execuție într-un vector de instrucțiuni.

Funcția restart este apelată la fiecare trecere prin ciclul de execuție și reinițializează culorile textului la albastru pentru a indica că linia de cod respectivă nu este în execuție.

Timerul timer1 este folosit pentru a simula execuția programului, prin trecerea prin secvența de instrucțiuni și schimbarea culorii textului corespunzător fiecărei linii de cod.

```
public partial class item15 : Form
{
    int i = 0, nr = 0, m, it = -1;
    Label[] l = new Label[100];
    int[] v = new int[100];
    public item15()
```

```
    {
        InitializeComponent();
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        if (it == nr - 1) timer1.Stop();
        else
        {
            it++;
            restart();
            l[v[it % nr]].ForeColor = System.Drawing.Color.Red;
        }
    }

    private void item15_Shown(object sender, EventArgs e)
    {
        using (StreamReader fin = new StreamReader("parcurgeri_matrici
\\matrice.txt"))
        {
            while (!fin.EndOfStream)
            {
                string s = fin.ReadLine();
                l[i] = new Label();
                l[i].Location = new Point(100, 30 + i * 30);
                l[i].Size = new Size(350, 30);
                l[i].Font = new Font("Times New Roman", 16);
                l[i].Text = s;
                this.Controls.Add(l[i]);
                i++;
```

```
        }
        fin.Close();
        m = i;

    }
    suma_matrice();
    timer1.Start();
    i = 0;
}
void suma_matrice()
{
    int x;
    int[,] a = new int[100, 100]; x = 2; v[nr] = x; nr++;
    int n = 3, i, j, s = 0; x = 3; v[nr] = x; nr++;
    for (i = 1; i <= n; i++)
    {
        x = 4; v[nr] = x; nr++;
        for (j = 1; j <= n; j++)
        {
            x = 5; v[nr] = x; nr++;
            { a[i, j] = i + j; x = 6; v[nr] = x; nr++; }
        }
    }
    for (i = 1; i <= n; i++)
    {
        x = 7; v[nr] = x; nr++;
        for (j = 1; j <= n; j++)
        {
            x = 8; v[nr] = x; nr++;
```

```
                if (a[i, j] % 2 == 0)
                {
                    x = 9; v[nr] = x; nr++;
                    s = s + a[i, j]; x = 10; v[nr] = x; nr++;
                }
            }
        }

    }
    void restart()
    {
        for (int i = 0; i < m; i++)
        {
            l[i].ForeColor = System.Drawing.Color.Blue;
        }
    }
}
```

15. Simularea răspândirii apei in vas

Acest cod utilizează o rețea de butoane pentru a reprezenta starea apei în diferite celule ale vasului și un timer pentru a actualiza și desena starea apei în mod continuu.

În funcția InitializeWaterGrid, se creează o rețea de butoane, fiecare reprezentând o celulă a vasului, și se inițializează nivelurile de apă pentru fiecare celulă cu valori aleatoare.

Funcția item16_Shown pornește timerul pentru a începe simularea.

Timerul timer1 declanșează actualizarea și desenarea stării apei la fiecare interval de timp specificat în Interval.

Funcția UpdateWater calculează nivelurile de apă pentru fiecare celulă a vasului, luând în considerare și apa din celulele adiacente.

Funcția DrawWater colorează butoanele reprezentând celulele vasului în funcție de nivelurile lor de apă, astfel încât culoarea să varieze de la albastru (nivel scăzut de apă) la alb (nivel înalt de apă).

```
        private const int marime_panou = 20;
        private Button[,] panou_apa = new Button[marime_panou, marime_panou];
        private float[,] nivel_apa = new float[marime_panou, marime_panou];
        private Random rand = new Random();
        private Timer timer = new Timer();
        public item16()
        {
            InitializeComponent();
            timer1.Interval = 250;
            InitializeWaterGrid();
        }
        private void InitializeWaterGrid()
        {
            for (int i = 0; i < marime_panou; i++)
            {
                for (int j = 0; j < marime_panou; j++)
                {
                    Button button = new Button
                    {
                        Size = new Size(30, 30),
                        Location = new Point(i * 30, j * 30),
                        BackColor = Color.Blue,
                        FlatStyle = FlatStyle.Flat
                    };
                    panou_apa[i, j] = button;
                    Controls.Add(button);
                }
```

```
    }
    for (int i = 0; i < marime_panou; i++)
    {
        for (int j = 0; j < marime_panou; j++)
        {
            nivel_apa[i, j] = (float)rand.NextDouble();
        }
    }
}

private void item16_Shown(object sender, EventArgs e)
{
    timer1.Start();
}

private void timer1_Tick(object sender, EventArgs e)
{
    UpdateWater();
    DrawWater();
}
private void UpdateWater()
{
    float[,] nivel_apa_urm = new float[marime_panou, marime_panou];
    for (int i = 0; i < marime_panou; i++)
    {
        for (int j = 0; j < marime_panou; j++)
        {
            float apa_totala = nivel_apa[i, j];
            int celule_adiacente = 1;
```

```
            if (i > 0)
            {
                apa_totala += nivel_apa[i - 1, j];
                celule_adiacente++;
            }
            if (i < marime_panou - 1)
            {
                apa_totala += nivel_apa[i + 1, j];
                celule_adiacente++;
            }
            if (j > 0)
            {
                apa_totala += nivel_apa[i, j - 1];
                celule_adiacente++;
            }
            if (j < marime_panou - 1)
            {
                apa_totala += nivel_apa[i, j + 1];
                celule_adiacente++;
            }
            nivel_apa_urm[i, j] = apa_totala / celule_adiacente;
        }
    }
    nivel_apa = nivel_apa_urm;
}

private void DrawWater()
{
    for (int i = 0; i < marime_panou; i++)
```

```
            {
                for (int j = 0; j < marime_panou; j++)
                {
                    int colorValue = (int)(nivel_apa[i, j] * 255);
                    panou_apa[i, j].BackColor = Color.FromArgb(colorValue,
colorValue, 255);
                }
            }
        }

        private void label4_Click(object sender, EventArgs e)
        {

        }
    }
```

16. Simularea propagării sunetului în mediu

Acest cod simulează propagarea sunetului în aer folosind o rețea de butoane pentru a reprezenta intensitatea sunetului în diferite locații.

