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

EVALUAREA PERFORMANȚELOR

Rezolvarea problemei „Magic Square" folosind algoritmul de verificare înainte

Coordonator,
Elena Iorga

Studenți,
Tiberiu Amăriuță
Dănuț Duciuc
Emanuel Strugaru
Grupa 1409A

An universitar 2023-2024

1. Enunțarea temei

Tema proiectului constă în implementarea algoritmului de verificare înainte (en. Forward Checking) pentru rezolvarea problemei "Magic Square".

Un pătrat magic este o structură matematică, sub formă de matrice, în care trebuie așezat un set de numere astfel încât suma elementelor de pe fiecare linie, coloană și cele două diagonale principale să fie identică.

Algoritmul de verificare înainte reprezintă o metodă recursivă de explorare a tuturor posibilităților în vederea identificării soluției optime. În contextul pătratelor magice, acesta ne permite să iterăm prin diferite aranjamente ale numerelor în matrice, evaluând în mod continuu dacă respectă condițiile pătratului magic. Atunci când se identifică o configurație incorectă, algoritmul revine la stadiul anterior pentru a încerca o altă variantă. Acest proces se repetă până când se găsește o soluție validă sau se explorează toate opțiunile posibile.

2. Arhitectura generală

Proiectul a fost realizat in C# , iar interfața a fost creată în WindowsForms. Principalele clase ale proiectului sunt:

- *FormMagicSquare* – conține interfața și este responsabilă de acțiunile pe care le poate face un utilizator: să aleagă dimensiunea pătratului și suma pentru care se dorește rezolvarea acestuia;
- *MagicSquareSolver* – conține toate funcțiile necesare pentru rezolvarea pătratului magic.

Metodele implementate in cadrul acestei clase sunt următoarele:

`MagicSquareSolver(int size, int targetSum = 15)` – constructorul clasei, inițializează dimensiunea pătratului, suma pentru care se dorește rezolvarea și domeniul valorilor pe care îl poate lua fiecare element al pătratului;

`PlaceNumber(int row, int col, List<int>[,] posSquare)` – funcție ce determină poziția în care trebuie pusă fiecare valoare, fiind apelată recursiv;

`DeepCopyDomains(List<int>[,] originalDomains)` - funcție ce copiază domeniile de valori pentru elementele pătratului între apelurile recursive;

`CheckIfDomainsVoid(List<int>[,] domain)` – funcție ce verifică dacă domeniul de valori pe care le poate lua un element al pătratului este vid;

`Solve()` – apelează recursive funcția *PlaceNumber* pentru a găsi o soluție;

`IsValid()` – funcție ce verifică dacă soluția găsită este corectă și daca sunt respectate toate condițiile.

3. Funcționalitatea

Aplicația își propune să rezolve problema pătratului magic în funcție de preferințele utilizatorului.

Utilizatorul poate alege dimensiunea pătratului ce trebuie rezolvat și de asemenea poate introduce și suma care trebuie obținută pe linie, coloană și diagonală. În continuare, datele furnizate de utilizatorul sunt preluate și folosite în aplicarea algoritmului de verificare înainte pentru a genera și valida diferite configurații ale pătratului magic. Algoritmul va itera prin diferite aranjamente ale numerelor, verificând constant dacă acestea respectă condițiile pătratului magic.

Dacă utilizatorul nu introduce nicio valoare, se vor folosi valorile predefinite în program (dimensiune 3x3 si suma minimă posibilă).

Rezultatul final va fi afișat utilizatorului în următoarele moduri: daca este găsit un pătrat magic valid, elementele sale vor fi afișate în căsuțele corespunzătoare de pe interfață, iar dacă nu este găsit niciunul, atunci se va afișa un mesaj ce va notifica utilizatorul în legătură cu aceasta.

