Documentație proiect IA

Rezolvarea unui puzzle Futoshiki utilizând algoritmul de backtracking cu forward-checking

Membri Echipa:

Pușcalău Robert-George – 1406A

Iovu Vali Cristian – 1406A

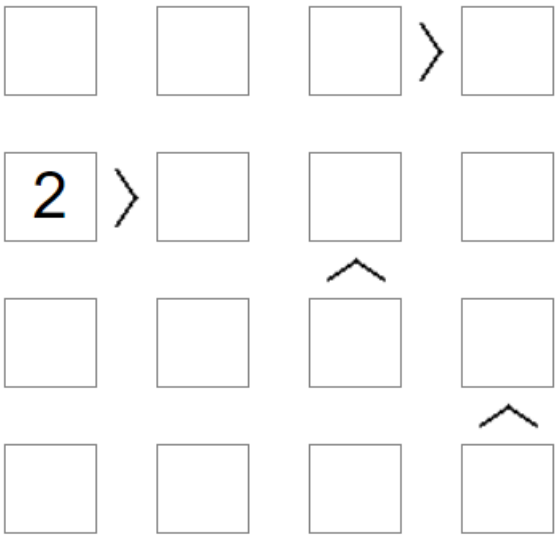
Mihălucă Sergiu Adrian – 1405A

Profesor Îndrumator: Prof. Leon Florin

Cuprins

1. Descrierea problemei

Puzzle-urile ***Futoshiki*** sunt puzzle-uri de tip planșă și sunt cunoscute și sub numele de ***Inegal.*** Ele constau într-o tablă de joc pătrata de o mărime fixă dată(de exemplu 4x4). Scopul jocului este de a completă pătratele fără cifre în așa fel încât să se respecte inegalitățile dintre pătrate, iar cifrele să fie unice(de la 1 la lungimea tablei) pe linii și coloane.



Exemplu de rezolvare:

Soluția unui puzzle Futoshiki este **unică.**

2. Aspecte teoretice privind alogoritmul

Algoritmul de forward-checking este, în estență, un algoritm de backtracking modificat. Schimbările fac ca algoritmul de backtracking să testeze modular validitatea plasării unei cifre într-un pătrățel, generând soluția treptat, comparativ cu backtrackingul clasic, care genereaza toate posibilitățile și testează validitatea contra datelor problemei.

Algoritmul de backtracking în pseudocod este banal:

```
func SolutiaEsteValida(matrice s)
    daca s contine 0-uri
        Solutia este invalida
    Solutia este valida

func GasesteSolutia(matricea s, linia l, coloana c, valoarea v)
    daca SolutiaEsteValida(s)
        Programul se opreste.
    else
        daca(pozitia (l,c) este necompletata)
            pentru (v de la 1 la lungimea matricii s)
                daca(v plasat pe pozitia (l,c) este valida)
                    pune v pe pozitia (l,c)
                    l_nou, c_nou <- IncrementeazaPozitia(l,c)
                    GasesteSolutia(s, l_nou, c_nou, v)
                    sterge v de pe pozitia (l,c)
        else
            l_nou, c_nou <- IncrementeazaPozitia(l,c)
            GasesteSolutia(s,l_nou, c_nou, v)
```

În acest mod, dacă soluția nu a fost încă găsită, algoritmul va testa plasarea unei valori pe o poziție, iar, dacă plasarea satisface condițiile de validare, o va plasa și va trece la următoarea poziție. Totuși, dacă, la un pas ulterior, algoritmul invalidează toate valorile posibile, acesta se va întoarce la o poziție anterioara pentru a testa alte valori, generând noi secvențe până la găsirea unei soluții satisfăcatoare. În caz contrar, dacă algoritmul își termină rularea și nu a găsit o soluție satisfăcătoare, înseamnă că puzzle-ul este imposibil de rezolvat.

