

# Jocului Vieții

## Proiectul 6

Capisizu Cosmin Louis  
IS1.1

January 9, 2022

## 1 Cerința

Proiectarea si implementarea unui sistem multi-agent pentru simularea jocului vieții, game of life. Se va defini si analiza cel puțin un joc de tip automat celular diferit de cel din programul demonstrativ pentru jocul vieții. Pentru jocul vieții se vor defini si experimenta si alte configurații inițiale, pe lângă cele predefinite in programul demonstrativ.

## 2 Prezentarea jocului

Jocul Vieții, cunoscut si sub numele de Viata, este un automat celular conceput de matematicianul britanic John Horton Conway in 1970. Este un joc fără jucători, ceea ce înseamna ca evoluția sa este determinata de starea sa inițiala, nefiind nevoie de alte informații.

Se interacționează cu Jocul Vieții creând o configurație inițială si observând cum evoluează. Este Turing complet si poate simula un constructor universal sau orice alta mașina Turing.

Universul Jocului Vieții este o rețea ortogonală infinită, bidimensională, de celule pătrate, fiecare dintre acestea fiind într-una dintre cele două stări posibile, vie sau moarta (sau populata si, respectiv, nepopulata). Fiecare celula interacționează cu cei opt vecini ai sai, care sunt celulele care sunt adiacente orizontal, vertical sau diagonal.

La fiecare pas in timp, au loc următoarele tranziții

- Orice celula vie cu mai puțin de doi vecini vii moare, ca prin subpopulare.
- Orice celula vie cu doi sau trei vecini vii trăiește in generația următoare.
- Orice celula vie cu mai mult de trei vecini vii moare, ca prin suprapopulare.
- Orice celula moarta cu exact trei vecini vii devine o celula vie, ca prin reproducere.

La fiecare pas in timp, au loc următoarele tranziții Aceste reguli, care compara comportamentul automatului cu viața reală, pot fi condensate in următoarele

- Orice celula vie cu doi sau trei vecini vii supraviețuiește.
- Orice celula moarta cu trei vecini vii devine o celula vie.

Toate celelalte celule vii mor in generația următoare. In mod similar, toate celelalte celule moarte rămân moarte.

Modelul inițial constituie sămânța sistemului. Prima generație este creata prin aplicarea regulilor de mai sus simultan la fiecare celula din sămânță, vie sau moarta; nașterile si decese au loc simultan, iar momentul discret in care se întâmplă acest lucru se numește uneori căpușa. Fiecare generație este o funcție pură a celei precedente. Regulile continua sa fie aplicate in mod repetat pentru a crea generații viitoare.

### 3 Problema studiata

În cadrul acestui proiect, problema constă în implementarea și stimularea agenților din cadrul jocului vieții. Această problemă are mai multe cerințe: Simularea sistemului multi-agent, analiza comportamentului agenților, și crearea unei modalități de alterare a parametrilor simulării.

### 4 Metoda de rezolvare folosită

Pentru implementarea sistemului, analiza datelor dar și alterarea parametrilor simulării s-au folosit mai multe tehnologii web.

Fundația aplicației este construită folosind framework-ul "React" construit de "Facebook", dar și alte tehnologii cum ar fi "TypeScript" și "Sass".

#### 4.1 Sistemul multi-agent


Acest sistem a fost construit folosind o clasă ce încorporează funcționalitatea dar și datele necesare simulării. Ca și structura de date ce stochează starea simulării s-a folosit matricea. Aceasta oferă un acces simplu și rapid asupra stării agenților.

#### 4.2 Parametri simulării

Pentru a permite utilizatorului să interacționeze cu sistemul multi-agent s-au implementat următoarele funcționalități: modificarea manuală a stării unei celule folosind mouseul, modificarea dimensiunii simulării folosind interfața grafică și modificarea și resetarea regulilor folosind editorul din simulare.

#### 4.3 Implementarea simulării

O **celula** poate fi vie sau mortă, aceste stări fiind reprezentate de 1 sau 0 logic. Tabloul ce este format din mulțimea celulelor este stocat într-o matrice. Această structură este implementată la rândul ei ca fiind un vector de vectorii de stări logice.



```
matrix: boolean[][];
```

Figure 1: Declararea matricii

**Starea celulelor** este modificată de către clasa din care fac parte, astfel celula în sine are doar funcționalitatea de a-și reține propria stare.

**Regulile de evoluție** sunt oferite de către un obiect ce deține o listă de reguli. Acest obiect este inițializat folosind setul de reguli de bază. Obiectul conține o listă de cerințe. Fiecare cerință are la rândul ei o listă de intervale și un set de condiții logice. Aceste condiții sunt folosite pentru a determina dacă un anumit scenariu se aplică. În cazul în care scenariul este confirmat ca fiind potrivit pentru starea actuală se folosește un alt set intern de condiții ce stabilesc starea următoare a celulei.

