Programsko rješenje logičke zagonetke nonogram u programskom jeziku C#.

Džijan, Matej

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:718571

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PROGRAMSKO RJEŠENJE LOGIČKE ZAGONETKE NONOGRAM U PROGRAMSKOM JEZIKU C#

Završni rad

Matej Džijan

Osijek, 2018.

SADRŽAJ

1. UVOD 1
1.1. Zadatak rada 1
2. RJEŠAVANJE NONOGRAMA 2
2.1. Koraci rješavanja 2
2.2. Primjer rješavanja nonograma 3
3. KORISNIČKO SUČELJE I KORIŠTENI ALATI 8
3.1. Microsoft Visual Studio 9
3.2. Microsoft Visual C# 9
4. VARIJABLE, POMOĆNE METODE, GENERIRANJE NONOGRAMA I PROVJERA RJEŠENJA 10
4.1. Varijable 10
4.2. Pomoćne metode i inicijalizacija Windows Forme 11
4.3. Generiranje nonograma 14
4.3.1. Generiranje prethodno definiranog nonograma “FERIT” 15
4.3.2. Generiranje nonograma unošenjem svih linija uz mrežu 17
4.3.3. Generiranje nonograma iscrtavanjem željene slike 18
4.4. Provjera rješenja nonograma 21
5. ALGORITAM RJEŠAVANJA NONOGRAMA 23
5.1. Rješavanje pojedinog retka (stupca) 23
5.1.1. Generiranje valjanog retka 23
5.1.2. Izmjena trenutnog rješenja retka (stupca) 27
5.1.3. Primjer 28
5.2. Ukupno rješenje nonograma 29
6. ZAKLJUČAK 33
LITERATURA 34
SAŽETAK 35

ABSTRACT 36
ŽIVOTOPIS 37

1. UVOD

Nonogrami, također poznati kao logičke ispunjaljke, su logičke zagonetke u kojima se polja u mreži moraju bojati ili ostaviti prazna prema brojevima iznad i pored mreže. Brojevi izvan mreže govore koliko je polja zaredom ispunjeno. Na primjer, "2 4 3" bi značilo da u danom retku (stupcu) se prvo nalaze ispunjena dva polja za redom, te razmak od jednog ili više polja, zatim četiri ispunjena polja, zatim razmak, i na poslijetku, tri ispunjena polja. Prije prvog ispunjenog polja i nakon zadnjeg ispunjenog polja također mogu stajati prazna polja.

Nonogrami mogu biti crno-bijeli ili u bojama. Kod obojanih nonograma vrijede slična pravila kao kod crno-bijelih, ali polja mogu biti obojana različitim bojama označenim izvan mreže. Dva različito obojena polja ne moraju imati razmak između sebe, odnosno, možemo imati dva plava polja, zatim četiri crvena polja bez praznih polja između istih. Ovaj rad će se baviti isključivo crno-bijelim nonogramima.

1988. godine Non Ishida, japanska grafička urednica, je objavila samo tri mrežne zagonetke pod nazivom "Prozorske umjetničke zagonetke" (eng. "Window Art Puzzles"). Gospodin Tetsuya Nishio je u više-manje isto vrijeme i potpuno neovisno također objavio ovaj tip zagonetke u različitom časopisu. Uz suradnju s Jamesom Dalgetyjem, 1990. godine se ovaj tip zagonetke počinje objavljivati tjedno u nacionalnom časopisu "The Telegraph" pod nazivom "Nonogram". [1]

1.1. Zadatak rada

Zadatak rada je razviti algoritam za rješavanje nonograma te ga implementirati u C# aplikaciji.

2. RJEŠAVANJE NONOGRAMA

Rješavanje nonograma postupak je koji se sastoji od nekoliko koraka. Nakon što se prođu svi ti koraci, postupak se ponavlja sve dok se slika ne dovrši i time riješi dani nonogram. Lakši nonogrami su jednostavna iteracija kroz ove korake, a oni malo teži mogu zahtjevaju korištenje kombiniranja i zaključivanja na temelju kontradikcija.

2.1. Koraci rješavanja

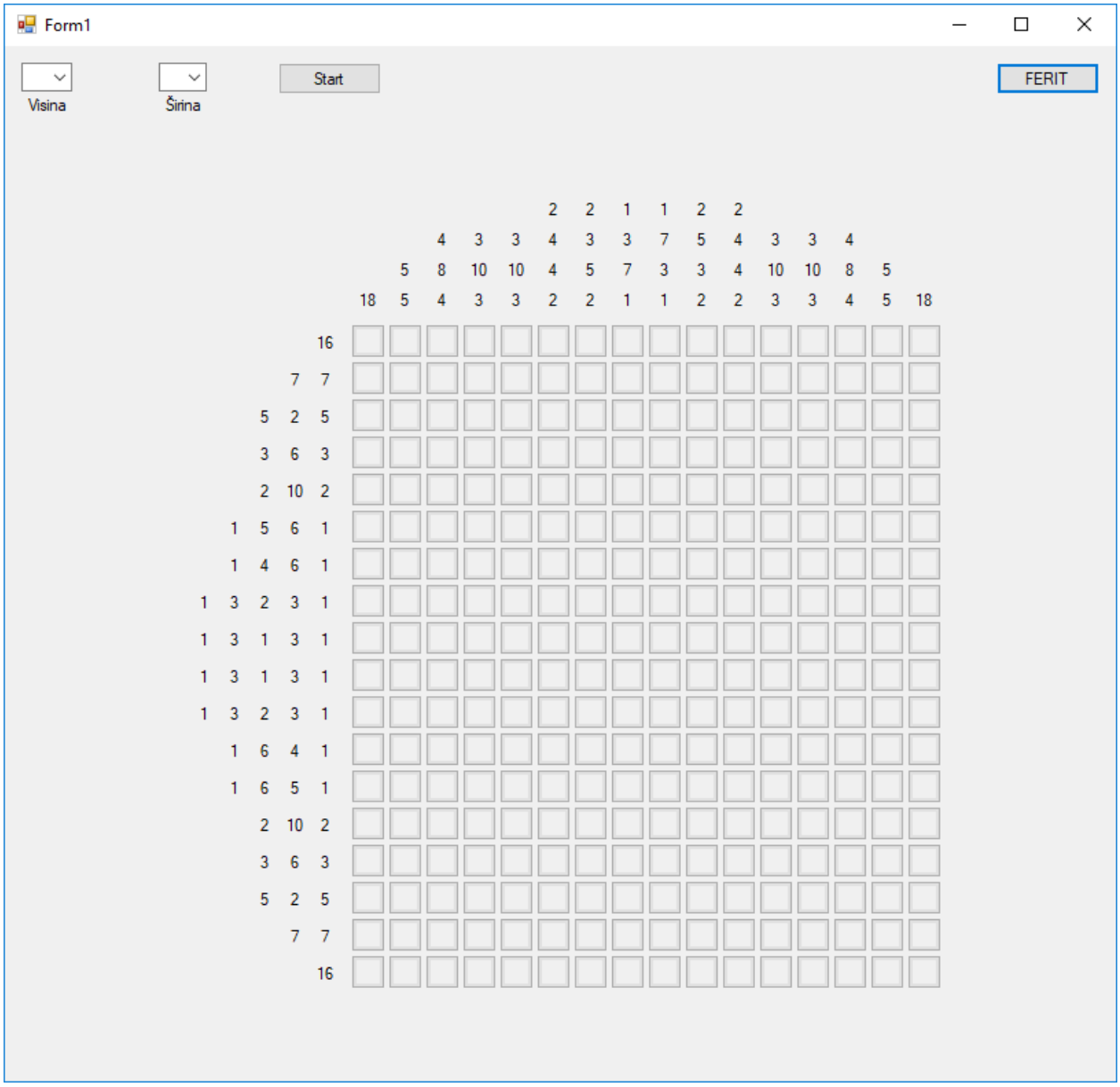
Postupak se sastoji od tri koraka:

- Definirati polja koja moraju biti obojana (u svim mogućim kombinacijama retka/stupca) – takva polja se bojaju;
- Definirati polja koja ne mogu biti obojana – takva polja se prekriže (ili označe točkom);
- Definirati brojeve koji su već riješeni – takvi brojevi se prekriže. [2]

Ovaj proces se ponavlja dok je to moguće ili dok se ne riješi zadani nonogram. Ukoliko dođe do zastoja uz pravilno rješavanje, potrebno je napraviti pretpostavku te nastaviti rješavati kao da je to ispravno. Ukoliko se nakon toga dođe do nemoguće situacije (neki od brojeva ne odgovara), napravljena je kriva pretpostavka i može se zaključiti da je suprotno istinito (npr. ukoliko smo prekrižili polje i tako došli do nemoguće situacije, možemo zaključiti da to polje mora biti obojano).

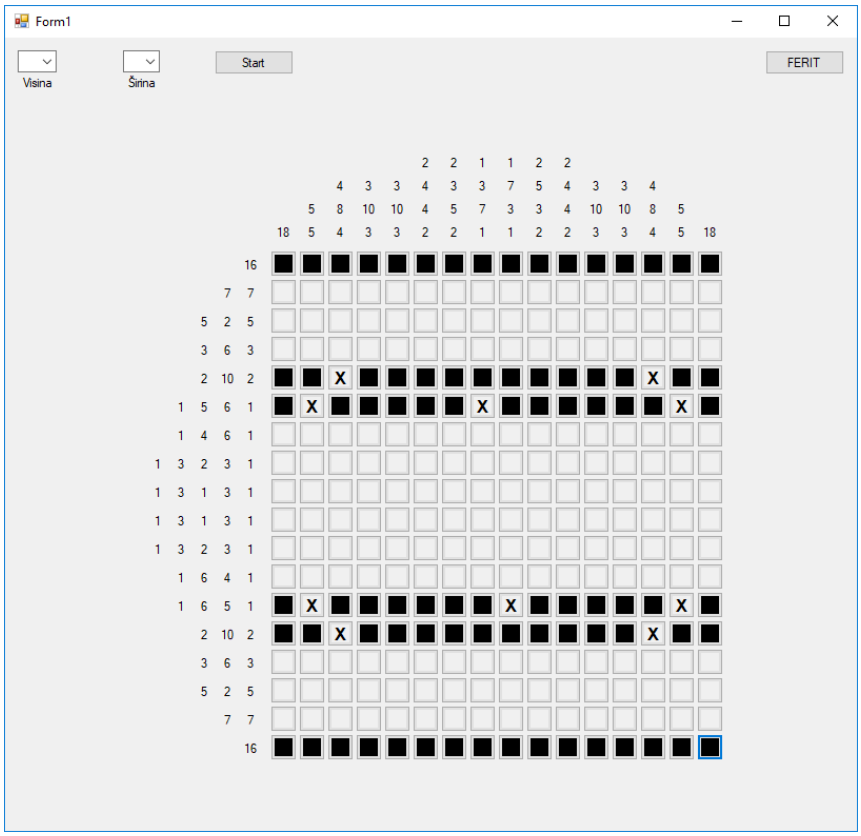
2.2. Primjer rješavanja nonograma

Kroz sljedeće isječke vlastitog programa ću prikazati primjer rješavanja jednostavnog nonograma.