Funția de validare este construită în 3 pași:

1. Testează că valoarea de plasat nu mai există pe linia și coloana poziției.

```
func Valideaza(matricea s, linia l, coloana c, valoarea v)
    pentru x de la 0 la lungimea_s-1
            daca s[x,c] = val
                returneaza fals
            daca s[l,x] = val
                returneaza fals
```

2. Testează inegalitățile pe orizontală

```
8       daca inegalitatea orizontala la stanga este nesatisfacuta
9           returneaza fals
10      daca inegalitatea orizontala la dreapta este nesatisfacuta
11          returneaza fals
```

3. Testează inegalitățile pe verticală

```
13      daca inegalitatea verticala in sus este nesatisfacuta
14          returneaza fals
15      daca inegalitatea verticala in jos este nesatisfacuta
16          returneaza fals
```

În finalul validării, dacă funcția nu a gaist nici un caz nesatisfăcut, aceasta va return adevărat, adică valoarea poate fi plasată pe poziția (l,c).

```
18      returneaza adevarat
```

4. Părți de cod semnificative

Avem functia **NewGrid** care initializeaza tabela de joc și reține datele de intrare(numerele deja prestabilite și inegalitatile orizontale și verticale)

```
public void NewGrid(string _numere, string _inegalitatiOrizontale, string _inegalitatiVerticale)
{
    try
    {
        int x = 0;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                numere[i, j] = Convert.ToInt32(_numere[x]) - 48;
                x++;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
    //<(stanga mai mic ca dreapta) = 2 && >(dreapta mai mic ca stanga) = 1 && -(nimic) = 0
    try
    {
        int x = 0;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                inegalitatiOrizontale[i, j] = Convert.ToInt32(_inegalitatiOrizontale[x]) - 48;
                x++;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
    // ^(sus mai mic ca jos) = 2 && v(jos mai mic ca sus) = 1 && -(nimic) = 0
    try
    {
        int x = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                inegalitatiVerticale[i, j] = Convert.ToInt32(_inegalitatiVerticale[x]) - 48;
                x++;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}
```

Functia **Valid** – Valideaza plasarea unei cifre la poziția i j, în contextul tabelei actuale.

```
1 reference
private bool Valid(int _i, int _j, int _digit, FutoshikiGrid grid)
{
    //verifica daca exista cifra _digit pe linia _i si coloana _j
    for (int x = 0; x < 4; x++)
    {
        if (grid.numere[x, _j] == _digit)
            return false;
        if (grid.numere[_i, x] == _digit)
            return false;
    }

    //verifica daca se potrivesc inegalitatile daca _digit este plasat pe pozitia [_i,_j]
    //in prima faza, testam inegalitatile orizontale
    //mai intai, cazurile in care _j se afla in interiorul liniei

    if (_j > 0 && _j < 3)
    {
        // la stanga casutei
        if (inegalitatiOrizontale[_i, _j - 1] == 2)
        {
            if (grid.numere[_i, _j - 1] != 0)
                if (!(grid.numere[_i, _j - 1] < _digit))
                    return false;
        }
        if (inegalitatiOrizontale[_i, _j - 1] == 1)
        {
            if (!(grid.numere[_i, _j - 1] > _digit))
                return false;
        }

        //la dreapta casutei
        if (inegalitatiOrizontale[_i, _j] == 2)
        {
            if (grid.numere[_i, _j + 1] != 0)
                if (!(grid.numere[_i, _j + 1] > _digit))
                    return false;
        }
        if (inegalitatiOrizontale[_i, _j] == 1)
        {
            if (!(grid.numere[_i, _j + 1] < _digit))
                return false;
        }
    }
```

```
//cazurile in care _digit se va pozitiona pe prima sau ultima coloana
if (_j == 0)
{
    //prima coloana
    if (inegalitatiOrizontale[_i, _j] == 2)
    {
        if (grid.numere[_i, _j + 1] != 0)
            if (!(grid.numere[_i, _j + 1] > _digit))
                return false;
    }
    if (inegalitatiOrizontale[_i, _j] == 1)
    {
        if (!(grid.numere[_i, _j + 1] < _digit))
            return false;
    }
}
if (_j == 3)
{
    //u  (parameter) int _j
    if (inegalitatiOrizontale[_i, _j - 1] == 2)
    {
        if (grid.numere[_i, _j - 1] != 0)
            if (!(grid.numere[_i, _j - 1] < _digit))
                return false;
    }
    if (inegalitatiOrizontale[_i, _j - 1] == 1)
    {
        if (!(grid.numere[_i, _j - 1] > _digit))
            return false;
    }
}
```

```
//validam inegalitatile verticale
if (_i > 0 && _i < 3)
{
    if (inegalitatiVerticale[_i - 1, _j] == 2)
    {
        if (grid.numere[_i - 1, _j] != 0)
            if (!(grid.numere[_i - 1, _j] < _digit))
                return false;
    }
    if (inegalitatiVerticale[_i - 1, _j] == 1)
    {
        if (!(grid.numere[_i - 1, _j] > _digit))
            return false;
    }
    if (inegalitatiVerticale[_i, _j] == 2)
    {
        if (grid.numere[_i + 1, _j] != 0)
            if (!(grid.numere[_i + 1, _j] > _digit))
                return false;
    }
    if (inegalitatiVerticale[_i, _j] == 1)
    {
        if (!(grid.numere[_i, _j] < _digit))
            return false;
    }
}

if (_i == 0)
{
    //prima linie
    if (inegalitatiVerticale[_i, _j] == 2)
    {
        if (grid.numere[_i + 1, _j] != 0)
            if (!(grid.numere[_i + 1, _j] > _digit))
                return false;
    }
    if (inegalitatiVerticale[_i, _j] == 1)
    {
        if (!(grid.numere[_i + 1, _j] < _digit))
            return false;
    }
}
if (_i == 3)
{
    //ultima linie
    if (inegalitatiVerticale[_i - 1, _j] == 2)
    {
        if (grid.numere[_i - 1, _j] != 0)
            if (!(grid.numere[_i - 1, _j] < _digit))
                return false;
    }
    if (inegalitatiVerticale[_i - 1, _j] == 1)
    {
        if (!(grid.numere[_i - 1, _j] > _digit))
            return false;
    }
}
return true;
```