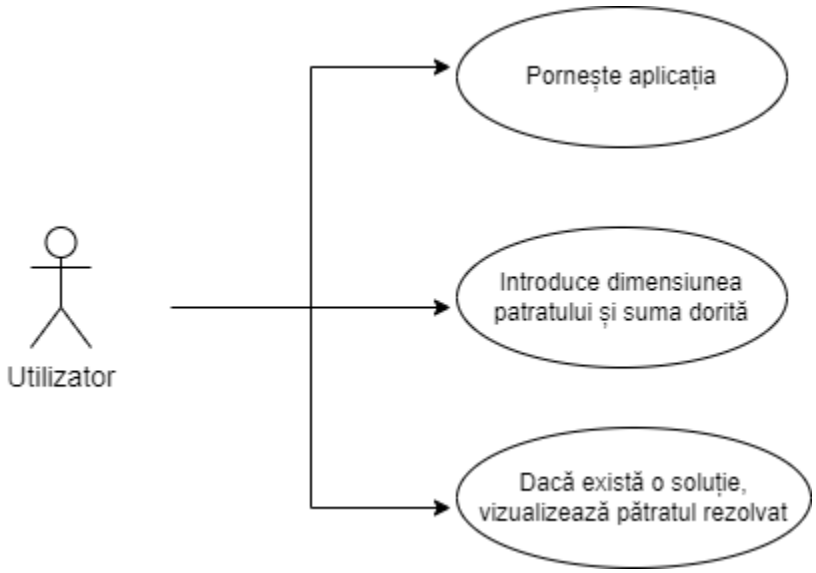
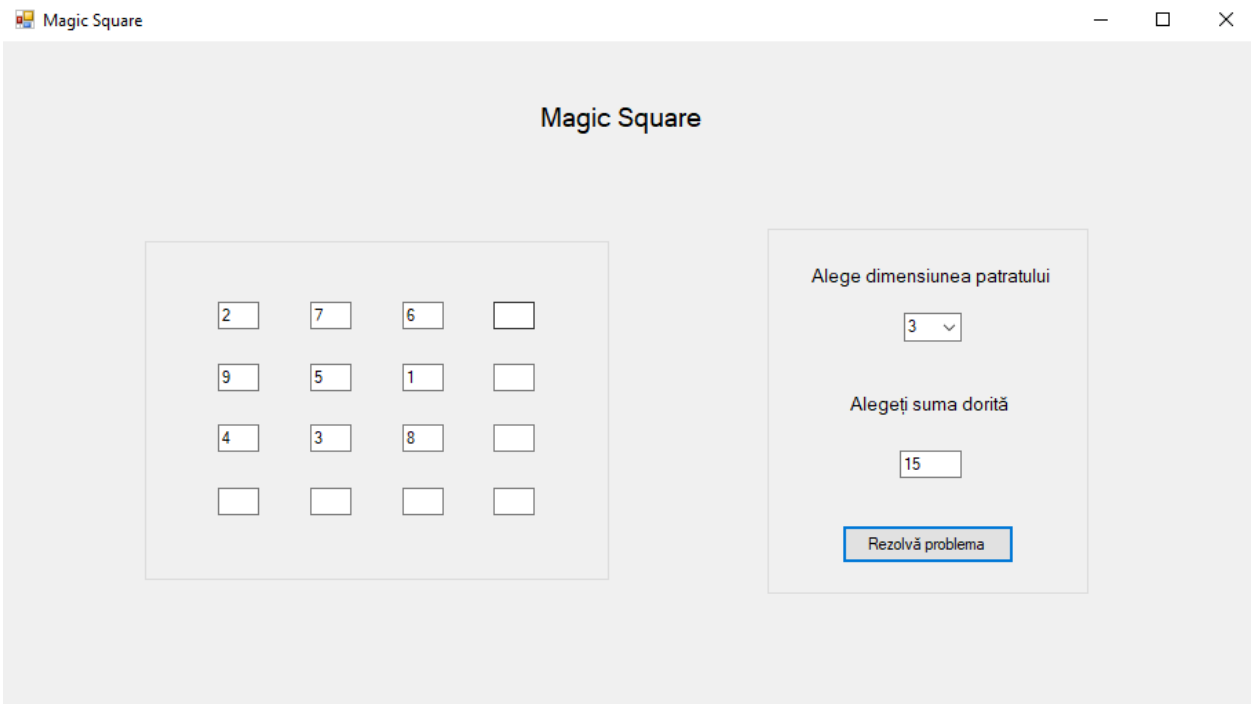
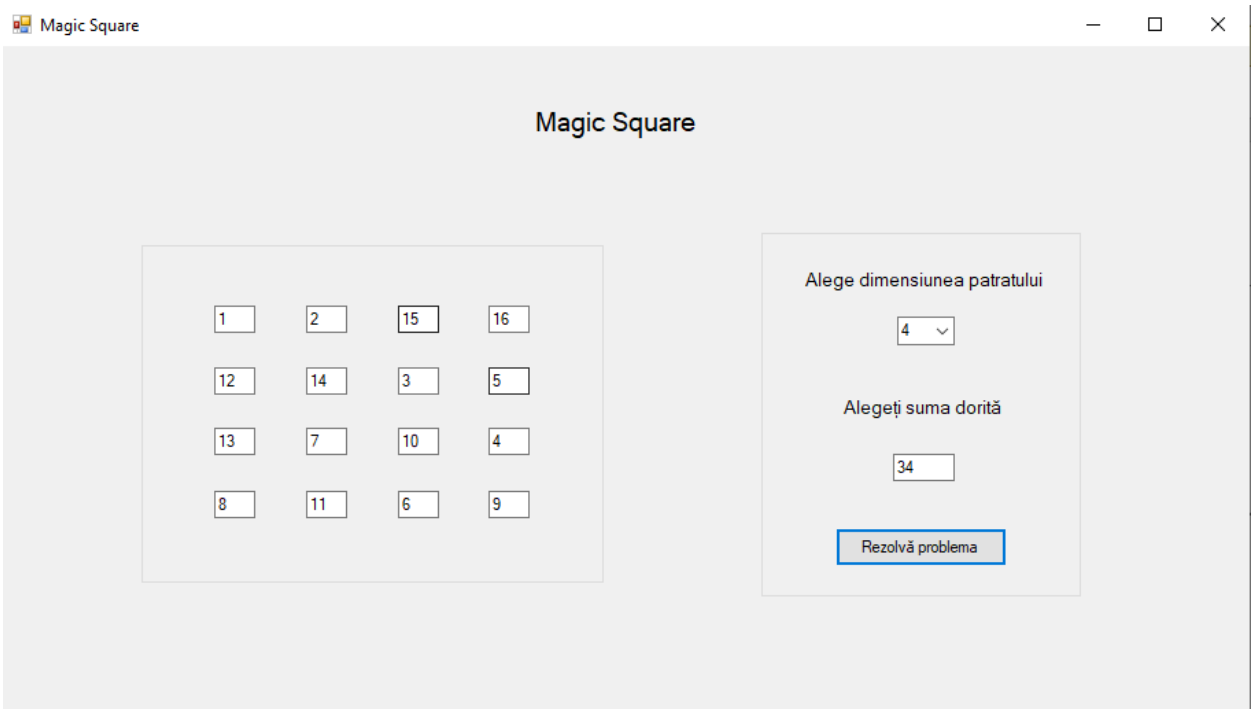


