Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

PROGRAMARE LOGICĂ ȘI FUNCȚIONALĂ

Minesweeper

Realizat de: Edveș Alexandru, Cîrstea Andrei

Brașov
Anul 3 2023-2024

Cuprins

1 Introducere

1.1 Descrierea temei

Proiectul propus oferă o implementare a cunoscutului joc Minesweeper, utilizând algoritmi dezvoltați prin programarea logică și funcțională. Variația complexă de reguli asociate acestui joc ne-a oferit oportunitatea de a aprofunda și de a lucra cu aceste ramuri ale programării.

1.2 Obiectivul proiectului

Scopul principal al acestui proiect este de a exemplifica o situație practică în care pot fi aplicați algoritmi de programare logică și funcțională.

Minesweeper, fiind un subiect popular la nivel mondial, este jocul ideal pentru a evidenția utilitatea acestor algoritmi într-un context captivant. Astfel, proiectul își propune să aducă la cunoștință unui public a cărui număr crește constant, beneficiile și aplicabilitatea acestor tehnologii în cadrul unuia dintre cele mai îndrăgite jocuri de logică din lume.

1.3 Contribuții personale

Algoritmii utilizați în acest proiect sunt în totalitate implementați de catre noi, fără a se baza pe soluții preexistente.

Interfața principală, unde se desfășoară jocul, a fost dezvoltată de către noi, oferind utilizatorului posibilitatea de a efectua orice acțiune disponibilă și în jocul original.

2 Tehnologii folosite

- **C#** - partea de programare funcțională a acestei aplicații a fost realizată în limbajul C#, acoperind atât implementarea algoritmilor propriu-ziși, cât și partea de cod destinată interfeței grafice.
- **WPF**(Windows Presentation Foundation) - pentru a crea interfața unde se desfășoară întreaga acțiune a jocului.
- **SWI-Prolog** - a fost folosit pentru partea de programare logică a acestei aplicații.
- **Github** - pentru a facilita colaborarea asupra proiectului, având în vedere că acesta a fost dezvoltat de două persoane. Prin intermediul platformei, am putut lucra simultan la aplicație, gestionând eficient versiunile, urmărind modificările și asigurând o sincronizare eficientă a codului.

3 Modul de Funcționare

Jocul Minesweeper începe cu o matrice de celule neexplorate, unde fiecare celulă poate ascunde o mină, un număr (indicând câte mine sunt în vecinătatea imediată), sau poate fi un spațiu gol. Utilizatorul descoperă celulele cu scopul de a curăța terenul de mine, folosindu-se de numerele afișate pentru a deduce unde se află minele.

3.1 Selectarea dificultății

În orice moment al rulării aplicației, utilizatorul poate selecta dificultatea cu ajutorul chenarului corespunzător. Opțiunile disponibile sunt următoarele:

- Easy (Ușor)
- Medium (Mediu)
- Hard (Greu)

În funcție de dificultate se stabilesc dimensiunile și numărul de mine pe harta de joc.

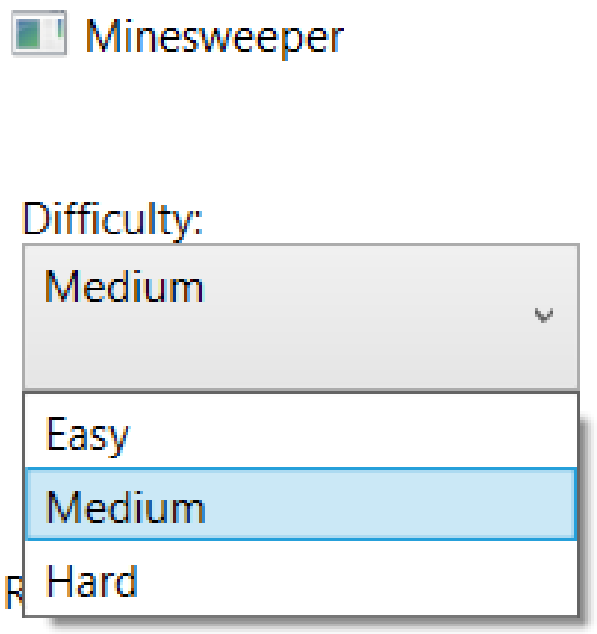

Figure 1: Selectarea dificultății

3.2 Descoperirea celulelor

Când o celulă este descoperită:

- Dacă celula conține o mină, jocul se termină.
- Dacă celula este un număr, acesta se afișează, indicând numărul de mine din celulele adiacente.
- Dacă celula este un spațiu gol (adică nu are mine în vecinătate), se descoperă automat și celulele adiacente până când sunt întâlnite numere.