Functia propriu-zisa de BKT **TrySolveBKT:**

```
3 references
public void TrySolveBKT(FutoshikiGrid grid, int i, int j, int digit)
{
    if (SolutieValida(grid))
    {
        solved = true;
    }
    else if (solved == false)
    {
        if (i < 4)
        {
            //if-ul asta face sa nu se modifice valorile din grid date initial
            if (grid.numere[i, j] == 0)
            {

                for (int aux = digit; aux <= 4; aux++)
                {
                    if (Valid(i, j, aux, grid))
                    {
                        grid.numere[i, j] = aux;
                        j++;
                        if (j == 4)
                        {
                            i++; j = 0;
                        }
                        TrySolveBKT(grid, i, j, 1);
                        if (j == 0)
                        {
                            j = 3; i--;
                        }
                        else
                        {
                            j--;
                        }
                        grid.numere[i, j] = 0;
                    }
                }
            }
            else
            {
                j++;
                if (j == 4)
                {
                    i++; j = 0;
                }
                TrySolveBKT(grid, i, j, 1);
            }
        }
    }
}
```

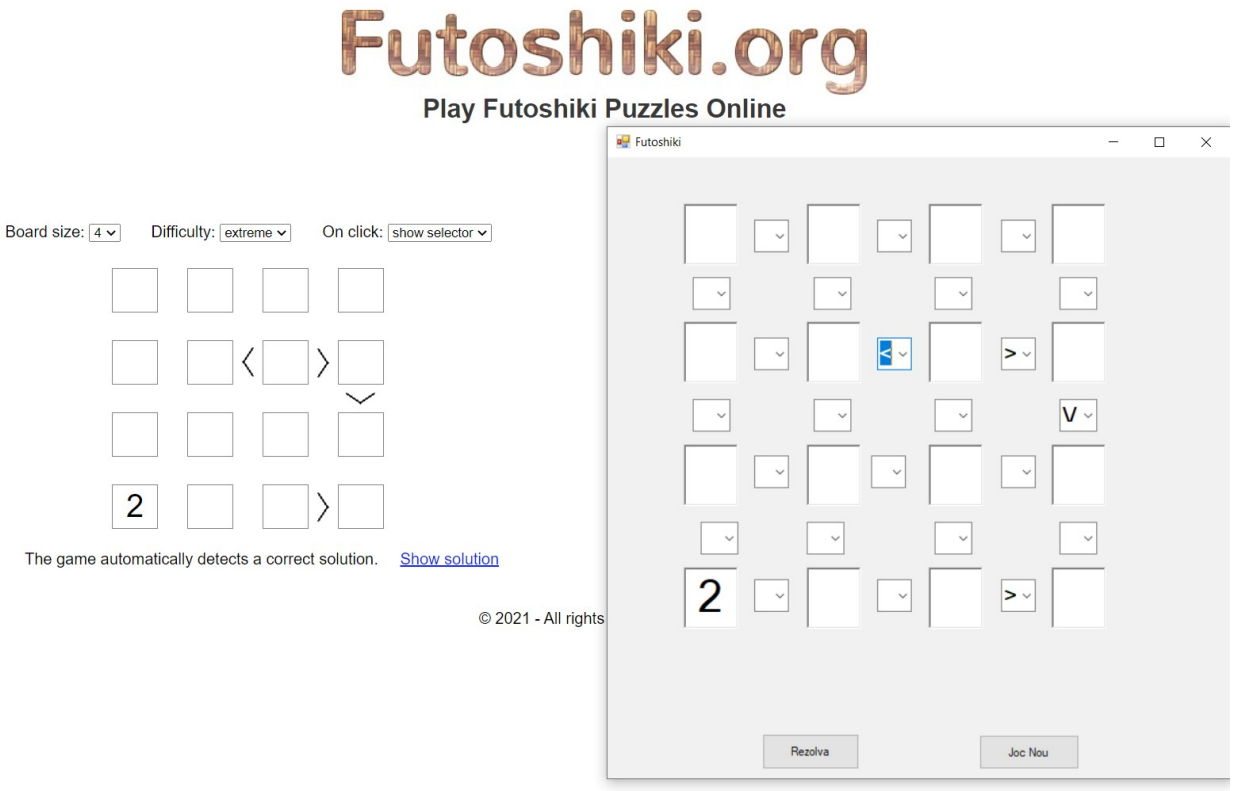
Și clasa statica **soluție** pe care o folosim pentru salvarea solutiei și afisarea acesteia.

```
namespace Futoshiki
{
    17 references
    static class Solutie
    {
        public static int[,] solutie = new int[4,4];

    }
}
```

5. Rularea programului