În cadrul constructorului, se inițializează intervalul timerului și rețeaua de butoane. De asemenea, se stabilește poziția centrală a sunetului în rețeaua de butoane.

Funcția intensitate calculează intensitatea sunetului în fiecare locație a rețelei de butoane bazându-se pe distanța față de centrul sunetului și utilizând funcția sinus pentru a simula variația intensității în funcție de distanță.

Funcția deseneaza_Sunet colorează butoanele corespunzătoare locațiilor în funcție de intensitatea sunetului calculată anterior. Cu cât intensitatea sunetului este mai mare, cu atât culoarea butonului asociat va fi mai închisă.

Timerul timer1 declanșează actualizarea și desenarea stării sunetului în rețeaua de butoane la fiecare interval de timp

specificat.

```
public partial class item17 : Form
{
    private const int panou = 20;
    private Button[,] panou_m = new Button[panou, panou];
    private double[,] intensityGrid = new double[panou, panou];
    private int centerX, centerY;
    public item17()
    {
        InitializeComponent();
        timer1.Interval = 1000;
        panou_m_p();
        centerX = panou / 6;
        centerY = panou / 2;
    }
    private void panou_m_p()
    {
        for (int i = 0; i < panou; i++)
        {
            for (int j = 0; j < panou; j++)
            {
                Button button = new Button
                {
                    Size = new Size(30, 30),
                    Location = new Point(i * 30, j * 30),
                    BackColor = Color.White,
                    FlatStyle = FlatStyle.Flat
                };
                panou_m[i, j] = button;
```

```
                Controls.Add(button);
            }
        }
    }
    private void intensitate()
    {
        for (int i = 0; i < panou; i++)
        {
            for (int j = 0; j < panou; j++)
            {
                double distance = Math.Sqrt(Math.Pow(i - centerX, 2) +
Math.Pow(j - centerY, 2));
                intensityGrid[i, j] = Math.Sin(distance * 0.1) * 0.5 + 0.5;
            }
        }
    }


    private void deseneaza_Sunet()
    {
        for (int i = 0; i < panou; i++)
        {
            for (int j = 0; j < panou; j++)
            {
                int culoare = (int)(intensityGrid[i,j] * 255);
                panou_m[i, j].BackColor = Color.FromArgb(culoare, culoare,
culoare);
            }
        }
        panou_m[centerX, centerY].BackColor = Color.Black;
    }
```

```
    private void timer1_Tick(object sender, EventArgs e)
    {
        intensitate();
        deseneaza_Sunet();
    }

    private void item17_Shown(object sender, EventArgs e)
    {
        timer1.Start();
    }
}
```

17. Algoritmul Lee

La început, aplicația inițializează o rețea de butoane care reprezintă matricea în care se va face căutarea drumului. Utilizatorul poate specifica dimensiunea matricei folosind un control NumericUpDown. După ce utilizatorul a selectat dimensiunea matricei și a apăsat butonul "Start", se afișează matricea de butoane și utilizatorul poate alege două puncte: punctul de start și punctul de sfârșit.

Apăsarea butonului "Start" începe căutarea drumului folosind algoritmul Lee. Acest algoritm folosește o coadă pentru a explora toate celulele adiacente începând cu celula de start și înregistrând distanța față de celula de start în matricea c. Când algoritmul găsește celula de sfârșit, se oprește, iar drumul cel mai scurt este marcat cu culoarea verde pe butoanele respective.

