Universitatea Tehnică "Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare
Domeniul Calculatoare și Tehnologia Informației
Specializarea Tehnologia Informației

INTELIGENȚĂ ARTIFICIALĂ

Implementarea jocului "Magic Square" cu ajutorul algoritmului Forward Checking

Coordonator,
Prof. dr. Florin Leon

Studenți,
Duciuc Dănuț, 1409A
Amăriuță Tiberiu, 1409A
Strugaru Emanuel-Grigore, 1409A

An universitar 2023-2024

Cuprins

1. Descrierea problemei considerate

Magic Square este o problemă clasică în domeniul matematicii recreative, definit ca un pătrat/matrice ce conține numere distincte, aranjate astfel încât suma finală de pe fiecare linie, coloană si diagonală, să fie aceeași. Fiecare utilizator poate obține un pătrat magic, introducând dimensiunea acestuia, cât și suma dorită.

2. Aspecte teoretice privind algoritmul

2.1. Algoritmul Forward Checking

Algoritmul Forward Checking este o "tehnică" de satisfacere a constrângerilor utilizată în algoritmi de căutare pentru a îmbunătății eficiența prin, eliminând posibilitățile invalide în timpul căutării. Acesta constă într-o funcție care se concentrează pe reducerea domeniilor variabilelor în funcție de valorile atribuite deja altor variabile.

2.2. Descrierea detaliată a algoritmului

Se inițializează variabilele și domeniile variabilelor conform problemei date. Se iterează peste domeniul variabilei selectate, iar pentru fiecare valoare din domeniu, se verifică dacă această este consistentă cu valorile deja atribuite altor variabile și dacă verifică targetul propus.

Se folosește Backtracking pentru a trece recursiv la fiecare variabilă, iar procesul se continuă până când este identificată o soluție.

Forward Checking ajută la evitarea explorării inutile a spațiului de căutare, concentrându-se pe domeniile variabilelor care sunt influențate de atribuirile anterioare. Acest lucru duce la o reducere semnificativă a spațiului de căutare și accelerează găsirea soluției sau identificarea faptului că nu există soluții.

3. Modalitatea de rezolvare

Algoritmul Forward Checking este folosit pentru o aplicație cu funcționalitatea de a calcula „pătratul magic". Pentru a determina un Magic Square, utilizatorul trebuie să introducă atât dimensiunea pătratului cât și suma pentru care se așteaptă rezultatul.

Se definesc următoarele atribute pentru clasa MagicSquareSolver:

- `private int[,] square;`
- `private int size;`
- `private int targetSum;`
- `private int[] rowsSum;`
- `private int[] colSum;`
- `private int primDiagSum, secDiagSum;`
- `private List<int>[,] posSquare;`

Câmpurile cu care vom completa pătratul din interfață se află în square, iar dimensiunea pătratului, împreună cu suma target de pe linii/coloane/diagonale se află în size, respectiv targetSum. Se folosesc liste pentru sumele de pe fiecare rând, coloană și doi întregi pentru sumele de pe diagonală.

În ceea ce privește algoritmul de verificare înainte, se folosește o matrice de liste, unde, pentru fiecare element din matrice, avem o listă de întregi folosiți pentru a

completa suma curentă rândului/coloanei/diagonalei. Algoritmul conține și o parte de implementare ce reduce domeniul valabil fiecărui element din matrice, pentru a contribui la accelerarea procesului de căutare a soluțiilor.