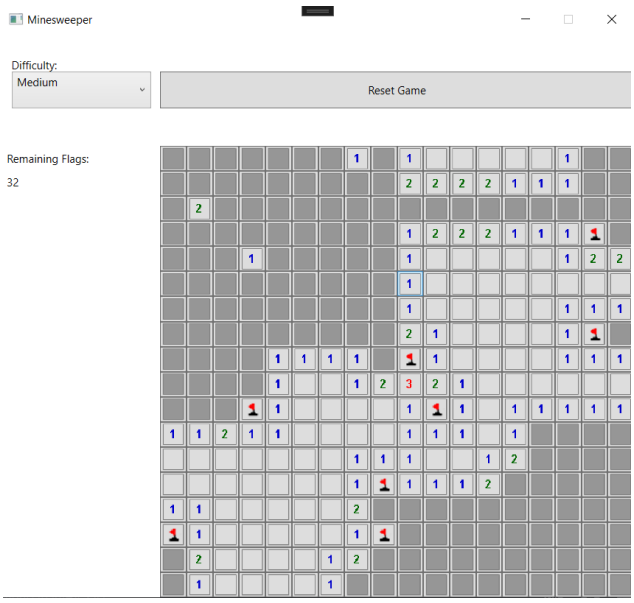

Figure 2: Aspectul jocului de joc a unei celule

3.3 Plasarea steagurilor

Utilizatorul poate plasa un steag pe o celulă neexplorată pe care o suspectează că ascunde o mină. Plasarea steagului este pur informativă și ajută la marcarea celulelor pentru a evita descoperirea accidentală a unei mine.

3.4 Verificarea finalului jocului

Jocul se termină când:

- O mină este descoperită (înfrângere).
- Toate celulele care nu ascund mine sunt descoperite (victorie).

Pentru a verifica victoria, jocul calculează dacă numărul celulelor neexplorate este egal cu numărul de mine neexplorate.

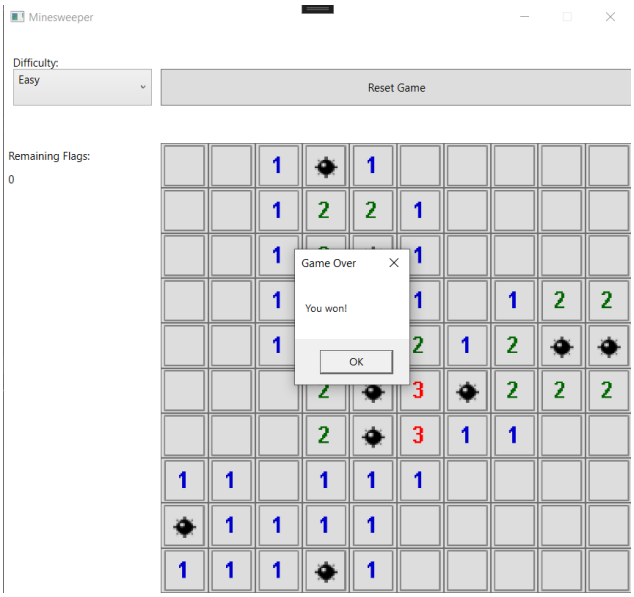

Figure 3: Ecran de final de joc

3.5 Funcționarea generală

La momentul pornirii aplicației, atunci când este schimbată dificultatea sau când se termină un joc, este pornit procesul în prolog care generează harta în funcție de dimensiunile și numărul de mine specificate. Procesul este pornit cu ajutorul System.Diagnostics din frameworkul .NET: acesta pornește programul SwiProlog și returnează rezultatul afișat de acesta.