Butonul "Obstacole" permite utilizatorului să marcheze celulele matricei ca obstacole, iar butonul "Reset" resetează selecția punctelor de start și de sfârșit pentru a permite utilizatorului să selecteze alte puncte.

```
public partial class item18 : Form
{
    int n, i = 0, j = 0, ip = 0, jp = 0, iso = 0, jso = 0, p = 1, u = 1, d, KK =
```

```
0, kp = 1, nr = 0, KK2 = 0, ifn, jfn, iv, jv;
      Random r = new Random();
      public Button[,] a = new Button[16, 16];
      public int[,] a1 = new int[15, 15];
      public int[,] c = new int[115, 115];
      public int[] di = new int[] { -1, 0, 1, 0 };
      public int[] dj = new int[] { 0, 1, 0, -1 };
      public int[] px = new int[115];
      public int[] py = new int[115];
      int xb, yb;
      public item18()
      {
          InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
          int i, j;
          n = int.Parse(numericUpDown1.Value.ToString());
          for (i = 0; i < n; i++)
              for (j = 0; j < n; j++)
                  a1[i, j] = 0;
          int k = 10 - n;
          for (i = 0; i < n; i++)
              for (j = 0; j < n; j++)
              {
                  a[i, j] = new Button();
                  a[i, j].Height = 50;
                  a[i, j].Width = 50;
```

```
                a[i, j].Text = "";
                a[i, j].Name = i + " " + j;
                a[i, j].BackColor = Color.Gray;
                a[i, j].Tag = Convert.ToString(n * i + j);
                a[i, j].Location = new Point(((k * 50) / 2) + i * 50, 100 + j *
50);
                a[i, j].Click += new EventHandler(v_click);
                this.Controls.Add(a[i, j]);
            }
        button1.Visible = false;
        numericUpDown1.Visible = false;
        KK++;
    }

    bool OK(int i, int j)
    {


        if (i < 0 || j < 0 || i > n - 1 || j > n - 1)
            return false;
        if (a1[i, j] == -1) return false;
        if (c[i, j] != 0) return false;
        if (i == iso && j == jso) return false;

        return true;
    }
    bool ultimul()
    {
        if (iv == ifn && jv == jfn)
            return true;
```

```
    return false;
}
private void timer1_Tick(object sender, EventArgs e)
{
    i = 0; j = 0;
    if (!(p <= u && a1[i, j] == 0) || ultimul() == true)
    {
        a[ifn, jfn].BackColor = Color.Yellow;
        a[ip, jp].BackColor = Color.Yellow;
        timer1.Stop();
        button4.Visible = false;
    }
    else
    {

        if (i < n)

            if (j < n)


                if (a1[i, j] == 0)
                    if (d < 4)
                    {


                        iv = px[p] + di[d];
                        jv = py[p] + dj[d];
                        d++;
```

```
                            if (OK(iv, jv) == true)
                            {

                                a[iv, jv].BackColor = Color.Green;
                                u++;
                                px[u] = iv; py[u] = jv;
                                c[px[u], py[u]] = c[px[p], py[p]] + 1;
                                a[iv, jv].Text = c[iv, jv].ToString();

                            }
                            if (OK(iv, jv) == true && a[iv, jv].BackColor ==
Color.Black)
                                nr++;


                        }
                        else { d = 0; p++; }
                    else j++;


                else { j = 0; i++; }



        }
    }

    private void button4_Click(object sender, EventArgs e)
    {
        if (KK2 == 0)
```

```
    {
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                a[i, j].Click += null;
        px[1] = ip;
        py[1] = jp;
        c[px[1], py[1]] = 1;
        timer1.Start();
        if (a[0, 0].BackColor == Color.Gray)
        {
            a[0, 0].BackColor = Color.Black;

        }
        KK2++;
        button4.Text = "Programul";
        button4.Enabled = false;
    }
    else
    {
        //Form36 f = new Form36();
        //f.Show();
    }
}

private void button3_Click(object sender, EventArgs e)
{
    if (kp == 1)
    {
        MessageBox.Show("Nu ai selectat locatie");
```

```
    }
    else
    {
        KK++;
        xb = 0;
        yb = 0;
        kp = 1;
        MessageBox.Show("Pune obstacole");
        button3.Visible = false;
        button4.Visible = true;
    }
}

private void button2_Click(object sender, EventArgs e)
{
    if (kp == 1)
    {
        MessageBox.Show("Nu ai selectat locatie");
    }
    else
    {
        KK++;
        xb = 0;
        yb = 0;
        kp = 1;
        MessageBox.Show("Alege urmatorul punct");
        button2.Visible = false;
        button3.Visible = true;
    }
```

```
}
public void v_click(object sender, EventArgs e)
{
    string s;
    string[] ss = new string[2];
    Button btn = (Button)sender;
    if (KK == 3)
    {
        int i1, j1;
        int nrb = Convert.ToInt32(((Button)sender).Tag);



        i1 = nrb / n;
        j1 = nrb % n;
        if (a[i1, j1].BackColor != Color.White)
        {
            a[i1, j1].BackColor = Color.Black;
            a1[i1, j1] = -1;
        }
    }
    if (KK == 1)
    {
        if (a[xb, yb].BackColor == Color.White && kp == 0)
        {
            a[xb, yb].BackColor = Color.Gray;


        }
```

```
    s = btn.Name;
    ss = s.Split(' ');
    xb = int.Parse(ss[0]);
    yb = int.Parse(ss[1]);
    ip = xb;
    jp = yb;
    kp = 0;
    btn.BackColor = Color.White;

}
if (KK == 2)
{

    if (a[xb, yb].BackColor == Color.White && kp == 0)
    {
        a[xb, yb].BackColor = Color.Gray;

    }
    s = btn.Name;

    ss = s.Split(' ');
    xb = int.Parse(ss[0]);
    yb = int.Parse(ss[1]);
    ifn = xb;
    jfn = yb;
    kp = 0;
    btn.BackColor = Color.White;

}
```

```
    }
}
```

18. Quiz 1

Inițializarea quiz-ului: În constructorul clasei quiz1, se încarcă întrebările și răspunsurile din fișierul text utilizând metoda IncarcaIntrebarile. Apoi, se afișează prima întrebare utilizând metoda AfiseazaIntrebare.

Încărcarea întrebărilor: Metoda IncarcaIntrebarile citește linie cu linie din fișierul text și împarte fiecare linie în întrebare și răspunsuri utilizând caracterul '/' pentru întrebare și caracterul ';' pentru răspunsuri. Aceste informații sunt stocate în tablourile intrebari și raspunsuri.

Afișarea întrebării curente: Metoda AfiseazaIntrebare primește un index și afișează întrebarea corespunzătoare din tablourile intrebari și raspunsuri. Răspunsurile sunt afișate pe butoane radio pentru a permite utilizatorului să selecteze un răspuns.

Verificarea răspunsului: În metoda button1_Click, se verifică dacă utilizatorul a selectat un răspuns înainte de a trece la următoarea întrebare. Dacă da, se apelează metoda VerificaRaspuns pentru a verifica dacă răspunsul este corect și se actualizează scorul. Apoi, se trece la următoarea întrebare sau se afișează un mesaj cu scorul final dacă nu mai sunt întrebări disponibile.

Verificarea corectitudinii răspunsului: Metoda VerificaRaspuns compară răspunsul selectat de utilizator cu răspunsul corect stocat în tabloul raspunsuri și incrementează scorul dacă răspunsul este corect.

```
public partial class quiz1 : Form
{
    private string[] intrebari;
    private string[][] raspunsuri;
    private int intrebareCurenta;
    private int scor;
    public quiz1()
    {
        InitializeComponent();
```

```
    IncarcaIntrebarile("quiz1.txt");
    AfiseazaIntrebare(intrebareCurenta);
}
private void IncarcaIntrebarile(string numeFisier)
{
    try
    {
        string[] linii = File.ReadAllLines(numeFisier);
        intrebari = new string[linii.Length];
        raspunsuri = new string[linii.Length][];

        for (int i = 0; i < linii.Length; i++)
        {
            string[] componente = linii[i].Split('/');
            intrebari[i] = componente[0];
            raspunsuri[i] = componente[1].Split(';');
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Eroare la citirea fisierului: " + ex.Message);
    }
}

private void AfiseazaIntrebare(int index)
{
    if (index >= 0 && index < intrebari.Length)
    {
        label4.Text = intrebari[index];
```

```
            radioButton1.Text = raspunsuri[index][0];
            radioButton2.Text = raspunsuri[index][1];
            radioButton3.Text = raspunsuri[index][2];
        }
        else
        {
            MessageBox.Show("Indexul intrebarii este invalid.");
        }
    }
    private void quiz1_Load(object sender, EventArgs e)
    {

    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (radioButton1.Checked || radioButton2.Checked || radioButton3.Checked)
        {
            VerificaRaspuns();
            intrebareCurenta++;
            if (intrebareCurenta < intrebari.Length)
            {
                AfiseazaIntrebare(intrebareCurenta);
                radioButton1.Checked = radioButton2.Checked =
radioButton3.Checked = false;
            }
            else
            {
                MessageBox.Show($"Ai raspuns corect la {scor} intrebari din
{intrebari.Length}.");
```

```
            }
        }
        else
        {
            MessageBox.Show("Selectează un răspuns înainte de a trece la
următoarea întrebare.");
        }
    }
    private void VerificaRaspuns()
    {
        string raspunsCorect = raspunsuri[intrebareCurenta][3];
        if ((radioButton1.Checked && radioButton1.Text == raspunsCorect) ||
            (radioButton2.Checked && radioButton2.Text == raspunsCorect) ||
            (radioButton3.Checked && radioButton3.Text == raspunsCorect))
        {
            scor++;
        }
    }
}
```