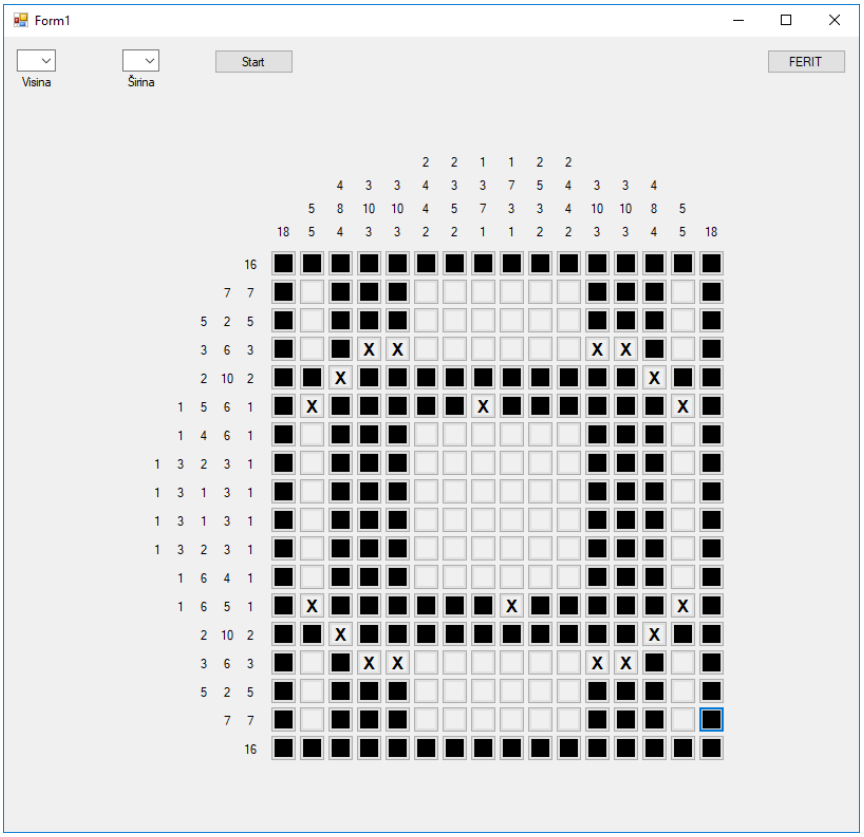


Slika 2.1. Zadani nonogram

Ovdje vidimo zadani nonogram dimenzija 18x16 (visina x širina). Prvi korak je pronaći trivijalne linije, odnosno linije koje možemo odmah cijele ispuniti. Počet ćemo od redaka pa ćemo preći na stupce. Možemo vidjeti da ima 6 takvih redaka i 8 takvih stupaca pa ćemo prvo njih ispuniti.

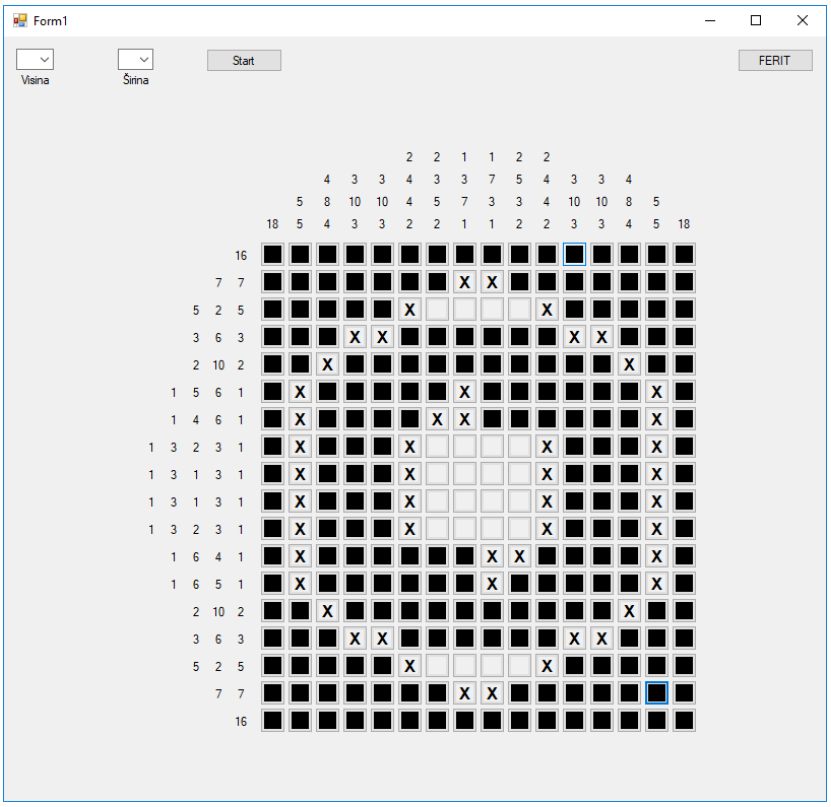


Slika 2.2. Riješeni trivijalni reci

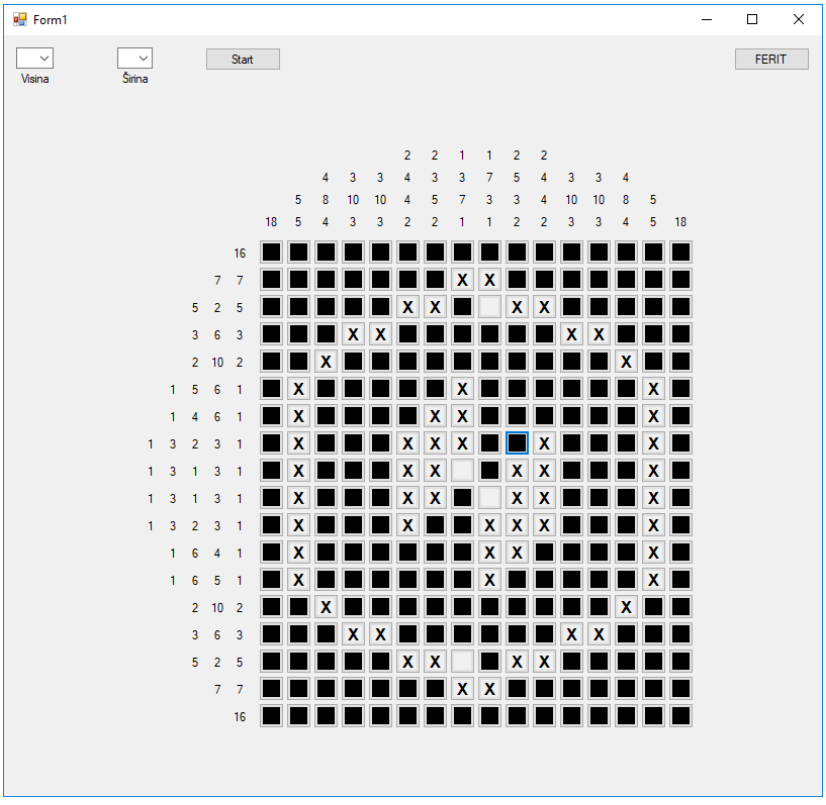


Slika 2.3. Riješeni trivijalni reci i stupci

Sljedeći korak je ispuniti polja koja za znamo da moraju biti ispunjena zatim prekrižiti polja za koja znamo da ne mogu biti ispunjena. Krenut ćemo od redaka pa preći na stupce.