Harta este actualizată în interfața grafică și se începe un joc nou.

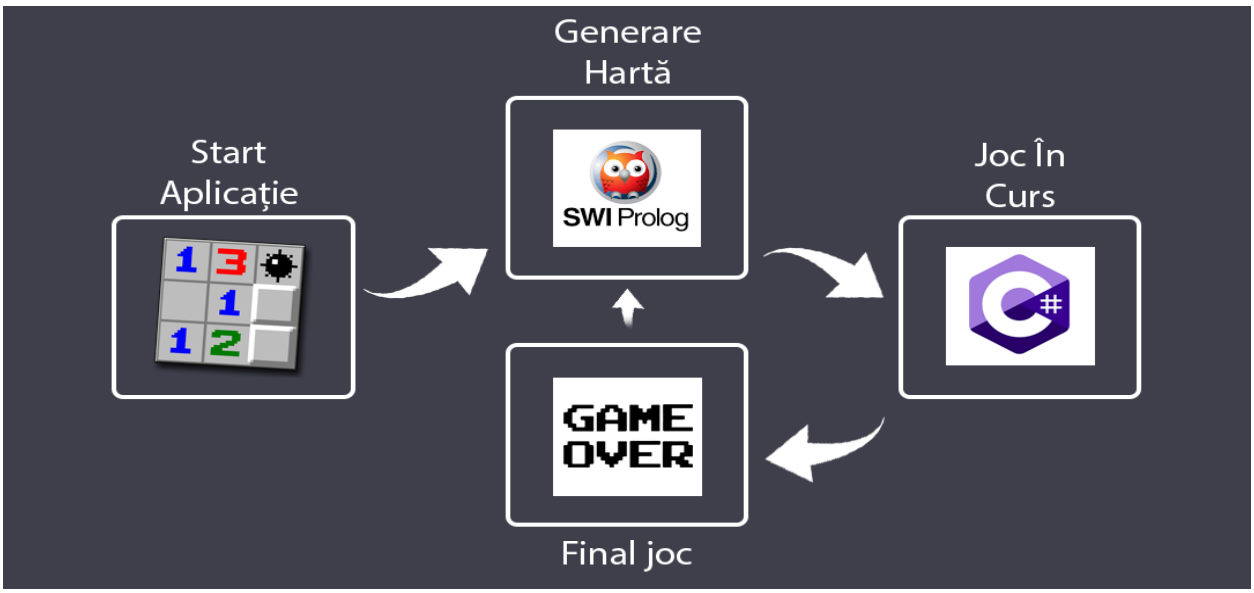

Figure 4: Ilustrare a modului de funcționare

4 Programare logică

4.1 Prezentare

Această etapă a proiectului este realizată în SWI-Prolog și constă în crearea hărții jocului Minesweeper. Procesul implică plasarea aleatoare a unui număr de bombe, precum și poziționarea numerelor corespunzătoare în jurul acestora. Harta este reprezentată sub forma unei matrice de cifre, în care cifra -1 indică prezența unei bombe, iar restul cifrelor reprezintă numărul de bombe adiacente fiecărei poziții.

```
main :-
    open('../../../PrologMap/input.txt', read, InputStream),
    read_line_to_codes(InputStream, NrowsCodes),
    number_codes(Nrows, NrowsCodes),
    read_line_to_codes(InputStream, NcolsCodes),
    number_codes(Ncols, NcolsCodes),
    read_line_to_codes(InputStream, BombsNumberCodes),
    number_codes(BombsNumber, BombsNumberCodes),
    close(InputStream),
    matrix(Nrows, Ncols, BombsNumber, Matrix),

    open('../../../PrologMap/matrix_output.txt', write, OutputStream),
    write_matrix_to_file(Matrix, OutputStream),
    close(OutputStream),

    halt.
```

Figure 5: Funcția **Main**

Funcția **Main** citește dintr-un fișier dimensiunile matricei și numărul de bombe, iar apoi apelează funcția **matrix** cu aceste valori ca parametri. Funcția **matrix** creează o matrice având dimensiunile date ca parametri și plasează bombele aleatoriu în interiorul acesteia, atribuind în mod corespunzător și numerele reprezentând numărul de bombe adiacente fiecărei poziții. Această matrice este salvată intr-un fișier pentru a putea fi folosită ulterior în partea de programare funcțională, care este detaliată în capitolul ce urmează.