19. Quiz 2

Inițializarea quiz-ului: În constructorul clasei quiz2, se încarcă întrebările și răspunsurile din fișierul text utilizând metoda CitesteEnunturiSiRaspunsuri, apoi se generează interfața utilizând metoda GenerareInterfata.

Citirea întrebărilor și răspunsurilor: Metoda CitesteEnunturiSiRaspunsuri citește fișierul text linie cu linie și împarte fiecare linie în întrebare și răspunsuri, stocând aceste informații într-o listă de șiruri de caractere.

Generarea interfeței grafice: Metoda GenerareInterfata creează o etichetă și un câmp text pentru fiecare întrebare și le adaugă în fereastra quiz-ului. De asemenea, adaugă un buton pentru verificarea răspunsurilor și stabilește evenimentul de

click al acestuia pentru a apela metoda VerificaRaspunsuri.

Verificarea răspunsurilor: În metoda VerificaRaspunsuri, se parcurg toate întrebările și se compară răspunsurile utilizatorului cu cele corecte. Dacă un răspuns este corect, eticheta corespunzătoare devine verde, altfel devine roșie. Se calculează și se afișează scorul final.

Redimensionarea interfeței grafice: Metoda RedimensionareInterfata se ocupă de redimensionarea elementelor interfeței grafice atunci când fereastra quiz-ului este redimensionată.

```
public partial class quiz2 : Form

{

    private List<string[]> enunturiSiRaspunsuri = new List<string[]>();

    private List<System.Windows.Forms.TextBox> textBoxes = new
List<System.Windows.Forms.TextBox>();

    private List<Label> labels = new List<Label>();

    private Font fontEnunt = new Font("Arial", 14, FontStyle.Regular);

    private Font fontRaspuns = new Font("Arial", 12, FontStyle.Regular);

    private void button1_Click(object sender, EventArgs e)

    {



    }



    int[] vf = new int[100];

    public quiz2()

    {

        InitializeComponent();

        CitesteEnunturiSiRaspunsuri("quiz2.txt");

        GenerareInterfata();

    }

    private void CitesteEnunturiSiRaspunsuri(string numeFisier)

    {
```

```
        try
        {
            string[] linii = File.ReadAllLines(numeFisier);
            foreach (string linie in linii)
            {
                string[] enuntSiRaspunsuri = linie.Split('/');
                enunturiSiRaspunsuri.Add(enuntSiRaspunsuri);
            }
        }
        catch (IOException e)
        {
            MessageBox.Show("Eroare la citirea fișierului: " + e.Message);
        }
    }

    private void GenerareInterfata()
    {
        int yOffset = 50;
        foreach (string[] enuntSiRaspunsuri in enunturiSiRaspunsuri)
        {
            Label label = new Label();
            label.Text = enuntSiRaspunsuri[0];
            label.Font = fontEnunt;
            label.AutoSize = true;
            label.Location = new Point(50, yOffset);
            labels.Add(label);
            this.Controls.Add(label);

            System.Windows.Forms.TextBox textBox = new
System.Windows.Forms.TextBox();
```

```
            textBox.Font = fontRaspuns;
            textBox.Location = new Point(50 + label.Width + 10, yOffset - 3);
            textBox.Width = this.ClientSize.Width - label.Width - 100;
            textBoxes.Add(textBox);
            this.Controls.Add(textBox);

            yOffset += 40;
        }

        System.Windows.Forms.Button verificareButton = new
System.Windows.Forms.Button();
        verificareButton.Text = "Verifică răspunsurile";
        verificareButton.Size = new Size(120, 50);
        verificareButton.BackColor = Color.MediumPurple;
        verificareButton.Font = fontEnunt;
        verificareButton.Location = new Point(this.Width / 2 -
verificareButton.Width / 2, this.Height - verificareButton.Height * 2);
        verificareButton.Click += VerificaRaspunsuri;
        this.Controls.Add(verificareButton);
        this.Resize += quiz2_Resize;
    }

    private void VerificaRaspunsuri(object sender, EventArgs e)
    {
        int scor = 0;
        for (int i = 0; i < enunturiSiRaspunsuri.Count; i++)
        {
            string[] raspunsuriCorecte =
enunturiSiRaspunsuri[i].Skip(1).ToArray();
            string raspunsUtilizator = textBoxes[i].Text.Trim();
```

```
            if (raspunsuriCorecte.Contains(raspunsUtilizator))
            {
                labels[i].ForeColor = Color.Green;
                scor++;
            }
            else
            {
                labels[i].ForeColor = Color.Red;
            }
        }

        MessageBox.Show("Scorul tău este: " + scor + " din " +
enunturiSiRaspunsuri.Count);
        this.Close();
    }
    private void quiz2_Load(object sender, EventArgs e)
    {
        RedimensionareInterfata();
    }
    private void RedimensionareInterfata()
    {
        for (int i = 0; i < enunturiSiRaspunsuri.Count; i++)
        {
            labels[i].Location = new Point(50, 50 + 40 * i);
            textBoxes[i].Location = new Point(50 + labels[i].Width + 10, 50 + 40
* i - 3);
            textBoxes[i].Width = this.ClientSize.Width - labels[i].Width - 100;
        }
    }
```

```
    private void quiz2_Resize(object sender, EventArgs e)
    {
        RedimensionareInterfata();
    }
}
```

20. Quiz 3

Inițializare și încărcare întrebări: În constructorul clasei quiz3, se leagă evenimentele de click ale butoanelor "Adevărat" și "Fals" la metoda verifica_raspuns. Apoi, în metoda button2 _Click, se încarcă întrebările și răspunsurile din fișierul text quiz3.txt și se afișează prima întrebare.