4. Părți semnificative din codul sursă

4.1. Funcția de plasare a elementelor din Magic Square

Metoda PlaceNumber implementează atât algoritmul de Backtracking pentru parcurgerea elementelor posibil valide pentru pătratul magic, cât și reducerea domeniilor.

```
private bool PlaceNumber(int row, int col, List<int>[,] posSquare)
{

    if (row == size)
    {
        return IsValid();
    }

    if(col == 0 && row > 0)
    {
        int s = 0;
        for(int i=0; i<size;i++)
        {
            s += square[row - 1,i];
        }
        if (s != targetSum)
            return false;
    }
    else if (row == size-1 && col > 0)
    {
        int s = 0;
        for (int i = 0; i < size; i++)
        {
            s += square[i, col-1];
        }
        if (s != targetSum)
            return false;
    }

    List<int>[,] savedPosSquare;

    int nextRow = col == size - 1 ? row + 1 : row;
    int nextCol = col == size - 1 ? 0 : col + 1;

    foreach(var num in posSquare[row,col])
    {
        square[row, col] = num;
        rowsSum[row] += num;
        colSum[col] += num;
        if(row == col)
            primDiagSum += num;
        if (row + col == size - 1)
            secDiagSum += num;
        if (rowsSum[row] <= targetSum && colSum[col] <= targetSum &&
primDiagSum <= targetSum && secDiagSum <= targetSum)
        {
            savedPosSquare = DeepCopyDomains(posSquare);
```

```
                    //Removing from all domains
                    for (int i = row; i < size; i++)
                    {
                        for (int j = 0; j < size; j++)
                        {
                            if (!(i == row && j == col))
                            {
                                savedPosSquare[i, j].Remove(num);
                            }
                        }
                    }
                    //Removing from column
                    for (int i=row+1;i<size;i++)
                    {
                            savedPosSquare[i, col].RemoveAll(x => x > (targetSum
- colSum[col]));
                    }
                    //Remove from row
                    for (int i = col+1; i < size; i++)
                    {
                            savedPosSquare[row, i].RemoveAll(x => x > (targetSum
- rowsSum[row]));
                    }
                    //Remove for primary diagonal
                    if(row == col)
                    {
                        for(int i=row+1;i<size;i++)
                        {
                                savedPosSquare[i, i].RemoveAll(x => x >
(targetSum - primDiagSum));
                        }
                    }
                    //Remove for secondary
                    if(row + col == size-1)
                    {
                        for(int i=row+1;i<size;i++)
                        {
                                savedPosSquare[i,size-1-i].RemoveAll(x => x >
(targetSum - secDiagSum));
                        }
                    }

                    if(!CheckIfDomainsVoid(savedPosSquare))
                        if (PlaceNumber(nextRow, nextCol, savedPosSquare))
                            return true;

                }

                square[row, col] = 0; // Backtrack
                rowsSum[row] -= num;
                colSum[col] -= num;
                if(row == col)
                    primDiagSum -= num;
                if (row + col == size - 1)
                    secDiagSum -= num;
            }

            return false;
               }
```

4.2. Funcția de verificare înainte cu reducerea domeniilor

Pentru a eficientiza procesul de determinare al elementelor pătratului magic, se face o verificare înainte. Se verifică da suma target este depășită. În caz contrar, se elimină elementul folosit pentru o poziție, din domeniile specifice celorlalte elemente ale pătratului.