Diagrama cazurilor de utilizare



Rezolvare pătrat 3x3



Rezolvare pătrat 4x4

4. Complexitatea algoritmului

- *Cazul cel mai nefavorabil* – este atunci când se găsește soluția abia la final, după ce s-a parcurs de foarte multe ori pătratul. Ordinul de complexitate ar fi $O(n^4)$ deoarece pentru fiecare element al pătratului noi păstrăm câte o matrice cu domeniul de valori care trebuie parcursă;
- *Cazul mediu* – se află la mijlocul dintre cele 2 extreme și ar putea fi reprezentată de situația în care găsim din prima o linie sau o coloană a pătratului. Complexitatea în acest caz este $O((n^2-1) * n^2)$;
- *Cazul cel mai favorabil* – este reprezentat de situația în care găsim elementele pătratului din prima încercare și nu trebuie să mai încercăm alte variante pentru eventualele elemente ale pătratului. În acest caz complexitatea este $O(n^2)$.

5. Demonstrația corectitudinii

Invariantul presupune iterarea pe fiecare linie si coloana a pătratului. Astfel, se introduce cate o valoare pentru fiecare element al pătratului si se generează noi domenii de valori (cu valorile rămase) pentru celelalte.

Demonstrație prin inducție:

P(n) => se încearcă adăugarea unei valori pentru elementul de pe poziția (0,n). Dacă se respectă condițiile pentru sumă, se caută următoarea valoare.

P(0) => se încearcă inserarea unei valori pentru elementul de pe poziția (0, 0).

P(k) => se încearcă adăugarea unei valori pentru elementul de pe poziția (0, k).