Verificarea răspunsurilor: În metoda verifica_raspuns, se verifică răspunsul selectat de utilizator. Dacă răspunsul este corect, scorul este incrementat. Apoi se afișează următoarea întrebare sau, dacă utilizatorul a răspuns la toate întrebările, se afișează scorul final.

Afișarea întrebărilor: Metoda scrie_intrebarea afișează întrebarea curentă pe eticheta label4.

```
public partial class quiz3 : Form
{
    string raspuns;
    int n1 = 0;
    int n_intrebare = 0;
    int scor = 0;
    int[] v1 = new int[100];
    System.Windows.Forms.CheckBox[] c = new System.Windows.Forms.CheckBox[10];
    int[] v = new int[100];
    string s;
    Class3[] vect = new Class3[100];
    void verificare()
    {
```

```
}
public quiz3()
{
    InitializeComponent();
    button1.Click += verifica_raspuns;
    button4.Click += verifica_raspuns;
}

private void quiz3_Shown(object sender, EventArgs e)
{

}
private void verifica_raspuns(object sender, EventArgs e)
{
    Button buton_apasat = sender as Button;
    if (buton_apasat.Text == "Adevărat")
    {
        raspuns = "da";
    }
    else
    {
        raspuns = "nu";
    }
    if (raspuns == vect[n_intrebare].raspuns)
    {
        scor++;
    }
    n_intrebare++;
```

```
        if (n_intrebare < n1)
        {
            scrie_intrebarea();
        }
        else
        {
            label4.Text = "Ai răspuns corect la " + scor + " din " + n1 + "
întrebări";
            button1.Visible = false;
            button4.Visible = false;
        }
    }
    private void scrie_intrebarea()
    {
        label4.Text = vect[n_intrebare].test;
    }

    private void button2_Click(object sender, EventArgs e)
    {
        button2.Visible = false;
        label4.Visible = true;
        button1.Visible = true;
        button4.Visible = true;
        using (StreamReader str = new StreamReader("quiz3.txt"))
        {
            string linie;
            while ((linie = str.ReadLine()) != null)
            {
                vect[n1] = new Class3();
                vect[n1].test = linie;
```

```
                vect[n1].raspuns = str.ReadLine();
                n1++;
            }
        }
        scrie_intrebarea();
    }
}
```

21. X și 0

Inițializarea tabelului: În metoda InitializeazaTabla, se creează și se plasează butoanele pentru tabelul X și O pe fereastră. Aceste butoane sunt aranjate într-o matrice 3x3.

Acțiunea de clic pe buton: Când un utilizator face clic pe un buton, metoda Buton_Click este activată. În această metodă, dacă jocul nu s-a încheiat încă, butonul pe care utilizatorul a făcut clic primește textul jucătorului curent (X sau O) și este dezactivat. Apoi, este verificat dacă un jucător a câștigat sau dacă este remiză, iar dacă nu, se trece la următorul jucător sau la mișcarea botului.

Mișcare bot: Metoda MiscareBot este responsabilă pentru mișcarea botului. În această metodă, botul analizează tabla și alege mișcarea optimă pentru a câștiga sau a bloca un adversar. Dacă nu există o mișcare strategică imediată, botul alege o mișcare aleatoare.

Verificarea câștigătorului: Metoda VerificaCastig este folosită pentru a verifica dacă un jucător a câștigat. Ea verifică liniile, coloanele și diagonalele pentru a vedea dacă toate butoanele de pe acea linie/coloană/diagonală sunt ocupate de același jucător.

Verificarea remizei: Metoda VerificaRemiza este folosită pentru a verifica dacă jocul s-a terminat cu remiză. Dacă toate butoanele sunt ocupate și niciun jucător nu a câștigat, atunci jocul se încheie cu remiză.