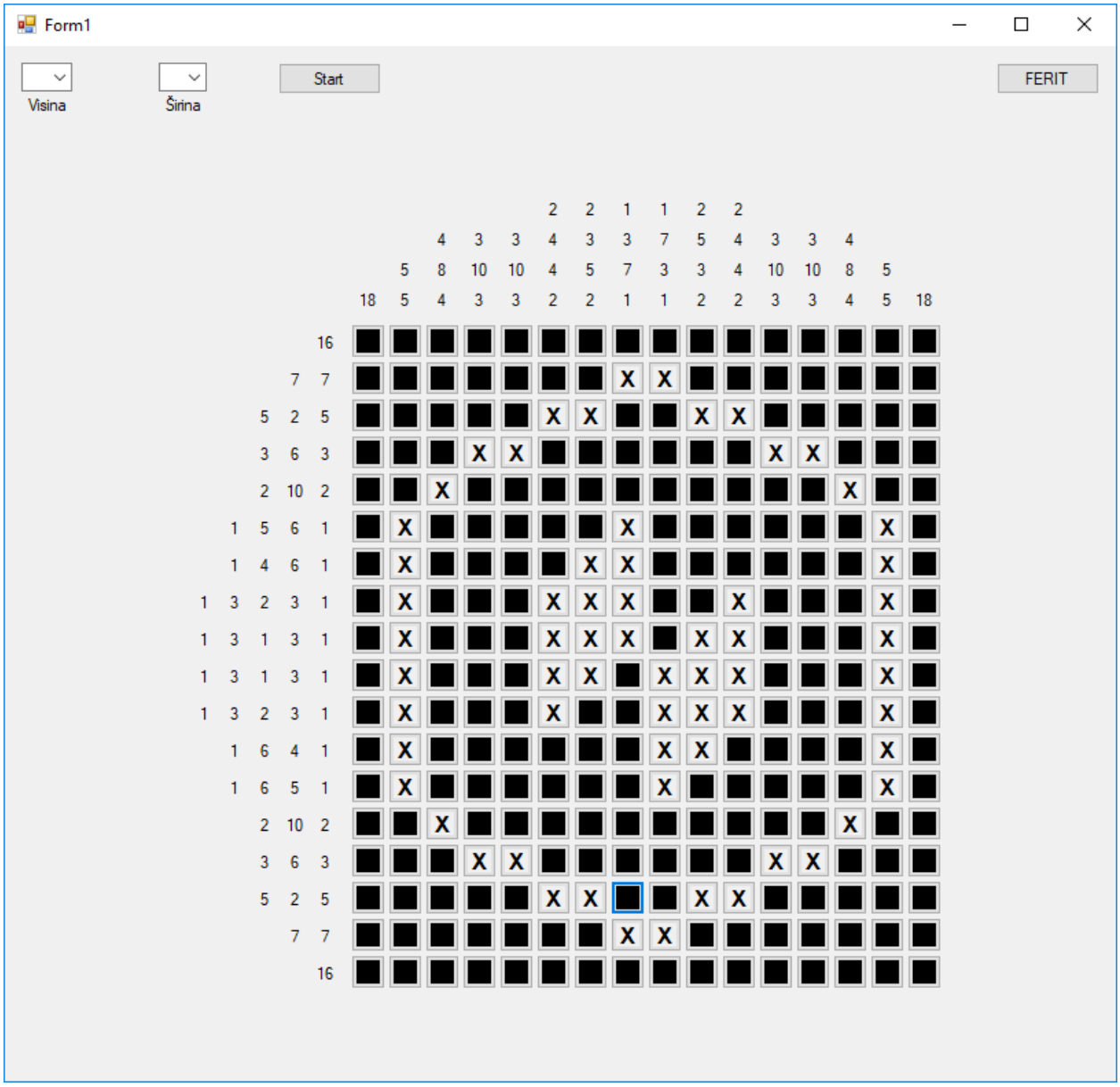


Slika 2.4. Riješeni reci



Slika 2.5. Riješeni reci i stupci

Vidimo da je slika gotovo ispunjena, uz još jednu iteraciju postupka možemo dovršiti sliku.



Slika 2.6. Riješeni nonogram

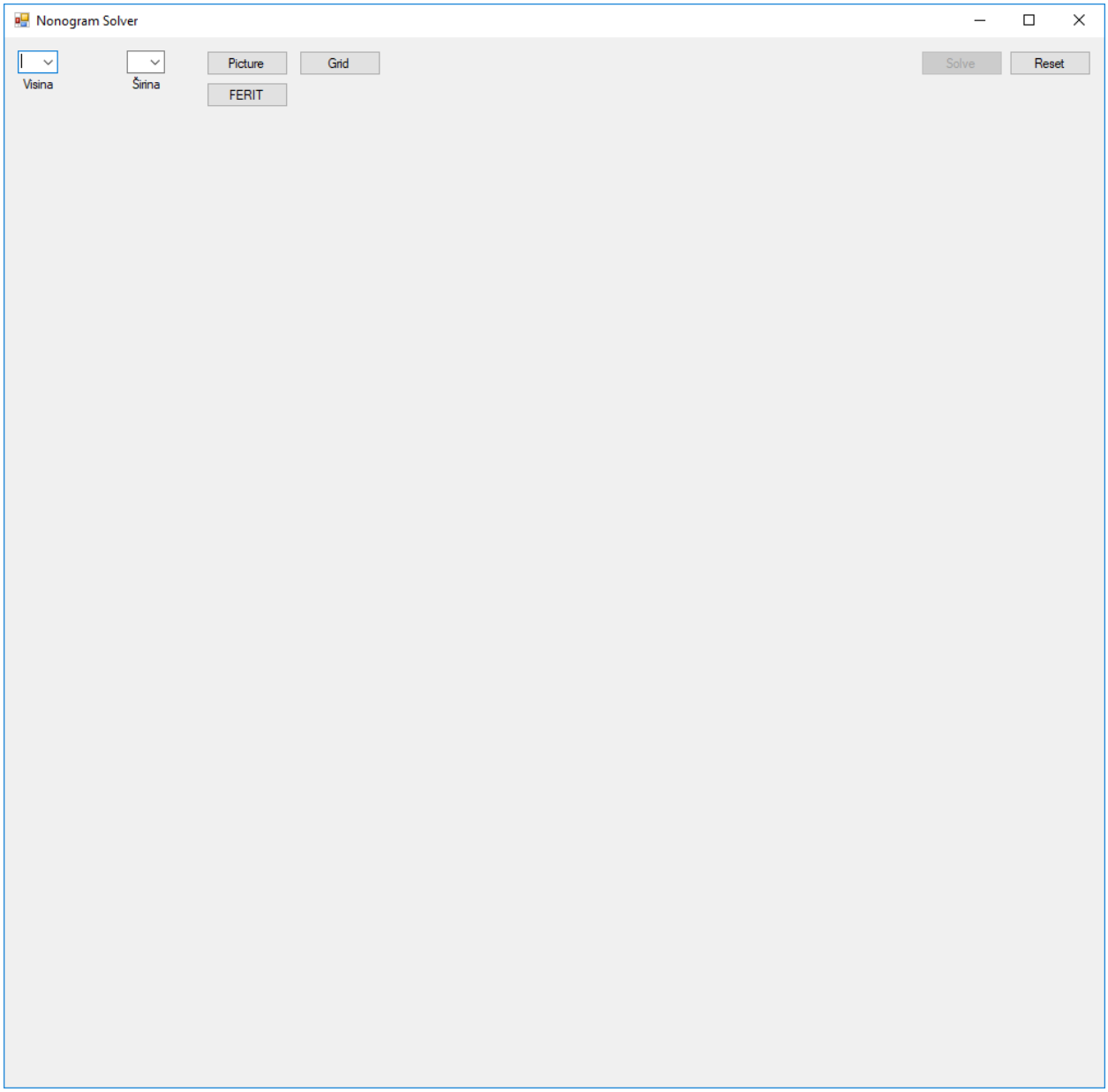
Ovo je bio primjer vrlo jednostavnog nonograma koji kroz samo par iteracija daje konačnu sliku koja predstavlja logo Fakulteta Elektrotehnike, Računarstva i Informacijskih Tehnologija.



Slika 2.7. Logo FERIT-a

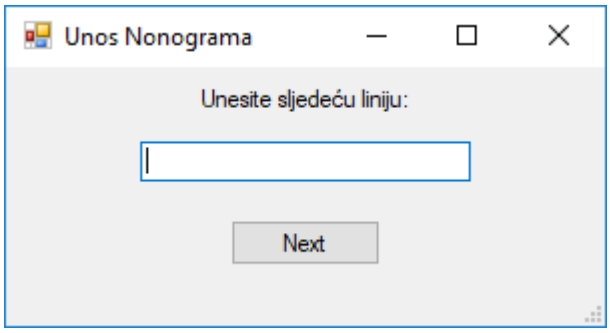
3. KORISNIČKO SUČELJE I KORIŠTENI ALATI

Aplikacija je napravljena u Microsoft Visual Studio okruženju u programskom jeziku C#. Na slici 3.1. se vidi prikaz glavne aplikacije, a na slici 3.2. prikaz pomoćne aplikacije.



Slika 3.1. Glavna aplikacija

Aplikacija se sastoji od padajućih izbornika za visinu i širinu nonograma, gumba “Picture” za generiranje nonograma danih dimenzija pomoću slike, gumba “Grid” za generiranje nonograma danih dimenzija pomoću unosa brojeva sa strane, gumba “FERIT” za generiranje gore riješenog nonograma, gumba “Solve” za rješavanje generiranog nonograma, te gumba “Reset” za ponovno pokretanje aplikacije. Ispod navedenih je prostor za mrežu i brojeve uz istu.



Slika 3.2. Pomoćna aplikacija

Pomoćna aplikacija služi za unos brojeva uz mrežu, poziva se pritiskom na gumb “Grid”. Pobliže objašnjenje načina rada i funkcionalnosti pomoćne aplikacije se nalazi u nastavku rada. Pomoćna aplikacija se sastoji od naljepnice “Unesite sljedeću liniju:”, prostora za pisanje za upis linija nonograma, te gumba “Next” za prijelaz na sljedeću liniju.

3.1. Microsoft Visual Studio

Microsoft Visual studio je integrirano razvojno okruženje. Upotrebljava se za razvoj računalnih programa, web stranica, web aplikacija, web usluga i mobilnih aplikacija. Podržava 36 različitih programskih jezika, a neki od njih su C, C++, C# (C Sharp), Python, JavaScript itd. [3], [4]

3.2. Microsoft Visual C#

Microsoft Visual C# je Microsoftova implementacija programskog jezika C#. Microsoft je oko 2000. godine razvio programski jezik C#. C# je opće namjenski, objektno orijentirani programski jezik. 2018. godine. izbačena je verzija C# 7.3. [5]

4. VARIJABLE, POMOĆNE METODE, GENERIRANJE NONOGRAMA I PROVJERA RJEŠENJA

4.1. Varijable

U ovog poglavlju objašnjene su sve korištene globalne varijabe, te struktura tuple.

```
public struct tuple
{
    public int i, j;
    1 reference
    public tuple(int a, int b)
    {
        i = a;
        j = b;
    }

}
```

Slika 4.1. Struktura tuple

Prikazana je struktura tuple koja sadrži dva cijela broja, “i” i “j”, te konstruktor koji postavlja brojeve “i” i “j” na predane mu vrijednosti. Ona predstavlja uređeni par brojeva.

```
int nonogramHeight, nonogramWidth, coloredSum, currentColored;
Button[,] buttons;
List<List<int>> topNumbers = new List<List<int>>();
List<List<int>> sideNumbers = new List<List<int>>();
List<int[,]> solutions;
int[,] finalSolution;
List<tuple> tried;
Color defaultColor;
bool started, generated, changed;

List<int> combination = new List<int>();
int[] tempLine;
int correctLines = 0;
Stopwatch sw = new Stopwatch();
```

Slika. 4.2. Popis globalnih varijabli

Na slici 4.3. se vidi popis svih korištenih globalnih varijabli. U nastavku, kako se koja varijabla bude koristila, tako će biti opisana svrha dane varijable.