P(k+1) => se încearcă inserarea unei valori pentru elementul de pe poziția (0, k+1)

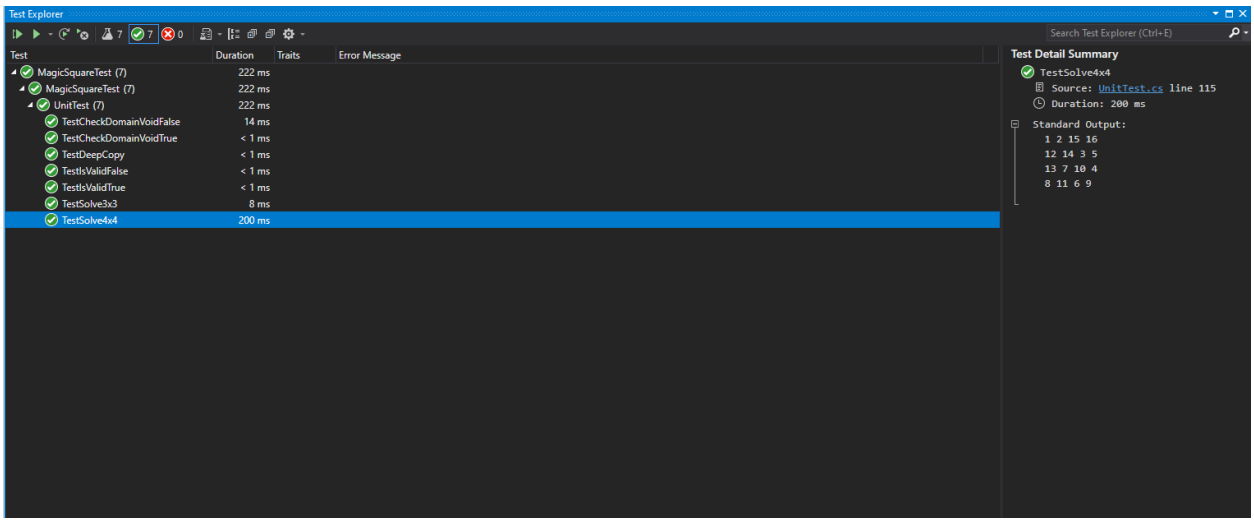
Primele k+1 elemente = primele k elemente + elementul de pe poziția k. De la 1 la k s-au adăugat deja valori, deci la iterația k+1 se va adăuga se va adăuga o valoare din domeniul de valori rămase pentru elementul k+1.

Din explicațiile d mai sus rezultă că P(k+1) este adevărată, deci P(n) este adeărată.

6. Explicarea testelor automate

Scopul testelor create este acela de a verifica corectitudinea și logica aplicației. Se așteaptă ca rezultatele funcțiilor de testare să respecte fluxul aplicației, astfel am creat următoarele teste:

- **TestIsValidFalse()** – verificăm dacă funcția *IsValid* returnează *False* atunci când îi dăm ca parametru un pătrat care nu respect condițiile pătratului magic;
- **TestIsValidTrue()** – verificăm dacă funcția *IsValid* returnează *True* atunci când îi dăm ca parametru un pătrat care respect[condițiile pătratului magic;
- **TestCheckDomainVoidFalse()** – verificăm dacă funcția *CheckIfDomainsVoid* returnează *False* atunci când mai sunt elemente în domeniul de valori
- **TestCheckDomainVoidTrue()** – verificăm dacă funcția *CheckIfDomainsVoid* returnează *True* atunci când nu mai sunt elemente în domeniul de valori
- **TestDeepCopy()** – verificăm dacă funcția *DeepCopyDomains* copiază corect valorile de care avem nevoie între apelurile recursive
- **TestSolve3x3()** – verificăm dacă programul rezolvă corect un pătrat de dimensiune 3x3
- **TestSolve4x4()** – verificăm dacă programul rezolvă corect un pătrat de dimensiune 4x4



Rezultatele testelor automate

7. Rolul fiecărui membru al echipei

Tiberiu Amăriuță

- realizare interfață grafică
- realizare teste automate
- documentație(punctele 1 și 6)

Dănuț Duciuc

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

Emanuel Strugaru

- implementare cod (PlaceNumber, DeepCopyDomains, GetSolution)
- documentație (punctele 5 și 6)