Resetarea jocului: Metoda Reseteaza este folosită pentru a reseta tabla și a începe un nou joc.

```
public partial class x_0 : Form
{
    private Button[,] tabla;
    private char jucatorCurent;
    private bool jocTerminat;
    public x_0()
    {
        InitializeComponent();
        InitializeazaTabla();
        jucatorCurent = 'X';
        jocTerminat = false;
    }
    private void InitializeazaTabla()
    {
        const int marime = 3;
        const int marimeButon = 100;
        const int padding = 5;

        tabla = new Button[marime, marime];

        for (int i = 0; i < marime; i++)
        {
            for (int j = 0; j < marime; j++)
            {
                Button buton = new Button
                {
                    BackColor = Color.Indigo,
                    Size = new System.Drawing.Size(marimeButon, marimeButon),
                    Location = new System.Drawing.Point(j * (marimeButon +
padding) + 30, i * (marimeButon + padding) + 60),
```

```
                    Font = new System.Drawing.Font("Arial", 36F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0))),
                    Tag = new System.Drawing.Point(i, j),
                    TextAlign = System.Drawing.ContentAlignment.MiddleCenter,
                    Enabled = true
                };
                buton.Click += Buton_Click;
                Controls.Add(buton);
                tabla[i, j] = buton;
            }
        }
    }
    private void Buton_Click(object sender, EventArgs e)
    {
        if (jocTerminat) return;

        Button buton = (Button)sender;
        buton.Text = jucatorCurent.ToString();
        buton.Enabled = false;

        int rand = ((System.Drawing.Point)buton.Tag).X;
        int coloana = ((System.Drawing.Point)buton.Tag).Y;

        if (VerificaCastig(rand, coloana, jucatorCurent))
        {
            MessageBox.Show($"Jucătorul {jucatorCurent} a câștigat!"); //trebuie
schimbat cu un label
            jocTerminat = true;
            Reseteaza();
        }
```

```
    else if (VerificaRemiza())
    {
        MessageBox.Show("E remiză!"); //trebuie schimbat cu un label si asta
        jocTerminat = true;
        Reseteaza();
    }
    else
    {
        jucatorCurent = (jucatorCurent == 'X') ? 'O' : 'X';
        if (jucatorCurent == 'O')
        {
            MiscareBot();
        }
    }
}
private void MiscareBot()
{
    int[] miscare = AlegeMiscareOptima();
    tabla[miscare[0], miscare[1]].PerformClick();
}
private int[] AlegeMiscareOptima()
{
    //MERGEEEEEEEEEE
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (tabla[i, j].Text == "")
            {
```

```
            tabla[i, j].Text = "O";
            if (VerificaCastig(i, j, 'O'))
            {
                tabla[i, j].Text = "";
                return new int[] { i, j };
            }
            tabla[i, j].Text = "";
        }
    }
}

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (tabla[i, j].Text == "")
        {
            tabla[i, j].Text = "X";
            if (VerificaCastig(i, j, 'X'))
            {
                tabla[i, j].Text = "";
                return new int[] { i, j };
            }
            tabla[i, j].Text = "";
        }
    }
}

List<int[]> pozitiiStrategice = new List<int[]>
```

```
{
    new int[] { 1, 1 },
    new int[] { 0, 0 },
    new int[] { 0, 2 },
    new int[] { 2, 0 },
    new int[] { 2, 2 }
};

Random random = new Random();
foreach (var pozitie in pozitiiStrategice.OrderBy(x => random.Next()))
{
    if (tabla[pozitie[0], pozitie[1]].Text == "")
    {
        return pozitie;
    }
}

List<int[]> pozitiiLibere = new List<int[]>();
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (tabla[i, j].Text == "")
        {
            pozitiiLibere.Add(new int[] { i, j });
        }
    }
}
return pozitiiLibere[random.Next(pozitiiLibere.Count)];
```

```
}


private bool VerificaCastig(int rand, int coloana, char jucator)
{
    for (int i = 0; i < 3; i++)
    {
        if (tabla[rand, i].Text != jucator.ToString()) break;
        if (i == 2) return true;
    }

    for (int i = 0; i < 3; i++)
    {
        if (tabla[i, coloana].Text != jucator.ToString()) break;
        if (i == 2) return true;
    }

    if (rand == coloana)
    {
        for (int i = 0; i < 3; i++)
        {
            if (tabla[i, i].Text != jucator.ToString()) break;
            if (i == 2) return true;
        }
    }

    if (rand + coloana == 2)
    {
        for (int i = 0; i < 3; i++)
```

```
        {
            if (tabla[i, 2 - i].Text != jucator.ToString()) break;
            if (i == 2) return true;
        }
    }

    return false;
}
private bool VerificaRemiza()
{
    foreach (Button buton in tabla)
    {
        if (buton.Text == "") return false;
    }
    return true;
}
private void Reseteaza()
{
    foreach (Button buton in tabla)
    {
        buton.Text = "";
        buton.Enabled = true;
    }
    jucatorCurent = 'X';
    jocTerminat = false;
}
private void x_0_Load(object sender, EventArgs e)
{
```

```
    }
}
```

22. Puzzle

OpenFileEvent: Această funcție se activează atunci când utilizatorul dorește să deschidă o imagine pentru a o folosi în joc. Utilizatorul poate selecta un fișier de imagine, iar apoi imaginea este încărcată și afișată în fereastra de joc.

CreazaImagini: Această funcție creează 9 imagini mici din imaginea principală, fiecare imagine mică reprezentând o bucată a puzzle-ului. Aceste imagini sunt stocate într-o listă de imagini.

OnPicClick: Această funcție este activată atunci când utilizatorul face clic pe o imagine mică din puzzle. Dacă imaginea mică pe care a făcut clic utilizatorul este vecină cu imaginea goală, atunci cele două imagini sunt interschimbate, permițând astfel mutarea bucății puzzle-ului. Apoi, funcția verifică dacă imaginea este plasată în poziția corectă pentru a rezolva puzzle-ul.

CropImage: Această funcție taie imaginea principală în bucăți mai mici, fiecare reprezentând o bucată a puzzle-ului. Imaginile tăiate sunt adăugate într-o listă de imagini.

AdaugaImagini: Această funcție adaugă imaginile tăiate la controalele PictureBox din formular și le plasează aleatoriu pe ecran pentru a începe jocul.

PlacePictureBoxesToForm: Această funcție plasează controalele PictureBox pe formular și stabilește pozițiile lor inițiale pe ecran.