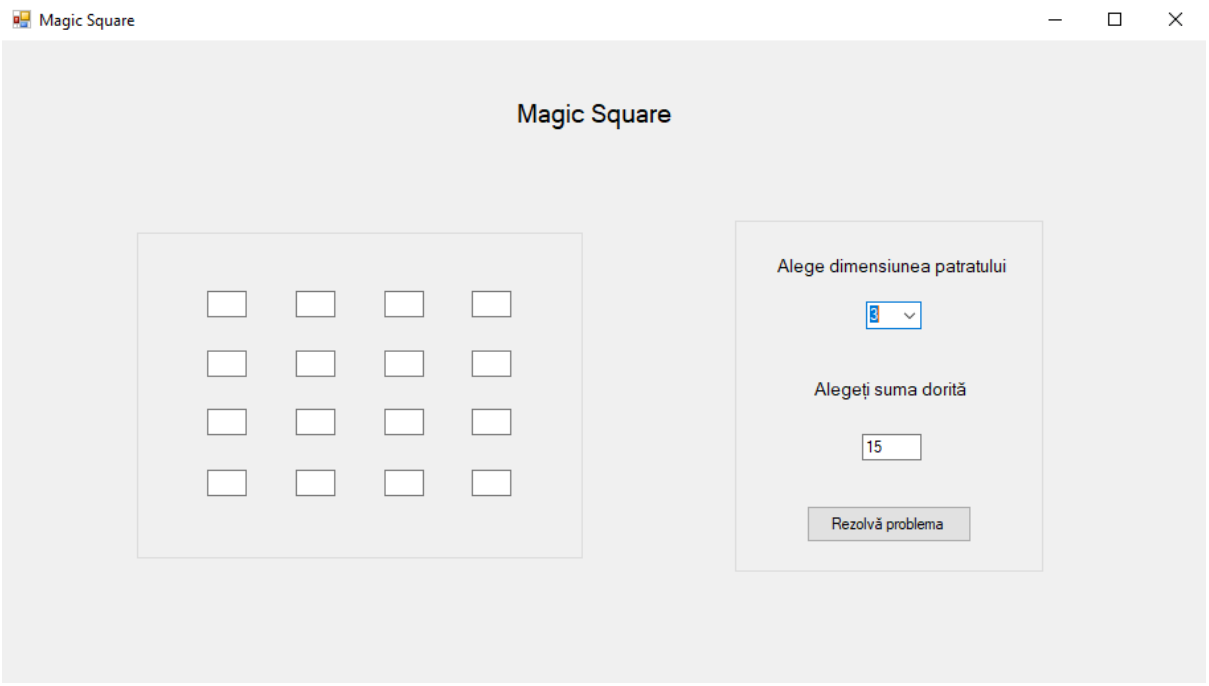
```
                if (rowsSum[row] <= targetSum && colSum[col] <= targetSum &&
primDiagSum <= targetSum && secDiagSum <= targetSum)
                {
                    savedPosSquare = DeepCopyDomains(posSquare);
                    //Removing from all domains
                    for (int i = row; i < size; i++)
                    {
                        for (int j = 0; j < size; j++)
                        {
                            if (!(i == row && j == col))
                            {
                                savedPosSquare[i, j].Remove(num);
                            }
                        }
                    }
                    //Removing from column
                    for (int i=row+1;i<size;i++)
                    {
                            savedPosSquare[i, col].RemoveAll(x => x > (targetSum
- colSum[col]));
                    }
                    //Remove from row
                    for (int i = col+1; i < size; i++)
                    {
                            savedPosSquare[row, i].RemoveAll(x => x > (targetSum
- rowsSum[row]));
                    }
                    //Remove for primary diagonal
                    if(row == col)
                    {
                        for(int i=row+1;i<size;i++)
                        {
                                savedPosSquare[i, i].RemoveAll(x => x >
(targetSum - primDiagSum));
                        }
                    }
                    //Remove for secondary
                    if(row + col == size-1)
                    {
                        for(int i=row+1;i<size;i++)
                        {
                                savedPosSquare[i,size-1-i].RemoveAll(x => x >
(targetSum - secDiagSum));
                        }
                    }

                    if(!CheckIfDomainsVoid(savedPosSquare))
                        if (PlaceNumber(nextRow, nextCol, savedPosSquare))
                            return true;

                }
```