Se deschide o fereastra in care introducem un scenariu dorit pentru rezolvare. Luam de pe site-ul Futoshiki.org un exemplu extrem.



Dupa ce apasam butonul rezolva, algoritmul va afisa un MessageBox in care ni se va preciza fie ca exista o solutie daca scenariul este corect si o fiseaza in form sau va afisa un mesaj corespunzator ca nu exista o solutie.

Cazul in care scenariul este corect:

Pentru a introduce un nou scenariu, se apasa butonul Joc Nou care care va elibera tabela pentru introducerea unui joc nou.

Iata si un scenariu in care Nu se poate gasi rezolvare(1 nu poate fi mai mare decat 2 si nici mai mare decat 4). In concluzie algoritmul ar trebui sa afiseze un mesaj corespunzator ca nu exista solutie pentru acest Scenariu.

Nota: Bug descoperit- Se pare ca la un scenariu gresit Nu am luat in considerare numerele deja introduse(doar inegalitatile) la ora 12.02 21.01.2021 si astfel chiar daca scenariul este gresit algoritmul afiseaza o solutie partiala.

6.Concluzii

Algoritmul Forward Checking este o varianta mult imbunatatia a BKT-ul clasic, acesta construind solutia pas cu pas, reducand mult spatiul de solutii si timpul de calcul.

7. Bibliografie

Futoshiki Online Puzzles

https://www.geeksforgeeks.org/backtracking-introduction/

Games (tuiasi.ro)

8 Contributia fiecarui student

Iovu Vali Cristian a proiectat interfața grafică și funționalitatea acesteia.
Pușcalău Robert-George a fost responsabil de algoritmul de backtracking și funcția de validare, implementându-l împreună cu Vali și Sergiu.
Mihălucă Sergiu-Adrian a redactat documentația proiectului și a integrat algoritmul de rezolvare a puzzle-ului cu interfața grafică.