CheckGame: Această funcție verifică dacă poziția curentă a bucăților de puzzle este identică cu poziția corectă pentru a rezolva puzzle-ul. Dacă poziția este corectă, se afișează un mesaj de felicitare și jocul este încheiat.

```
public partial class puzzle : Form

{

    private int miscariEfectuate = 0;
```

```
List<PictureBox> listaPictureBox = new List<PictureBox>();
List<Bitmap> imagini = new List<Bitmap>();
List<string> locatii = new List<string>();
List<string> locatiiCurente = new List<string>();
string pozitieCastig;
string pozitieCurenta;
Bitmap imaginePrincipala;
private int secunde, minut;

public puzzle()
{
    InitializeComponent();
    timer1.Interval = 1000;
}

private void OpenFileEvent(object sender, EventArgs e)
{
    if (listaPictureBox != null)
    {
        foreach (PictureBox pictureBox in listaPictureBox.ToList())
        {
            this.Controls.Remove(pictureBox);
        }

        listaPictureBox.Clear();
        imagini.Clear();
        locatii.Clear();
        locatiiCurente.Clear();
        pozitieCastig = string.Empty;
```

```
            pozitieCurenta = string.Empty;
        }

        OpenFileDialog open = new OpenFileDialog();
        open.Filter = "Image Files Only | *.jpg; *.jpeg; *.gif; *.png";

        if (open.ShowDialog() == DialogResult.OK)
        {
            imaginePrincipala = new Bitmap(open.FileName);
            CreazaImagini();
            AdaugaImagini();
            timer1.Start();
        }
    }

    private void CreazaImagini()
    {
        for (int i = 0; i < 9; i++)
        {
            PictureBox pictureBoxTemp = new PictureBox();
            pictureBoxTemp.Size = new Size(130, 130);
            pictureBoxTemp.Tag = i.ToString();
            pictureBoxTemp.Click += OnPicClick;
            listaPictureBox.Add(pictureBoxTemp);
            locatii.Add(pictureBoxTemp.Tag.ToString());
        }
    }

    private void OnPicClick(object sender, EventArgs e)
```

```
    {

        PictureBox pictureBoxApasat = (PictureBox)sender;

        PictureBox pictureBoxGol = listaPictureBox.Find(x => x.Tag.ToString() ==
"0");

        Point punct1 = new Point(pictureBoxApasat.Location.X,
pictureBoxApasat.Location.Y);

        Point punct2 = new Point(pictureBoxGol.Location.X,
pictureBoxGol.Location.Y);

        int index1 = this.Controls.IndexOf(pictureBoxApasat);

        int index2 = this.Controls.IndexOf(pictureBoxGol);


        if ((pictureBoxApasat.Right == pictureBoxGol.Left &&
pictureBoxApasat.Location.Y == pictureBoxGol.Location.Y)

            || (pictureBoxApasat.Left == pictureBoxGol.Right &&
pictureBoxApasat.Location.Y == pictureBoxGol.Location.Y)

            || (pictureBoxApasat.Top == pictureBoxGol.Bottom &&
pictureBoxApasat.Location.X == pictureBoxGol.Location.X)

            || (pictureBoxApasat.Bottom == pictureBoxGol.Top &&
pictureBoxApasat.Location.X == pictureBoxGol.Location.X))

        {

            miscariEfectuate++;

            label1.Text = "Mișcări efectuate: " + miscariEfectuate;

            pictureBoxApasat.Location = punct2;

            pictureBoxGol.Location = punct1;

            this.Controls.SetChildIndex(pictureBoxApasat, index2);

            this.Controls.SetChildIndex(pictureBoxGol, index1);

        }


        locatiiCurente.Clear();

        CheckGame();

    }


    private void CropImage(Bitmap imaginePrincipala, int inaltime, int latime)
```

```
    {
        int x, y;
        x = 0;
        y = 0;

        for (int blocuri = 0; blocuri < 9; blocuri++)
        {
            Bitmap imagineTaiata = new Bitmap(inaltime, latime);

            for (int i = 0; i < inaltime; i++)
            {
                for (int j = 0; j < latime; j++)
                {
                    imagineTaiata.SetPixel(i, j, imaginePrincipala.GetPixel((i +
x), (j + y)));
                }
            }

            imagini.Add(imagineTaiata);
            x += 130;

            if (x == 390)
            {
                x = 0;
                y += 130;
            }
        }
    }

    private void AdaugaImagini()
```

```
    {
        Bitmap imagineTemporara = new Bitmap(imaginePrincipala, new Size(390,
390));
        CropImage(imagineTemporara, 130, 130);

        for (int i = 1; i < listaPictureBox.Count; i++)
        {
            listaPictureBox[i].BackgroundImage = (Image)imagini[i];
        }
        PlacePictureBoxesToForm();
    }

    private void PlacePictureBoxesToForm()
    {
        var imaginiAmestecate = listaPictureBox.OrderBy(a =>
Guid.NewGuid()).ToList();
        listaPictureBox = imaginiAmestecate;

        int x = 50;
        int y = 15;

        for (int i = 0; i < listaPictureBox.Count; i++)
        {
            listaPictureBox[i].BackColor = Color.Purple;

            if (i == 3 || i == 6)
            {
                y += 130;
                x = 50;
            }
```

```
        listaPictureBox[i].BorderStyle = BorderStyle.FixedSingle;
        listaPictureBox[i].Location = new Point(x, y);
        this.Controls.Add(listaPictureBox[i]);
        x += 130;
        pozitieCastig += locatii[i];
    }
}

private void CheckGame()
{
    foreach (Control control in this.Controls)
    {
        if (control is PictureBox pictureBox)
        {
            locatiiCurente.Add(pictureBox.Tag.ToString());
        }
    }

    string pozitieCurenta = string.Join("", locatiiCurente);
    if (pozitieCastig == pozitieCurenta)
    {
        MessageBox.Show("Felicitări! Ai potrivit imaginea.");
        this.Close();
    }
}

private void button1_Click(object sender, EventArgs e)
{
```

```
        miscariEfectuate = 0;
        label1.Text = "Mișcări efectuate: ";
        if (listaPictureBox != null)
        {
            foreach (PictureBox pictureBox in listaPictureBox.ToList())
            {
                this.Controls.Remove(pictureBox);
            }

            listaPictureBox.Clear();
            imagini.Clear();
            locatii.Clear();
            locatiiCurente.Clear();
            pozitieCastig = string.Empty;
            pozitieCurenta = string.Empty;
        }

        OpenFileDialog open = new OpenFileDialog();
        open.Filter = "Image Files Only | *.jpg; *.jpeg; *.gif; *.png";

        if (open.ShowDialog() == DialogResult.OK)
        {
            imaginePrincipala = new Bitmap(open.FileName);
            CreazaImagini();
            AdaugaImagini();
        }
    }
}
```

23. Minesweeper

Funcția PornesteJoc() inițializează jocul, creând butoane pentru fiecare căsuță a tabloului și configurându-le dimensiunea, locația și evenimentul de click.

PlaseazaMine() este responsabilă de plasarea minelor pe tablă. Ea generează aleatoriu locațiile minelor și se asigură că acestea sunt distribuite uniform.

Atunci când un jucător face clic pe o căsuță a tabloului, este activată funcția buton_Click(). Aceasta verifică dacă căsuța conține o mină sau nu și dezvăluie căsuțele goale din jurul acesteia, dacă este cazul.

NumaraBombe(int x, int y): Numără câte mine sunt învecinate cu o anumită căsuță de la coordonatele (x, y).

UmpleCeluleleGoale(int x, int y): Descoperă recursiv toate căsuțele goale din jurul unei căsuțe care a fost descoperită și care nu are mine în vecinătate.