4.3. Verificare sumă pentru fiecare coloană/linie/diagonală

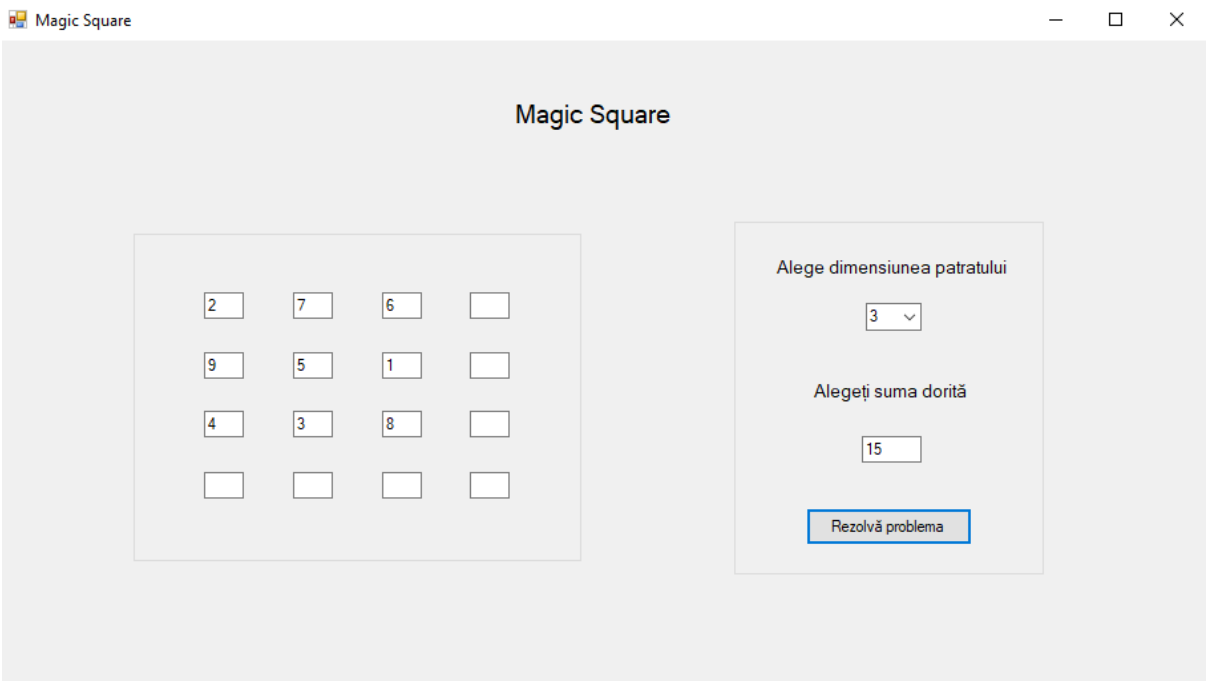
Atunci când indexul rândului atinge rândul maxim din matrice, se verifică dacă pătratul este valid prin metoda IsValid.

```
public bool IsValid()
{
    int sum;

    // Check rows and columns
    for (int i = 0; i < size; i++)
    {
        sum = 0;
        for (int j = 0; j < size; j++)
            sum += square[i, j];
        if (sum != targetSum)
            return false;

        sum = 0;
        for (int j = 0; j < size; j++)
            sum += square[j, i];
        if (sum != targetSum)
            return false;
    }

    // Check diagonals
    sum = 0;
    for (int i = 0; i < size; i++)
        sum += square[i, i];
    if (sum != targetSum)
        return false;

    sum = 0;
    for (int i = 0; i < size; i++)
        sum += square[i, size - 1 - i];
    if (sum != targetSum)
        return false;

    return true;
}
```

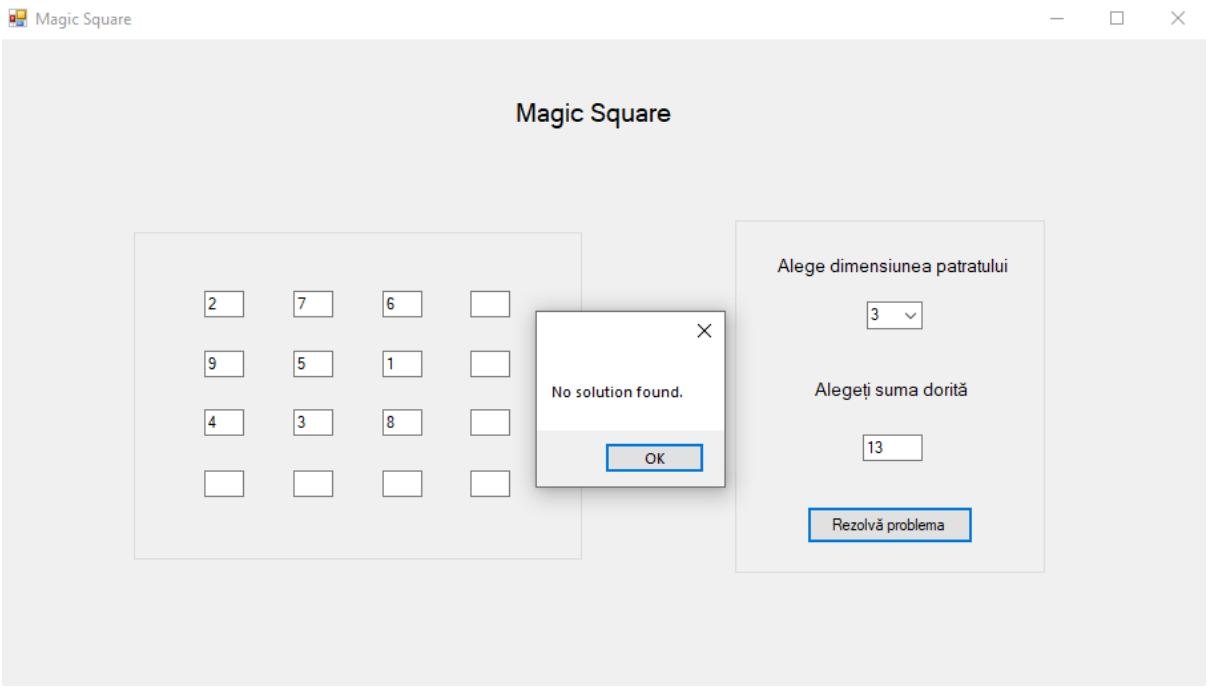
5. Rezultatele obținute prin rularea programului