4.2. Pomoćne metode i inicijalizacija Windows Forme

Slijedi popis i objašnjenje pomoćnih metoda, te opis inicijalizacije glavne aplikacije. Pri stvaranju novog objekta razreda Form1 poziva se zadani konstruktor razred Form1.

```
public Form1()
{
    InitializeComponent();
    for (int i = 1; i < 21; i++)
    {
        comboBoxHeigth.Items.Add(i);
        comboBoxWidth.Items.Add(i);
    }
    defaultColor = buttonStart.BackColor;
    currentColored = 0;
    started = false;
    generated = false;
    buttonSolve.Enabled = false;
}
```

Slika 4.3. Zadani konstruktor frome Form1

Konstruktor inicijalizira komponente, zatim popunjava padajuće izbornike za odabir visine i širine nonograma vrijednostima od 1 do 20. Postavlja broj trenutno obojenih polja na 0, postavlja pomoćne varijable “started” i “generated” u “false”, te onemogućava korištenje gumba “Solve”, koji se otključa tek nakon generiranja nonograma.

```
private bool compareLists(List<int> first, List<int> second)
{
    if (first.Count != second.Count)
    {
        return false;
    }
    for (int i = 0; i < first.Count; ++i)
    {
        if (first[i] != second[i])
        {
            return false;
        }
    }
    return true;
}
```

Slika 4.4. Metoda compareLists

Metoda compare Lists prima dva vezana popisa cijelih brojeva te vraća “true” ukoliko su jednaki, a “false” ukoliko nisu. Koristi se u metodama provjere ispravnosti rješenja.

```
private void changeButtonState(int i, int j, bool state)
{
    if (state)
    {
        if (buttons[i, j].BackColor != Color.Black)
        {
            buttons[i, j].Text = "";
            buttons[i, j].BackColor = Color.Black;
            ++currentColored;
            if (!changed)
            {
                changed = true;
            }
        }
    }
    else if (buttons[i, j].Text != "X")
    {

        buttons[i, j].BackColor = defaultColor;
        buttons[i, j].Text = "X";
        if (!changed)
        {
            changed = true;
        }
    }

}
```

Slika 4.5. Metoda changeButtonState

Metoda changeButtonState prima cjelobrojne koordinate “i” i “j” te bool vrijednost “state”. Ukoliko je “state” jednak “true”, metoda boja polje s koordinatama “i” i “j” u crno, u suprotnom, označava polje znakom “X”.

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    Application.Exit();
}
```

Slika 4.6. Metoda Form1_FormClosing

Metoda Form1_FormClosing je pomočna metoda koja, pri izlasku iz bilo koje instance klase Form1, zatvara cijelu aplikaciju. Potrebna je zbog metode za ponovno pokretanje programa.

```
private void buttonReset_Click(object sender, EventArgs e)
{
    Form1 NewForm = new Form1();
    NewForm.Show();
    this.Dispose(false);
}
```

Slika 4.7. Metoda buttonReset_Click

Metoda pravi novu instancu forme Form1 te uništava trenutnu instancu.

```
private void NewPanelButton_MouseDown(object sender, MouseEventArgs e)
{
    Button currentButton = (Button)sender;
    if (e.Button == MouseButtons.Left)
    {
        currentButton.Text = "";
        if (currentButton.BackColor == Color.Black)
        {
            currentButton.BackColor = defaultColor;
            --currentColored;
        }
        else
        {
            currentButton.BackColor = Color.Black;
            ++currentColored;
        }
    }
    else if (e.Button == MouseButtons.Right)
    {
        if (currentButton.BackColor == Color.Black)
        {
            currentButton.BackColor = defaultColor;
            --currentColored;
        }
        if (currentButton.Text == "X")
        {
            currentButton.Text = "";
        }
        else
        {
            currentButton.Text = "X";
        }
    }
    else
    {
        if (currentButton.BackColor == Color.Black)
        {
            currentButton.BackColor = defaultColor;
            --currentColored;
        }
        currentButton.Text = "";
    }
    if (started && currentColored == coloredSum)
    {
        if (checkNonogram())
        {
            MessageBox.Show("Bravo!", "Točno riješen nonogram");
        }
    }
}
```

Slika 4.8. Metoda NewPanelButton_MouseDown

Metoda služi za gumbe koji predstavljaju mrežu nonograma. Dodaje se svakom napravljenom gumbu i služi za promjenu izgleda gumba ovisno o pritisnutoj tipci na mišu. Također, metoda provjerava je li nonogram ispravno riješen nakon svakog klika na gumb iz mreže nonograma.

4.3. Generiranje nonograma

Generiranje nonograma se vrši na jedan od tri načina:

- Generiranje prethodno definiranog nonograma “FERIT”
- Generiranje nonograma unoseći sve linije uz mrežu
- Generiranje nonograma iscrtavajući željenu sliku

```
private void createNonogram()
{
    int Top = 2, Left = 2;
    for (int i = 0; i < nonogramHeight; i++)
    {
        for (int j = 0; j < nonogramWidth; j++)
        {
            buttons[i, j] = new Button();
            buttons[i, j].Top = Top;
            buttons[i, j].Left = Left;
            buttons[i, j].Height = 25;
            buttons[i, j].Width = 25;
            buttons[i, j].BackColor = defaultColor;
            buttons[i, j].MouseDown += new System.Windows.Forms.MouseEventHandler(this.NewPanelButton_MouseDown);
            buttons[i, j].Font = new Font("Arial", 12, FontStyle.Bold);
            buttons[i, j].Margin = new Padding(0, 0, 0, 0);
            panelButtons.Controls.Add(buttons[i, j]);
            Left += 27;
        }
        Top += 27;
        Left = 2;
    }
}
```

Slika 4.9. Metoda createNonogram

Metoda createNonogram dodaje mrežu gumba u panel “panelButtons” i dodaje im potrebne kontrole i vrijednosti.

```
private void addTopNumbers()
{
    int Top = panelTop.Height - 16, Left = 5;
    for (int i = 0; i < nonogramWidth; i++)
    {
        int n = topNumbers[i].Count, j = 0;
        int[] temp = new int[n];
        foreach (int c in topNumbers[i])
        {
            temp[n - j - 1] = c;
            ++j;
        }
        for (j = 0; j < n; ++j)
        {
            Label newLabel = new Label();
            newLabel.Text = temp[j].ToString();
            newLabel.Top = Top;
            newLabel.Left = Left;
            newLabel.Height = 20;
            newLabel.Width = 20;
            newLabel.TextAlign = ContentAlignment.MiddleCenter;
            panelTop.Controls.Add(newLabel);
            Top -= 22;
        }
        Top = panelTop.Height - 16;
        Left += 27;
    }
}
```

Slika 4.10. Metoda addTopNumbers

Metoda addTopNumbers dodaje oznake za brojeve iznad mreže nonograma u panel “panelTop”.

```
private void addSideNumbers()
{
    int Left = panelSide.Width - 16, Top = 5;
    for (int i = 0; i < nonogramHeight; i++)
    {
        int n = sideNumbers[i].Count, j = 0;
        int[] temp = new int[n];
        foreach (int c in sideNumbers[i])
        {
            temp[n - j - 1] = c;
            ++j;
        }
        for (j = 0; j < n; ++j)
        {
            Label newLabel = new Label();
            newLabel.Text = temp[j].ToString();
            newLabel.Top = Top;
            newLabel.Left = Left;
            newLabel.Height = 20;
            newLabel.Width = 20;
            newLabel.TextAlign = ContentAlignment.MiddleCenter;
            panelSide.Controls.Add(newLabel);
            Left -= 22;
        }
        Left = panelSide.Width - 16;
        Top += 27;
    }
}
```

Slika 4.11. Metoda addSideNumbers

Metoda addSideNumbers dodaje oznake za brojeve pored mreže nonograma u panel “panelSide”.

4.3.1. Generiranje prethodno definiranog nonograma “FERIT”

Prva opcija generiranja nonograma je generiranje prethodno definiranog nonograma. Taj nonogram je riješen u 2. poglavlju. Proces se pokreće gumbom “FERIT”.

```
private void buttonFERIT_Click(object sender, EventArgs e)
{
    nonogramHeight = 18;
    nonogramWidth = 16;
    buttons = new Button[nonogramHeight, nonogramWidth];
    generateFeritNonogram();
    createNonogram();
    addTopNumbers();
    addSideNumbers();
    comboBoxHeigth.Enabled = false;
    comboBoxWidth.Enabled = false;
    buttonStart.Enabled = false;
    buttonGrid.Enabled = false;
    buttonFERIT.Enabled = false;
    buttonSolve.Enabled = true;
}
```

Slika 4.12. Metoda buttonFERIT_Click

Metoda se poziva na pritisak gumba “FERIT”. Postavlja visinu i širinu nonograma, alocira memoriju za bolje gumba “buttons”, zatim poziva metodu “generateFeritNonogram” koja generira prethodno definirati nonogram te poziva metode “createNonogram”, “addTopNumbers” i “addSideNumbers” koje su gore objašnjene. Također onemogućava daljnje postavljanje visine i

širine, te ponovno generiranje nonograma. Otključava gumb “Solve” koji se koristi za rješavanje nonograma.