VerificaCastigat(): Verifică dacă toate căsuțele care nu conțin mine au fost descoperite, ceea ce înseamnă că jucătorul a câștigat jocul.

```
   private static int numarColoane = meniu_minesweeper.numarColoane;
    private static int numarMine = meniu_minesweeper.numarMine;
    Color culoare;
    private Button[,] panou = new Button[numarColoane, numarColoane];
    private bool[,] eMina = new bool[numarColoane, numarColoane];
    private int unrevealedNonBombCells;
    public Form8()
    {
        InitializeComponent();
        PornesteJoc();
        this.Resize += Form8_Resize;
        PlaseazaMine();
        unrevealedNonBombCells = numarColoane * numarColoane - numarMine;
    }
```

```
    private void PornesteJoc()
    {
        const int buttonSize = 30;
        const int padding = 5;
        const int formMargin = 20;

        for (int i = 0; i < numarColoane; i++)
        {
            for (int j = 0; j < numarColoane; j++)
            {
                var button = new Button
                {
                    Size = new Size(buttonSize, buttonSize),
                    Location = new Point(j * (buttonSize + padding), i *
(buttonSize + padding)),
                    Tag = new Point(i, j),
                    TextAlign = ContentAlignment.MiddleCenter,
                };

                if (eMina[i, j])
                {
                    button.Text = "";
                }

                button.Click += buton_Click;
                Controls.Add(button);
                panou[i, j] = button;
            }
        }
```

```
    int formWidth = numarColoane * (buttonSize + padding) + formMargin;
    int formHeight = numarColoane * (buttonSize + padding) + formMargin;

    this.MinimumSize = new Size(formWidth, formHeight);

    Size = new Size(formWidth, formHeight);
}
private void PlaseazaMine()
{
    Random random = new Random();
    for (int i = 0; i < numarMine; i++)
    {
        int x = random.Next(0, numarColoane);
        int y = random.Next(0, numarColoane);
        if (!eMina[x, y])
        {
            eMina[x, y] = true;
            panou[x, y].Text = "";
        }
        else
        {
            i--;
        }
    }
}
private void buton_Click(object sender, EventArgs e)
{
    Button button = (Button)sender;
    Point location = (Point)button.Tag;
```

```
        int x = location.X;
        int y = location.Y;

        if (eMina[x, y])
        {
            DezvalueMine();
            MessageBox.Show("Ai declanșat o bombă! Joc pierdut.");
            Reseteaza();
        }
        else
        {
            int bombCount = NumaraBombe(x, y);
            button.Text = bombCount.ToString();
            culoare =  culoareNumar(bombCount);
            button.ForeColor = culoare;
            button.Enabled = false;

            if (bombCount == 0)
            {
                UmpleCeluleleGoale(x, y);
            }

            if (VerificaCastigat())
            {
                MessageBox.Show("Felicitări! Ai câștigat.");
                Reseteaza();
            }
        }
    }
```

```
    private void DezvaluieMine()
    {
        for (int i = 0; i < numarColoane; i++)
        {
            for (int j = 0; j < numarColoane; j++)
            {
                if (eMina[i, j])
                {
                    panou[i, j].Text = "*";
                }
            }
        }
    }
    private int NumaraBombe(int x, int y)
    {
        int count = 0;
        for (int i = Math.Max(0, x - 1); i <= Math.Min(numarColoane - 1, x + 1);
i++)
        {
            for (int j = Math.Max(0, y - 1); j <= Math.Min(numarColoane - 1, y +
1); j++)
            {
                if (eMina[i, j])
                {
                    count++;
                }
            }
        }
        return count;
    }
```

```
    private void resizeText()
    {
        const int spatiu = 5;
        const int minMarimeButon = 30;
        const int minMarimeFont = 8;
        const int maxMarimeFont = 16;

        int marimeButon = Math.Max(Math.Min((ClientSize.Width - (numarColoane +
1) * spatiu) / numarColoane, (ClientSize.Height - (numarColoane + 1) * spatiu) /
numarColoane), minMarimeButon);

        float marimeFont = Math.Max(Math.Min((float)marimeButon / 2,
maxMarimeFont), minMarimeFont);

        Font font = new Font("Arial", marimeFont);

        foreach (Button buton in panou)
        {
            buton.Font = font;
        }
    }
    private void resizeButoane()
    {
        const int spatiu = 1;

        int maxButtonSize = Math.Min((ClientSize.Width - (numarColoane + 1) *
spatiu) / numarColoane, (ClientSize.Height - (numarColoane + 1) * spatiu) /
numarColoane);

        int buttonSize = Math.Max(maxButtonSize, 10);

        for (int i = 0; i < numarColoane; i++)
```

```
        {
            for (int j = 0; j < numarColoane; j++)
            {
                var button = panou[i, j];
                button.Size = new Size(buttonSize, buttonSize);
                button.Location = new Point(spatiu + j * (buttonSize + spatiu),
spatiu + i * (buttonSize + spatiu));
            }
        }
    }
    private void UmpleCeluleleGoale(int x, int y)
    {
        for (int i = Math.Max(0, x - 1); i <= Math.Min(numarColoane - 1, x + 1);
i++)
        {
            for (int j = Math.Max(0, y - 1); j <= Math.Min(numarColoane - 1, y +
1); j++)
            {
                if (panou[i, j].Enabled && NumaraBombe(i, j) == 0)
                {
                    panou[i, j].Enabled = false;
                    UmpleCeluleleGoale(i, j);
                }
                else if (panou[i, j].Enabled)
                {
                    panou[i, j].Text = NumaraBombe(i, j).ToString();
                    panou[i, j].Enabled = false;
                }
            }
        }
    }
```

```
private bool VerificaCastigat()
{
    for (int i = 0; i < numarColoane; i++)
    {
        for (int j = 0; j < numarColoane; j++)
        {
            if (!eMina[i, j] && panou[i, j].Enabled)
            {
                return false;
            }
        }
    }
    return true;
}
private void Reseteaza()
{
    foreach (Button buton in panou)
    {
        buton.Text = "";
        buton.Enabled = true;
    }
    Array.Clear(eMina, 0, eMina.Length);
    PlaseazaMine();
}
private void Form8_Load(object sender, EventArgs e)
{

}
```

```
private void Form8_Resize(object sender, EventArgs e)
{
    resizeButoane();
    resizeText();
}
private Color culoareNumar(int number)
{
    switch (number)
    {
        case 1:
            culoare = Color.Blue;
            return culoare;
        case 2:
            culoare = Color.Green;
            return culoare;
        case 3:
            culoare = Color.Red;
            return culoare;
        case 4:
            culoare = Color.DarkBlue;
            return culoare;
        case 5:
            culoare = Color.DarkRed;
            return culoare;
        case 6:
            culoare = Color.Blue;
            return culoare;
        case 7:
            culoare = Color.Blue;
```

```
                return culoare;
            default:
                culoare = Color.Blue;
                return culoare;
        }
    }
}
```