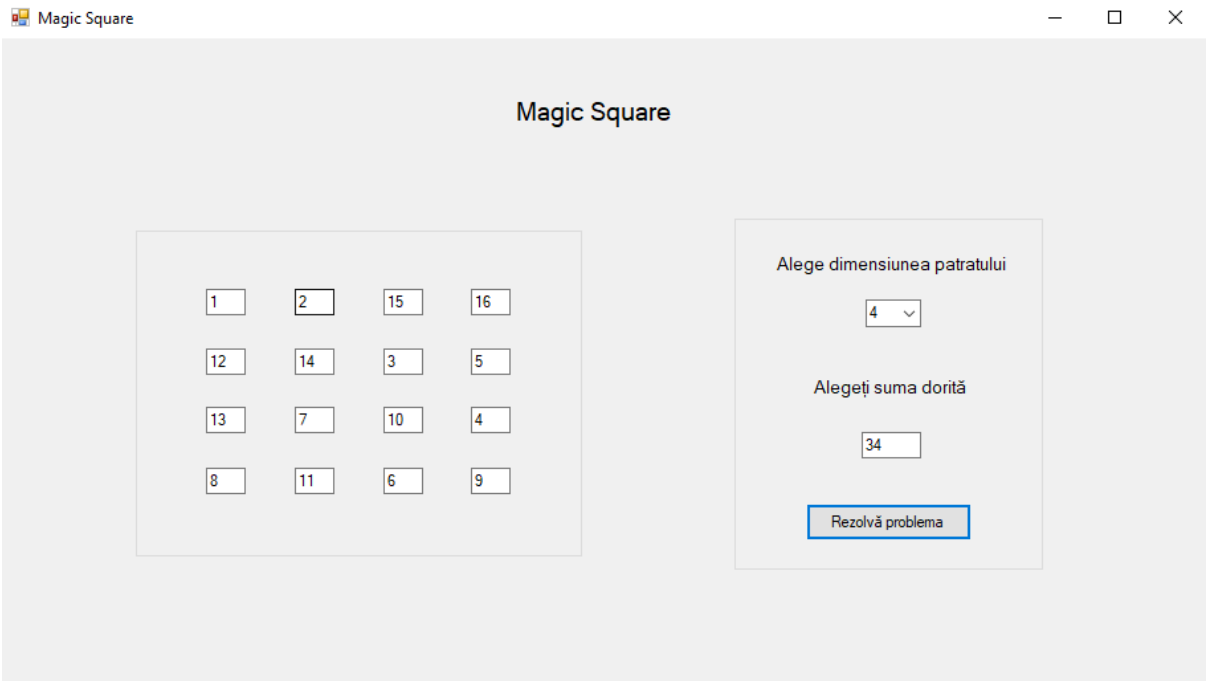
Figură 1 Interfață aplicație



Figură 2 Exemplu determinare Magic Square



Figură 3 Exemplu sumă invalidă



Figură 4 Exemplu sumă valid pentru 4x4

6. Concluzii

Algoritmul Forward Checking este o tehnică eficientă utilizată în rezolvarea problemelor de satisfacere a constrângerilor, cu eficiență în reducerea domeniilor, eliminând valorile care nu sunt consistente cu constrângerile impuse. Această abordare poate contribui semnificativ la accelerarea procesului de căutare a soluțiilor.

7. Rolul fiecărui membru al echipei

Duciuc Dănuț

- implementare cod (CheckIfDomainsVoid, IsValid, PrintSquare)
- documentație (punctele 1, 2 și 3)

Amăriuță Tiberiu

- realizare interfață grafică
- documentație (punctele 6, 4.2, și 4.3)

Strugaru Emanuel-Grigore

- implementare cod (PlaceNumber, DeepCopyDomains, GetSolution)
- documentație (punctele 4.1, 5)