```
private void generateFeritNonogram()
{
    coloredSum = 0;
    int[][] matrixTop = new int[16][];
    matrixTop[0] = new int[] { 18 };
    matrixTop[1] = new int[] { 5, 5 };
    matrixTop[2] = new int[] { 4, 8, 4 };
    matrixTop[3] = new int[] { 3, 10, 3 };
    matrixTop[4] = new int[] { 3, 10, 3 };
    matrixTop[5] = new int[] { 2, 4, 4, 2 };
    matrixTop[6] = new int[] { 2, 3, 5, 2 };
    matrixTop[7] = new int[] { 1, 3, 7, 1 };
    matrixTop[8] = new int[] { 1, 7, 3, 1 };
    matrixTop[9] = new int[] { 2, 5, 3, 2 };
    matrixTop[10] = new int[] { 2, 4, 4, 2 };
    matrixTop[11] = new int[] { 3, 10, 3 };
    matrixTop[12] = new int[] { 3, 10, 3 };
    matrixTop[13] = new int[] { 4, 8, 4 };
    matrixTop[14] = new int[] { 5, 5 };
    matrixTop[15] = new int[] { 18 };
    int[][] matrixSide = new int[18][];
    matrixSide[0] = new int[] { 16 };
    matrixSide[1] = new int[] { 7, 7 };
    matrixSide[2] = new int[] { 5, 2, 5 };
    matrixSide[3] = new int[] { 3, 6, 3 };
    matrixSide[4] = new int[] { 2, 10, 2 };
    matrixSide[5] = new int[] { 1, 5, 6, 1 };
    matrixSide[6] = new int[] { 1, 4, 6, 1 };
    matrixSide[7] = new int[] { 1, 3, 2, 3, 1 };
    matrixSide[8] = new int[] { 1, 3, 1, 3, 1 };
    matrixSide[9] = new int[] { 1, 3, 1, 3, 1 };
    matrixSide[10] = new int[] { 1, 3, 2, 3, 1 };
    matrixSide[11] = new int[] { 1, 6, 4, 1 };
    matrixSide[12] = new int[] { 1, 6, 5, 1 };
    matrixSide[13] = new int[] { 2, 10, 2 };
    matrixSide[14] = new int[] { 3, 6, 3 };
    matrixSide[15] = new int[] { 5, 2, 5 };
    matrixSide[16] = new int[] { 7, 7 };
    matrixSide[17] = new int[] { 16 };

    for (int i = 0; i < nonogramHeight; i++)
    {
        sideNumbers.Add(new List<int>(matrixSide[i]));
        coloredSum += matrixSide[i].Sum();
    }
    for (int i = 0; i < nonogramWidth; i++)
    {
        topNumbers.Add(new List<int>(matrixTop[i]));
    }
}
```

Slika 4.13. Metoda generateFeritNonogram

Metoda generira nonogram s prethodno zadanim brojevima uz mrežu te dodaje brojeve u povezane popise “sideNumbers” i “topNumbers” koje poslije metode “addSideNumbers” i “addTopNumbers” koriste za generiranje brojeva uz mrežu. Također se isti povezani popisi koriste za pronalazak rješenja i provjeravanje istog.

4.3.2. Generiranje nonograma unošenjem svih linija uz mrežu

Druga opcija generiranja nonograma je generiranjem unošenjem svake od linija uz mrežu. Prvo se unose redovi od gornjeg prema dolje, zatim stupci od lijevog prema desno. Za ovu funkcionalnost aplikacije koristi se spomenuta pomoćna forma. Proces se pokreće pritiskom na gumb “Grid”.

```
private void buttonGrid_Click(object sender, EventArgs e)
{
    if (comboBoxHeigth.Text == "")
    {
        nonogramHeight = 2;
    }
    else
    {
        nonogramHeight = int.Parse(comboBoxHeigth.Text);
    }
    if (comboBoxWidth.Text == "")
    {
        nonogramWidth = 2;
    }
    else
    {
        nonogramWidth = int.Parse(comboBoxWidth.Text);
    }
    int checkViability;
    coloredSum = 0;
    for (int i = 0; i < nonogramHeight; ++i)
    {
        Form2 dialog = new Form2();
        string temp = "";
        if (dialog.ShowDialog(this) == DialogResult.OK)
        {
            temp += dialog.textBox1.Text;
        }
        List<int> tempList = new List<int>();
        if (temp == "")
        {
            tempList.Add(0);
        }
        else
        {
            string[] t = temp.Split(' ');
            foreach (string s in t)
            {
                tempList.Add(int.Parse(s));
                coloredSum += int.Parse(s);
            }
        }
        sideNumbers.Add(tempList);
        dialog.Dispose();
    }
    checkViability = coloredSum;
    for (int i = 0; i < nonogramWidth; ++i)
    {
        Form2 dialog = new Form2();
        string temp = "";
        if (dialog.ShowDialog(this) == DialogResult.OK)
        {
            temp += dialog.textBox1.Text;
        }
        List<int> tempList = new List<int>();
        if (temp == "")
        {
            tempList.Add(0);
        }
        else
        {
            string[] t = temp.Split(' ');
            foreach (string s in t)
            {
                tempList.Add(int.Parse(s));
                checkViability -= int.Parse(s);
            }
        }
        topNumbers.Add(tempList);
        dialog.Dispose();
    }
    if (checkViability != 0)
    {
        MessageBox.Show("Nije jednak zbroj brojeva sa strane i gore.", "Greška!", MessageBoxButtons.OK, MessageBoxIcon.Error);
        buttonReset.PerformClick();
    }
    else
    {
        buttons = new Button[nonogramHeight, nonogramWidth];
        currentColored = 0;
        started = true;
        createNonogram();
        addSideNumbers();
        addTopNumbers();
        comboBoxHeigth.Enabled = false;
        comboBoxWidth.Enabled = false;
        buttonFERIT.Enabled = false;
        buttonGrid.Enabled = false;
        buttonStart.Enabled = false;
        buttonSolve.Enabled = true;
    }
}
```

Slika 4.14. Metoda buttonGrid_Click

Metoda buttonGrid_Click prvo provjerava jesu li unesene vrijednosti za visinu i širinu nonograma, te, ukoliko nisu, postavlja na 2, inače na odabranu vrijednost. Zatim pokreće pomoćnu aplikaciju pomoću koje se unose svi reci, zatim svi stupci. Metoda zatim provjerava jesu li ispravno unesene vrijednosti, mora biti jednak zbroj brojeva pored mreže zbroju brojeva iznad mreže. Ukoliko nisu, pokazuje se pogreška, u suprotnom, generira se nonogram kao i kod prvog načina.

```
private void buttonNext_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.OK;
}
1 reference
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        DialogResult = DialogResult.OK;
    }
}
```

Slika 4.15. Metode pomoćne aplikacije

Metode pomoćne aplikacije omogućavaju komunikaciju s glavnom aplikacijom. Također, omogućuju dva načina prelaza na sljedeću liniju, pritiskom na gumb “Next” i pritiskom tipke “Enter” na tipkovnici.

4.3.3. Generiranje nonograma iscrtavanjem željene slike

Posljednji način generiranja nonograma je generiranje icrtavanjem željene slike. Proces se pokreće pritiskom na gumb “Picture”.

```
private void buttonStart_Click(object sender, EventArgs e)
{
    if (!generated)
    {
        if (comboBoxHeigth.Text == "")
        {
            nonogramHeight = 2;
        }
        else
        {
            nonogramHeight = int.Parse(comboBoxHeigth.Text);
        }
        if (comboBoxWidth.Text == "")
        {
            nonogramWidth = 2;
        }
        else
        {
            nonogramWidth = int.Parse(comboBoxWidth.Text);
        }
        comboBoxHeigth.Enabled = false;
        comboBoxWidth.Enabled = false;
        buttonGrid.Enabled = false;
        buttonFERIT.Enabled = false;
        buttons = new Button[nonogramHeight, nonogramWidth];
        createNonogram();
        buttonStart.Text = "Generate";
        generated = true;
    }
```

Slika 4.16. Prvi dio metode buttonStart_Click

Pri prvom pritisku na gumb “Picture”, poziva se prvi dio metode buttonStart_Click, koji postavlja veličine nonograma na sličan način kao prethodna metoda generiranja nonograma, zatim generira polje gumba i mijenja oznaku na istom gumbu u “Generate” i postavlja vrijednost varijable “generated” u “true”. Nakon generiranja praznog polja gumba, može se nacrtati željena slika iz koje će se, ponovnim pritiskom na istu tipku, generirati željeni nonogram.

```
    else
    {
        int counter = 0;
        foreach (Button f in panelButtons.Controls)
        {
            int i, j;
            i = (f.Top - 2) / 27;
            j = (f.Left - 2) / 27;
            buttons[i, j] = f;
            ++counter;
            if (counter == nonogramHeight * nonogramWidth)
            {
                break;
            }
        }
        generateNumbers();
        started = true;
        for (int i = 0; i < nonogramHeight; ++i)
        {
            for (int j = 0; j < nonogramWidth; ++j)
            {
                buttons[i, j].Text = "";
                buttons[i, j].BackColor = defaultColor;
            }
        }
        currentColored = 0;
        buttonStart.Enabled = false;
        buttonSolve.Enabled = true;
        panelButtons.Controls.Clear();
        createNonogram();
    }
}
```

Slika 4.17. Drugi dio metode buttonStart_Click

Ponovnim pritiskom iste tipke, koja sada ima oznaku “Generate”, poziva se drugi dio metode buttonStart_Click. Metoda zatim generira nonogram pomoću funkcije “generateNonogram” i čisti sve gumbe mreže nonograma.