**Evoluția celulelor** este realizată de o metodă a clasei. Această funcție analizează datele ce sunt prezente în generația actuală și folosește instrucțiunile oferite de către serul de reguli pentru a genera următoare generație.

**Dimensiunea simulării** este predefinita dar se poate mari sau micșora folosind interfața grafica. Modificarea simulării nu duce la pierderea stării, excepție de la aceasta regula o face micșorarea simulării astfel încât celulele depășesc limita acesteia. Atât mărirea cat si micșorarea se face începând in coltul cel mai îndepărtat fata de origine.

#### 4.4 Implementarea interfeței grafice

Acest sistem pune accent pe implementarea modelului multi-agent, astfel interfața comanda si afișează starea simulării. Astfel arhitectura necesara implementării interfeței nu afectează sistemul simulat.

**Compatibilitatea dintre sistem si interfața grafica** se face folosind interfețele. Sistemul multi-agent implementează o interfața ce permite interfeței grafice sa acceseze resursele necesare acesteia.

```
export interface IMatrix {  
  ... width: number,  
  ... height: number,  
  ... matrix: boolean[][],  
  
  ... getCell(x: number, y: number): boolean,  
  ... inverCell(x: number, y: number): void,  
  ... resize(width: number, height: number): void  
  ... // getCloneInstance(): Matrix  
}
```

Figure 2: Interfața implementata de matrice

De asemenea interfața grafica are mai multe implementări ce implementează si ele la rândul lor o interfață.

**Interfața grafica** este implementata folosind doua sisteme diferita: componente web native si implementarea grafica folosind o librărie grafica specializata. Acestea doua implementări sunt abstractizate si pot fi interschimbate in timp real de către utilizator.

```
export interface ICMatrixRenderer {  
  ... matrix: IMatrix  
  ... onCellClickCallback(x: number, y: number): void  
}
```

Figure 3: Interfața implementata de sistemele de randare

## 5 Funcționarea programului pentru experimentare

## 6 Cazuri experimentale

## 7 Probleme întâlnite

Simulările multi-agent prezintă multe obstacole si probleme. Principalele dificultăți in cadrul acestui proiect sunt:

- Implementarea sistemului ce retine si modifica starea agenților. Acest sistem poate fi implementat folosind diferite structuri de date, cum ar fi: matricile sau listele. Principalul avantaj al matricii este accesul direct al oricărui agent dar si al vecinilor acestuia, un mare dezavantaj este faptul ca odată cu creșterea dimensiunii sumulării crește si mărimea matricii. Simularea folosind liste permite o scalare mai ușoara si o separare dintre numărul total de agenți posibil si mărimea mediului de simulare. Cel mai mare dezavantaj consta in faptul ca anumite operații sunt mult mai greu de implementat si necesita mult mai mulți pași de execuție. In cazul acestui proiect matricea a fost folosita ca si mijloc de stocare al datelor.
- Implementarea si unei interfețe grafice ce permite interacțiunea directa cu simularea. Interfața grafica si simularea necesita arhitecturi total diferite. Deci simularea sistemului este punctul de referință, interfața grafica afișează si modifica datele la cererea utilizatorului.

## 8 Modificările si îmbunătățirile necesare

Deși modelul de baza este implementat se pot adauga multe funcționalități. Simularea acestei probleme se poate face cu un număr foarte mare de parametri si condiții.

Multe componente deja implementate necesita anumite funcționalități pentru a opera mult mai satisfăcător. De asemenea parte de organisme ce pot fi create se poate implementa folosind o lista predefinita.

### 8.1 Modificări necesare

Din punct de vedere tehnic sunt necesare urmatoarele:

- Optimizarea funcțiilor ce generează datele următoarei generații.
- Implementarea a unei liste ce are rolul de a stoca doar celulele ce sun active. Aceasta structura poate fi folosita pentru eficientizarea procesului se afișare.
- Modificări asupra sistemelor de randare.

### 8.2 Potențiale îmbunătățiri

- Posibilitatea de a alege organisme ce pot fi simulate din cadrul unei liste.
- Implementarea unui sistem mai bun de randare.
- Implementarea unui sistem mai bun de reguli.
- Implementarea unui sistem de analiza a datelor simulării.
- Opțiunea de a salva starea curenta.
- Posibilitatea de a întoarce sistemul într-o stare anterioara.

## 9 Referințe

- [wiki — Conway's Game of Life](#)