4.2 Implementare

```
matrix(Nrows, Ncols, BombsNumber, R) :-
    length(Matrix, Nrows),
    length(Row, Ncols),
    maplist(=(0), Row),
    maplist(=(Row), Matrix),
    length(Matrix, ListLength),
    NewLen is ListLength ** 2,
    randset(BombsNumber, NewLen, ResSet),
    placeNumbersForBigMatrix(Matrix, ResSet, Nrows, NumMatrix),
    flatten(NumMatrix, ResMatrix),
    placeBombsInList(ResMatrix, ResSet, BombsNumber, BombsList),
    unflatten(BombsList, Ncols, R).
```

Figure 6: Funcția **matrix**

1. Se crează o matrice de dimensiunile citite din fișier și se initializeaza fiecare element cu valoarea 0.
2. Se inițializează o listă cu elemente unice de dimensiunea numărului de bombe.Această listă conține pozițiile aleatorii pe care vor fi inserate bombele în matrice.
3. Se apelează funcția **placeNumbersForBigMatrix** care parcurge recursiv matricea si incrementeaza toate elementele de pe pozitiile vecine cu pozitiile bombelor determinate la pasul anterior.

```
placeNumbersForBigMatrix([H|T], [], _, [H|T]).

placeNumbersForBigMatrix([H|T], [HSet|TSet], NRows, R):-
        Div is HSet // NRows,
    Mod is HSet mod NRows,
    placeNumbersForMatrix([H|T], Mod, Div, 0, R2),
    placeNumbersForBigMatrix(R2, TSet, NRows, R).
```

Figure 7: Funcția **placeNumbersForBigMatrix**

(a) Se apelează funcția **placeNumbersForMatrix** pentru fiecare element din matrice, funcție care incrementeaza fiecare vecin al elementului curent în cazul în care acesta este bombă.

```
placeNumbersForMatrix([], _, _, _, []).

placeNumbersForMatrix([H|T], Y, X, Contor, R):-
    in_range(Contor, X-1, X+1),
    placeNumbersForOneList(H, Y, 0, R2),
    placeNumbersForMatrix(T, Y, X, Contor+1, R4),
    R = [R2|R4].

placeNumbersForMatrix([H|T], Y, X, Contor, R):-
    placeNumbersForMatrix(T, Y, X, Contor+1, R4),
    R = [H|R4].
```

Figure 8: Funcția **placeNumbersForMatrix**

(b) Funcția **placeNumbersForOneList** este utilizată pentru a itera și a incrementa valorile fiecărei linii din matricea de vecini asociată elementului curent.

4. Funcția **placeBombsInList** este destinată inițializării elementelor de pe pozițiile bombelor cu valoarea -1, indicând astfel prezența bombei la acele poziții.

5 Programare Funcțională

Abordarea noastră în dezvoltarea jocului Minesweeper a fost concentrată pe principiile programării funcționale. Această alegere ne-a ghidat în structurarea codului și în implementarea logicii jocului, asigurându-ne că fiecare parte a sistemului este atât eficientă, cât și ușor de înțeles.

5.1 Imutabilitatea în Structura Jocului

Principiul imutabilității a fost esențial în gestionarea stării celulelor și a tablei de joc. De exemplu, la descoperirea unei celule sau la plasarea unui steag, nu modificăm direct starea existentă; în schimb, generăm o nouă stare a jocului. Această abordare elimina riscurile asociate cu mutabilitatea și efectele secundare, facilitând o logică predictibilă și ușor de urmărit.

```
public static GameState RevealCell(GameState currentState, int x, int y)
{
    var newState = CloneGameState(currentState);
    int availableFlags = newState.AvailableFlags;

    if (!IsValidCell(newState.Board.Cells, x, y) || newState.Board.Cells[x][y].IsRevealed || newState.GameOver || newState.GameWon)
    {
        return newState;
    }

    var (updatedBoard, flagChange) = UpdateBoard(newState.Board, x, y);
    availableFlags += flagChange;

    bool gameOver = updatedBoard.Cells[x][y].IsMine;

    return new GameState(
        updatedBoard,
        availableFlags,
        gameOver,
        gameOver ? false : CheckForWin(updatedBoard));
}
```