```
private void generateNumbers()
{
    for (int i = 0; i < nonogramHeight; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramWidth; j++)
        {
            if (buttons[i, j].BackColor == Color.Black)
            {
                ++counter;
            }
            else if (counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramWidth - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        coloredSum += temp.Sum();
        sideNumbers.Add(temp);
    }
    for (int i = 0; i < nonogramWidth; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramHeight; j++)
        {
            if (buttons[j, i].BackColor == Color.Black)
            {
                ++counter;
            }
            else if (counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramHeight - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        topNumbers.Add(temp);
    }

    addTopNumbers();
    addSideNumbers();
}
```

Slika 4.18. Metoda generateNumbers

Metoda “generateNumbers” generira brojeve uz mrežu nonograma koristeći trenutne gumbe mreže nonograma. Za svaki dani redak, prebrojava se broj uzastopnih obojanih polja i zapisuje u varijablu “counter”. Kad se dođe do prekida, broj prethodno uzastopnih polja se zapisuje u vezani popis “temp”. Proces se ponavlja dok se ne dođe do kraja retka. Ukoliko dani redak nema obojanih polja, dodaje se vrijednost 0. Na poslijetku se vezana lista “temp” dodaje na kraj vezane liste vezanih lista “sideNumbers”. Sličan proces se ponavlja za sve stupce. Na poslijetku se dobiveni brojevi dodaju iznad i pored mreže nonograma.

U svakoj od navedenih metoda za generiranje nonograma, provjerava se broj obojanih polja u konačnom rješenju i sprema se u varijablu “coloredSum”.

4.4. Provjera rješenja nonograma

Postoje dvije metode za provjeru nonograma. Jedna se koristi za provjeru rješenja nonograma nakon pritiska gumba u mreži nonograma. Druga se koristi za provjeru rješenja nonograma za vrijeme izvođenja algoritma rješavnja zadanog nonograma.

```
private bool checkNonogram()
{
    for (int i = 0; i < nonogramHeight; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramWidth; j++)
        {
            if (buttons[i, j].BackColor == Color.Black)
            {
                ++counter;
            }
            else if (counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramWidth - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        if (!compareLists(temp, sideNumbers[i]))
        {
            return false;
        }
    }
    for (int i = 0; i < nonogramWidth; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramHeight; j++)
        {
            if (buttons[j, i].BackColor == Color.Black)
            {
                ++counter;
            }
            else if (counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramHeight - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        if (!compareLists(temp, topNumbers[i]))
        {
            return false;
        }
    }
    return true;
}
```

Slika 4.19. Metoda checkNonogram

```
private bool checkSolution()
{
    for (int i = 0; i < nonogramHeight; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramWidth; j++)
        {
            if (solutions[solutions.Count - 1][i, j] == 1)
            {
                ++counter;
            }
            else if (solutions[solutions.Count - 1][i, j] == 0 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramWidth - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        if (!compareLists(temp, sideNumbers[i]))
        {
            return false;
        }
    }
    for (int i = 0; i < nonogramWidth; i++)
    {
        List<int> temp = new List<int>();
        int counter = 0;
        for (int j = 0; j < nonogramHeight; j++)
        {
            if (solutions[solutions.Count - 1][j, i] == 1)
            {
                ++counter;
            }
            else if (solutions[solutions.Count - 1][j, i] == 0 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
            if (j == nonogramHeight - 1 && counter != 0)
            {
                temp.Add(counter);
                counter = 0;
            }
        }
        if (temp.Count == 0)
        {
            temp.Add(0);
        }
        if (!compareLists(temp, topNumbers[i]))
        {
            return false;
        }
    }
    return true;
}
```

Slika 4.20. Metoda checkSolution

Metode “checkNonogram” i “checkSolution” provjeravaju rješenja nonograma. “checkNonogram” provjerava ispravnost nonograma prema 2D polju gumba “buttons” te vraća “true” ukoliko je nonogram ispravno riješen, a “false” u suprotnom. “checkSolution” provjerava ispravnost zadnjeg predloženog rješenja u vezanom popisu 2D polja cijelih brojeva “solutions”. Također vraća “true” ukoliko je nonogram ispravan, a “false” u suprotnom. U obje metode se koristi sličan algoritam kao u metodi generateNumbers, ali se, nakon što se generira dani redak (stupac), provjerava je li redak (stupac) ispravan, te vraća “false” ukoliko nije. Ukoliko nijedan redak (stupac) nije neispravan, metode vraćaju “true”.

5. ALGORITAM RJEŠAVANJA NONOGRAMA

Algoritam prolazi kroz sljedeće korake dok ne dođe do rješenja:

1. Generiranje svih mogućih ispravnih rješenja za dani redak (stupac)
2. Zaključivanje o rješenju retka (stupca) na temelju generiranih mogućnosti
3. Ponavljanje prva dva koraka za sve retke (stupce)

Ukoliko algoritam na ovaj način ne dođe do rješenja, odnosno dođe do zastoja, prolazi kroz sljedeće korake:

4. Pretpostavljanje da je jedno od neodređenih polja obojano
5. Ponavljanje prva tri koraka iz prvog dijela algoritma
6. Zaključivanje o ispravnosti pretpostavke iz četvrtog koraka.
7. Ukoliko je pretpostavka neispravna, algoritam se vraća na prvi korak sa zaključkom iz šestog koraka

5.1. Rješavanje pojedinog retka (stupca)

Prva dva koraka algoritma odnose se na rješavanje svakog zasebnog retka (stupca). Ovaj proces se provodi u nekoliko koraka:

- Generira se kombinacija za razmake
- Generira se redak (stupac) za danu kombinaciju za razmake
- Provjerava se valjanost generiranog retka (stupca) u usporedbi s trenutnim djelomičnim rješenjem retka (stupca)
- Dodaju se vrijednosti valjanog generiranog retka (stupca) privermenom polju cijelih brojeva
- Proces se ponavlja za sve kombinacije za razmake
- Na temelju privremenog polja i broju valjanih redaka (stupaca), zaključuje se o mogućim promjenama na trenutnom rješenju retka (stupca)

5.1.1. Generiranje valjanog retka

Prvo se generira kombinacija za razmake. Pogledajmo redak koji ima brojeve uz mrežu “3 4 4” i širina nonograma je 15 polja. Znamo da između svakog segmenta obojanih polja mora biti jedno prazno polje. Stoga ovaj redak ostavlja 2 prazna polja koja se mogu razmjestiti ispred prvog

segmenta, između prva dva segmenta, između drugog i trećeg segmenta, te nakon trećeg segmenta u bilo kojoj kombinaciji. Uzmu li se slobodna prazna polja kao jabuke, a segmenti s pripadajućim praznim poljima (svako sigurno prazno polje pripada prethodnom segmentu) kao štapovi, imamo jednostavan problem razmještanja štapova i jabuka. To rezultira s $\binom{5}{3} = 10$ kombinacija. Ovaj proces obavlja metoda "combinations". [6, str. 24]

```
private void combinations(int start, int k, int s, int line, bool row)
{
    if (k == 0)
    {
        if (row)
        {
            int[] temp = generateLine(combination, sideNumbers[line], nonogramWidth);
            if (checkLineViability(temp, line, nonogramWidth, true))
            {
                for (int i = 0; i < nonogramWidth; ++i)
                {
                    tempLine[i] += temp[i];
                }
                ++correctLines;
            }
        }
        else
        {
            int[] temp = generateLine(combination, topNumbers[line], nonogramHeight);
            if (checkLineViability(temp, line, nonogramHeight, false))
            {
                for (int i = 0; i < nonogramHeight; ++i)
                {
                    tempLine[i] += temp[i];
                }
                ++correctLines;
            }
        }
        return;
    }
    for (int i = start; i <= s - k; ++i)
    {
        combination.Add(i + 1);
        combinations(i + 1, k - 1, s, line, row);
        combination.RemoveAt(combination.Count - 1);
    }
}
```

Slika 5.1. Metoda combinations

Argument "start" označava prvi promatrani element, argument "k" označava broj preostalih elemenata do kraja, argument "s" označava ukupnu veličinu danog polja", argument "line" označava o kojoj liniji (retku/stupcu) se radi, te argument "row" označava radi li se o retku ili stupcu. Metoda combinations se rekurzivno poziva dok joj se ne preda "k" jednak nuli. Za jednosatvni primjer 4 elementa i 2 "mjesta", $\binom{4}{2}$, algoritam će do prvog zastoja, gdje se izvode drugi i treći korak procesa rješavanja retka (stupca), doći na kombinaciji (1, 2), zatim na kombinacijama (1, 3), (1, 4), (2, 3), (2, 4), te (3, 4). [7, str. 207]

Sljedeći korak je generirati mogući redak iz dane kombinacije za razmake. Ovaj proces obavlja metoda “generateLine”.

```
private int[] generateLine(List<int> combination, List<int> line, int n)
{
    int[] temp = Enumerable.Repeat(1, n).ToArray();
    List<int> tempLine = new List<int>(line);
    List<int> tempCombination = new List<int>(combination);
    int current = 0;
    for (int i = tempCombination.Count - 1; i > 0; --i)
    {
        tempCombination[i] -= tempCombination[i - 1];
    }
    tempCombination[0]--;
    foreach (int i in tempCombination)
    {
        for (int j = 0; j < i; ++j)
        {
            temp[current++] = 0;
        }
        current += tempLine[0];
        tempLine.RemoveAt(0);
    }
    while (current != n)
    {
        temp[current++] = 0;
    }
    return temp;
}
```

Slika 5.2. Metoda generateLine

Metoda generateLine prima tri parametra, kombinaciju za razmake, brojeve uz mrežu za taj redak (stupac), te veličinu retka (stupca). Metoda smanjuje prvi broj iz kombinacije kako bi se dobio broj praznih polja koja idu ispred prvog segmenta obojanih polja. Kako bi se dobili traženi razmaci između segmenata, od svakog broja iz kombinacije oduzima se prethodni. Uzmimo za primjer redak “3 4 4 “ u nonogramu širine 15 polja, te je metodi predana kombinacija (1, 3, 4). Na početak ne treba doći nijedan razmak (1 – 1 = 0), zatim 2 razmaka između prvog i drugog segmenta (3 – 1 = 2), te 1 razmak između drugog i trećeg segmenta (4 – 3 = 1). Na kraj dolazi ostatak neiskorištenih razmaka (15 – 0 – 3 – 2 – 4 – 1 – 4 = 1). To je prvi korak algoritma metode. Drugi korak je samo generiranje linije iz zadanih razmaka i brojeva uz mrežu. Postavljaju se sve vrijednosti linije na 1 (obojano) te se naknadno dodaju razmaci. Na ovom primjeru, prvo se dodaje 0 razmaka, zatim se preskoči 3 polja, dodaju se 2 razmaka, preskaču se 4 polja, dodaje se jedan razmak, preskaču se 4 polja i, na poslijetku, ostaje još jedan razmak za dodati. Na kraju polje “temp” izgleda ovako: [1 1 1 0 0 1 1 1 1 0 1 1 1 1 0].

Treći korak je provjera ispravnosti generiranog retka (stupca) u usporedbi s trenutnim djelomičnim rješenjem danog retka. Ovaj proces obavlja metoda “checkLineViability”.

```
private bool checkLineViability(int[] array, int line, int n, bool row)
{
    if (row)
    {
        int i = line;
        for (int j = 0; j < n; ++j)
        {
            if (!(solutions[solutions.Count - 1][i, j] == 2 || array[j] == solutions[solutions.Count - 1][i, j]))
            {
                return false;
            }
        }
    }
    else
    {
        int j = line;
        for (int i = 0; i < n; ++i)
        {
            if (!(solutions[solutions.Count - 1][i, j] == 2 || array[i] == solutions[solutions.Count - 1][i, j]))
            {
                return false;
            }
        }
    }
    return true;
}
```

Slika 5.3. Metoda checkLineViability

Metoda provjerava postoji li ikoje polje, gdje trenutno djelomično rješenje polja nije neodlučeno (nema vrijednost 2), u generiranom retku (stupcu) da je različito od trenutnog djelomičnog rješenja. Vraća “false” ukoliko naiđe na takvo polje, a “true” u suprotnom.

Na poslijetku još treba dodati ispravna rješenja privremenom polju cijelih brojeva “tempLine” i povećati broj ispravnih linija. To se obavlja u metodi “combinations”.

```
if (checkLineViability(temp, line, nonogramWidth, true))
{
    for (int i = 0; i < nonogramWidth; ++i)
    {
        tempLine[i] += temp[i];
    }
    ++correctLines;
}
```

Sliika 5.4. Dodavanje ispravnih rješenja redaka

5.1.2. Izmjena trenutnog rješenja retka (stupca)

Ukoliko se u nekom polju za svako ispravno rješenje pojavljuje 1, na tom mjestu mora biti 1 u trenutnom rješenju retka, slično za pojavljivanje 0. Ovaj proces obavlja funkcija “solveLine”.

```
private void solveLine(int line, bool row)
{
    if (row)
    {
        int i = line;
        for (int j = 0; j < nonogramWidth; ++j)
        {
            if (tempLine[j] == 0 && solutions[solutions.Count - 1][i, j] != 0)
            {
                solutions[solutions.Count - 1][i, j] = 0;
                changed = true;
            }
            else if (tempLine[j] == correctLines && solutions[solutions.Count - 1][i, j] != 1)
            {
                solutions[solutions.Count - 1][i, j] = 1;
                changed = true;
            }
        }
    }
    else
    {
        int j = line;
        for (int i = 0; i < nonogramHeight; ++i)
        {
            if (tempLine[i] == 0 && solutions[solutions.Count - 1][i, j] != 0)
            {
                solutions[solutions.Count - 1][i, j] = 0;
                changed = true;
            }
            else if (tempLine[i] == correctLines && solutions[solutions.Count - 1][i, j] != 1)
            {
                solutions[solutions.Count - 1][i, j] = 1;
                changed = true;
            }
        }
    }

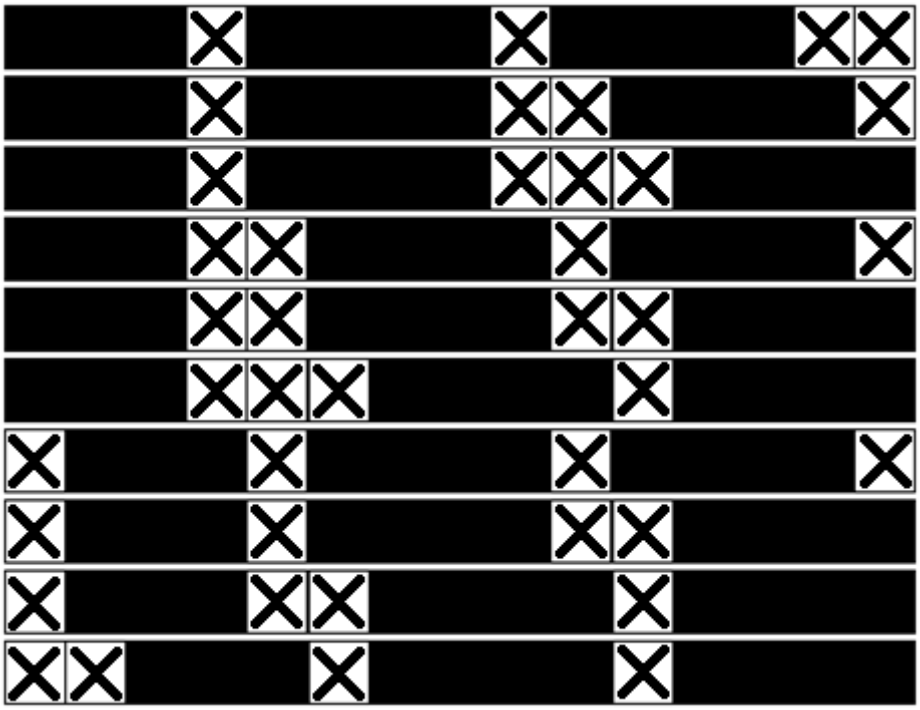
}
```

Slika 5.5. Metoda solveLine

Metodi se predaje redni broj retka (stupca) te je li u pitanju redak ili stupac. Ukoliko je broj na danom mjestu u polju “tempLine” jednak broju valjanih redaka (stupaca), polje koje odgovara tom mjestu, treba biti obojano, odnosno postaje 1, slično ako je broj na danom mjestu u polju “tempLine” jedan 0, polje koje odgovara tom mjestu, treba biti prazno, odnosno postaje 0.

5.1.3. Primjer

Za zadani primjer retka “3 4 4” nonograma širine 15, moguća rješenja retka su kako je prikazano na slici 5.6.



Slika 5.6. Sva moguća rješenja retka “3 4 4” sa širinom nonograma 15

Iz slike 5.6. se lagano vidi koja polja su definitivno obojana, a koja su definitivno prazna, što je prikazano na slici 5.7.



Slika 5.7. Konačno djelomično rješenje retka “3 4 4” sa širinom nonograma 15

5.2. Ukupno rješenje nonograma

Proces opisan u poglavlju 5.1. se ponavlja za svaki redak (stupac). Ukoliko se dobije ispravno rješenje, nonogram je rješen. U suportnom, pretpostavlja se za jedno polje da je obojano, odnosno ima vrijednsot 1, zatim se ponavlja prvi dio algoritma dok se ne riješi nonogram ili se ne dođe do kontradikcije. Ukoliko se dođe do kontradikcije, može se zaključiti da je ono polje za koje je pretpostavljeno da je obojano, prazno, odnosno 0. Nakon ove ispravke, ponavlja se algoritam dok se ne dođe do točnog rješenja. Sva djelomična rješenja sa zastojima prije pretpostavki se spremaju u vezani popis "solutions", a koordinate polja, za koja je isprobana jedinica, u vezani popis "tried". Cijeli ovaj proces obavlja metoda "buttonSolve_Click".

Na slici 5.8. je prikazan dio metode "buttonSolve_Click" zadužen za izvrešenje prva tri koraka, te šesti i sedmi korak algoritma rješavanja nonograma. Postavlja zastavice "changed" i "mistake" na "false", koje govore je li došlo do zastoja, te je li došlo do kontradikcije. Zatim se izvršava, za svaki redak, proces opisan u poglavlju 5.1. Unutar metode "solveLine" se zastavica "changed" mijenja u "true" ukoliko dođe do bilo kakve promjene trenutnog rješenja. Ukoliko se, nakon izvođenja metode "combinations" ispostavi da dani redak nema nijedno rješenje, došlo je do kontradikcije, te se pogreška ispravlja, zastavica "mistake" se postavlja u "true" da bi se prva dva koraka nastavila izvršavati na sada ispravljenom trenutnom rješenju. Sličan proces se ponavlja za sve stupce nonograma. Ukoliko je došlo do napretka, "changed" je jednak "true", ili je došlo do pogreške, "mistake" je jednak "true". Prva dva koraka se nastavljaju izvršavati sve dok se ne dođe do zastoja, zastoj se dogodi i kada se dođe do konačnog rješenja.

Na slici 5.9. je prikazan dio metode "buttonSolve_Click" zadužen za početak samog alogirtma prije prvog koraka, te za izvršavanje četvrtog koraka algoritma rješavanja nonograma. Metoda također pokreće štopericu koja poslije pokaže vrijeme potrebno za izvršavanje algoritma u milisekundama. Ukoliko je prvo izvođenje algoritma, pravi se 2D polje "solutions[0]" koje predstavlja trenutno rješenje nonograma te se postavljaju sve vrijednosti istog rješenja na 2, što označava neodlučeno polje. Ukoliko nije prvo izvođenje ovog dijela algoritma, odnosno, došlo je do zastoja u nastavku algoritma, pravi se novo 2D polje koje predstavlja novo trenutno rješenje nonograma, te se postavlja na kraj vezanog popisa "solutions". U ovo novo trenutno rješenje kopiraju se vrijednosti iz prethodnog trenutnog rješenja na kojem je došlo do zastoja. Pronalazi se prvi redak koji nije u potpunosti riješen, te se pronalazi polje, koje već nije obojano, za koje je najveća vjerojatnost da bude obojano. Vrijednost tog polja se postavlja na "1", odnosno obojano, te se, ukoliko je potrebno, kopira ostatak prethodnog trenutnog rješenja u sadašnje. Nakon toga, algoritam nastavlja s izvođenjem dijela metode prikazaog na slici 5.8.