Figure 9: Tranziția imutabilă a stării jocului

5.2 Funcții Pure pentru Logica Jocului

Utilizarea funcțiilor pure a fost o altă piatră de temelie în implementarea noastră. De la calculul celulelor adiacente până la verificarea condițiilor de victorie, ne-am asigurat că funcțiile noastre sunt pure. Acest lucru înseamnă că output-ul unei funcții este determinat exclusiv de input-urile sale, fără a depinde de sau a modifica stări externe. Un exemplu concret este funcția de actualizare a tablei de joc, care primește starea curentă a tablei și coordonatele celulei acționate de utilizator, returnând o nouă stare actualizată a tablei.

```
public static (Board, int) UpdateBoard(Board currentBoard, int x, int y)
{
    int flagChange = 0;
    var newCells = currentBoard.Cells.Select(row => Map<Cell, Cell>(row.ToList(), cell => CloneCell(cell))).ToList();

    var (revealedCell, cellFlagChange) = RevealCell(newCells[x][y]);
    newCells[x][y] = revealedCell;
    flagChange += cellFlagChange;

    if (newCells[x][y].AdjacentMines == 0)
    {
        var (updatedCells, neighborsFlagChange) = RevealNeighbors(newCells, x, y);
        newCells = updatedCells;
        flagChange += neighborsFlagChange;
    }

    return (new Board(newCells), flagChange);
}
```

Figure 10: Fluxul unei funcții pure în actualizarea tablei

5.3 Utilizarea Funcțiilor de Mapare

În cadrul implementării jocului Minesweeper, am introdus o adaptare specifică a funcțiilor de mapare pentru a manipula și transforma stările celulelor. Acest lucru ne-a permis să aplicăm principii de programare funcțională într-un mod care este atât expresiv, cât și eficient. În particular, pe lângă folosirea '.Select()' din LINQ am demonstrat implementarea unei funcții de mapare cu o funcție 'Map' definită manual, care ilustrează aplicarea noastră intenționată a transformării datelor.

```
public static List<TOutput> Map<TInput, TOutput>(List<TInput> inputList, Func<TInput, TOutput> transformation)
{
    var outputList = new List<TOutput>();
    foreach (var item in inputList)
    {
        outputList.Add(transformation(item));
    }
    return outputList;
}
```

Figure 11: Exemplu de implementare a funcției Map

```
public static (Board, int) UpdateBoard(Board currentBoard, int x, int y)
{
    int flagChange = 0;
    var newCells = currentBoard.Cells.Select(row => Map<Cell, Cell>(row.ToList(), cell => CloneCell(cell))).ToList();
```

Figure 12: Exemplu de utilizare a funcției Map în metoda UpdateBoard

În concluzie, aplicarea principiilor programării funcționale în dezvoltarea jocului Minesweeper nu numai că a îmbunătățit claritatea și structura codului, dar a și facilitat o mai bună gestionare a complexității logicii jocului. Rezultatul este un cod mai curat.

6 Concluzii și referințe

Concluzii

Utilizarea programării logice și funcționale a permis o implementare eficientă a jocului Minesweeper, cu un cod curat și ușor de înțeles. Folosirea logicii pentru gestionarea regulilor jocului și a funcționalităților pure pentru manipularea datelor a condus la un cod modular și extensibil.

Conceptele funcționale, cum ar fi imutabilitatea datelor și funcțiile pure, au ajutat la reducerea erorilor și la creșterea stabilității jocului.

Proiectul oferă oportunități excelente pentru a învăța concepte legate de programarea logică și funcțională. Aceste ramuri ale programării sunt explorate și aplicate în detaliu în cadrul proiectului, permițând dezvoltarea și consolidarea abilităților în aceste domenii.

Link proiect github - `https://github.com/AlexE10/MinesweeperGame.git`

Referințe

1. Asist. drd. Nuțu Maria, *Programare logică și funcțională (Curs)*.