```
do
{
    changed = false;
    mistake = false;
    for (int i = 0; i < nonogramHeight; ++i)
    {
        tempLine = new int[nonogramWidth];
        correctLines = 0;
        int s = nonogramWidth;
        foreach (int a in sideNumbers[i])
        {
            s -= a;
        }
        ++s;
        combinations(0, sideNumbers[i].Count, s, i, true);
        if (correctLines != 0)
        {
            solveLine(i, true);
        }
        else
        {
            if (solutions.Count != 1)
            {
                solutions[solutions.Count - 2][tried[tried.Count - 1].i, tried[tried.Count - 1].j] = 0;
                solutions.RemoveAt(solutions.Count - 1);
                tried.RemoveAt(tried.Count - 1);
            }
            else
            {
                finalMistake = true;
            }
            mistake = true;
            break;
        }
    }
    if (finalMistake)
    {
        break;
    }
    for (int j = 0; j < nonogramWidth; ++j)
    {
        tempLine = new int[nonogramHeight];
        correctLines = 0;
        int s = nonogramHeight;
        foreach (int a in topNumbers[j])
        {
            s -= a;
        }
        ++s;
        combinations(0, topNumbers[j].Count, s, j, false);
        if (correctLines != 0)
        {
            solveLine(j, false);
        }
        else
        {
            if (solutions.Count != 1)
            {
                solutions[solutions.Count - 2][tried[tried.Count - 1].i, tried[tried.Count - 1].j] = 0;
                solutions.RemoveAt(solutions.Count - 1);
                tried.RemoveAt(tried.Count - 1);
            }
            else
            {
                finalMistake = true;
            }
            mistake = true;
            break;
        }

    }
    if (finalMistake)
    {
        break;
    }
} while (changed || mistake);
```

Slika 5.8. Dio metode buttonSolve_Click zadužen za prva tri koraka algoritma

```
solutions.Add(new int[nonogramHeight, nonogramWidth]);
if (solutions.Count == 1)
{
    for (int i = 0; i < nonogramHeight; ++i)
    {
        for (int j = 0; j < nonogramWidth; ++j)
        {
            solutions[solutions.Count - 1][i, j] = 2;
        }
    }
}
else
{
    bool guessed = false;
    for (int i = 0; i < nonogramHeight; ++i)
    {
        int unsolved = 0;
        for (int j = 0; j < nonogramWidth; ++j)
        {
            solutions[solutions.Count - 1][i, j] = solutions[solutions.Count - 2][i, j];
            if (solutions[solutions.Count - 1][i, j] == 2)
            {
                ++unsolved;
            }
        }

        if (unsolved != 0 && !guessed)
        {
            correctLines = 0;
            tempLine = new int[nonogramWidth];
            int s = nonogramWidth;
            foreach (int a in sideNumbers[i])
            {
                s -= a;
            }
            ++s;
            combinations(0, sideNumbers[i].Count, s, i, true);
            int max = 0;
            for (int j = 1; j < nonogramWidth; ++j)
            {
                if (tempLine[j] > tempLine[max] && tempLine[j] != correctLines)
                {
                    max = j;
                }
            }
            tried.Add(new tuple(i, max));
            solutions[solutions.Count - 1][i, max] = 1;
            guessed = true;
        }
    }
}
```

Slika 5.9. Dio metode buttonSolve_Click zadužen za četvri korak algoritma

Nakon što algoritam dođe do rješenja, trenutno rješenje se sprema u varijablu “finalSolution”. Nakon pronalska rješenja, algoritam ne mora prestati s izvršavanjem. Algoritam se prekida ukoliko zaključi da je to jedino rješenje, pronađe sva druga rješenja ili pronađe 1000 različitih rješenja, kao što je prikazano na slici 5.10. U nastavku se konačno rješenje prema polju “finalSolution” iscrtava na grafičko sučelje, te iskače poruka o trajanju izvođenja algoritma, kao što je prikazano na slici 5.11. Ukoliko nonogram ima više od jednog rješenja, nonogram nije valjan, te se ne iscrtava nijedno rješenje.

```
    Label:
    do...

    if (checkSolution())
    {
        countSolutions++;
        if (countSolutions == 1)
        {
            finalSolution = solutions[solutions.Count - 1];
            timer.Restart();
        }
        if (solutions.Count == 1)
        {
            break;
        }
        else
        {
            solutions[solutions.Count - 2][tried[tried.Count - 1].i, tried[tried.Count - 1].j] = 0;
            solutions.RemoveAt(solutions.Count - 1);
            tried.RemoveAt(tried.Count - 1);
            goto Label;
        }
    }

    if (timer.Elapsed.Seconds >= 4)
    {
        timer.Stop();
        break;
    }
    if (finalMistake)
    {
        break;
    }
    if (countSolutions >= 1000)
    {
        break;
    }
} while (true);
```

Slika 5.10. Uvijeti za prekid algoritma za rješavanje nonograma

```
    sw.Stop();

    if (countSolutions == 1)
    {
        for (int i = 0; i < nonogramHeight; ++i)
        {
            for (int j = 0; j < nonogramWidth; ++j)
            {
                if (finalSolution[i, j] == 1)
                {
                    changeButtonState(i, j, true);
                }
                else if (finalSolution[i, j] == 0)
                {
                    changeButtonState(i, j, false);
                }
            }
        }
    }
    else
    {
        if (countSolutions >= 1000)
        {
            MessageBox.Show("Neispravan nonogram. Ukupno više od 1000 rješenja.", "Neispravan nonogram!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("Neispravan nonogram. Ukupno " + countSolutions + " različitih rješenja.", "Neispravan nonogram!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    MessageBox.Show("Vrijeme izvršavanja algoritma: " + sw.ElapsedMilliseconds.ToString() + " ms.", "Nonogram riješen!", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Slika 5.11. Završetak metode buttonSolve_Click

6. ZAKLJUČAK

Cilj ovog rada je razvitak algoritma za rješavanje nonograma, te njegova implementacija u C#-u. Kao rezultat je dobiven program koji ne samo rješava zadani nonogram, nego i provjerava je li zadani nonogram valjan. Također, program omogućava korisniku rješavanje nonograma te provjeru konačnog rješenja.

Pogodnosti ovog programa su vrlo brz pronalazak rješenja i kod teških nonograma te jednostavno i intuitivno korisničko sučelje. Moguća primjena programa je testiranje ispravnosti nonograma ukoliko korisnik želi kreirati vlastiti nonogram. Još jedna moguća primjena je pronalazak rješenja poznatog nonograma.

Ograničenja programa su ograničena veličina nonograma ovisna o veličini zaslona, potencijanlo vrlo dugo izvođenje programa za vrlo velike i teške nonograme, te nespretan unos nonograma. Uz bolje rukovanje memorijom, smanjilo bi se vrijeme izvođenja algoritma i poboljšalo korištenje rasursa računala. Unos pomoću unošenja brojeva uz mrežu bi se mogao popraviti omogućavanjem pregleda i promjene dotad unešenih linija ili unosom cijele mreže iz tekstualne datoteke.

LITERATURA

[1] James Dagelty, Origins of cross reference grid & picture grid puzzles, The Puzzle Museum, 2017, dostupno na: https://www.puzzlemuseum.com/griddler/gridhist.htm [29.6.2018.]

[2] Chugunnyy K.A. (KyberPrizrak), Learn to solve Japanese crosswords, dostpuno na: https://www.nonograms.org/instructions [29.6.2018.]

[3] Visual C++ Team Blog, dostupno na: http://www.pythonicquest.com/article/best-python-ide/ [12.9.2018.]

[4] Best Python IDE for Python Programming, dostupno na: http://www.pythonicquest.com/article/best-python-ide/ [12.9.2018.]

[5] Visual Studio 2017 version 15.7 Release Notes, dostupno na: https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes-v15.7 [12.9.2018.]

[6] B. Dakić, N. Elezović, Matematika 4, Element, Zagreb, 2006.

[7] M. Cvitković, Kombinatorika, Element, Zagreb, 1994.

SAŽETAK

Glavni zadatak rada bio je razviti algoritam za rješavanje nonograma te ga implementirati u C# aplikaciji. Aplikacija omogućuva unos nonograma, provjeru rješenja, rješavanje unesenog nonograma, te provjeru valjanosti nonograma. Unos nonograma omogućen je na dva načina, iscrtavanjem slike nonograma ili unošenjem brojeva uz mrežu. Algoritam rješavanja nonograma razvijen je po uzoru na način na koji čovjek rješava nonograme.

Ključne riječi: C#, nonogram, algoritam

ABSTRACT

Nonogram logic puzzle solution in C# programming language

The main task of this B.A. thesis was development of an algorithm for solving nonograms and the implementation of it in C# application. The application allows input of nonograms, solution checking, solving nonograms, and nonogram validation. The application allows input in form of a picture of a nonogram or numbers along the network. The algorithm for solving nonograms is based on the method which humans use to solve nonograms.

Keywords: C#, nonogram, algorithm

ŽIVOTOPIS

Matej Džijan rođen je 17. veljače 1997. godine u Vinkovcima. Osnovnu školu Ivana Gorana Kovačića završava u Vinkovcima 2011. godine. Iste godine upisuje Gimnaziju Matije Antuna Reljkovića, smjer prirodoslovno-matematički. Maturira 2015. godine, te iste godine upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku, današnji Fakultet elektrotehnike, računrastva i informacijskih tehnologija.

Uspjesi na natjecanjima: drugo mjesto na regionalnom i prvo mjesto na županijskom natjecanju iz matematike, te treće mjesto na županijskom natjecanju iz informatike – LOGO u petom razredu osnovne škole, druga mjesta na županijskim natjecanjima iz matematike u sedmom i osmom razredu osnovne škole, druga mjesta na županijskim natjecanjima iz matematike (A razina) u prvom i drugom razredu gimnazije, te prva mjesta na županijskim natjecanjima iz matematike (A razina) u trećem i četvrtom razredu gimnazije. Također se natjecao na Elektrijadi u Budvi 2017. godine u matematici, te u tehnološkoj areni na STEM Games-u 2018. godine, gdje je s ekipom osvojio 4. mjesto.

Potpis: _____