UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

CellyGen - aplicație pentru explorarea automatelor celulare folosind algoritmi genetici

propusă de

Bita Mihai-Alexandru

Sesiunea: iunie/iulie, 2022

Coordonator științific

Asist. Dr. Croitoru Eugen

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

CellyGen - aplicație pentru explorarea automatelor celulare folosind algoritmi genetici

Bita Mihai-Alexandru

Sesiunea: iunie/iulie, 2022

Coordonator științific

Asist. Dr. Croitoru Eugen

Cuprins

Listă de figuri

Listă de tabele

Introducere

„CellyGen” este o aplicație de tip *desktop* ce permite lucrul cu și vizualizarea automatelor celulare. Mai mult decât atât, programul dispune de o funcționalitate aparte față de celelalte aplicații similare, și anume instrumentul ajutător de explorare pe baza algoritmilor genetici. Elementele de interfață grafică puse la dispoziția utilizatorului sunt menite să ofere atât simplitate în utilizare, cât și flexibilitate asupra configurării diverselor automate celulare, dar și al parametrilor folosiți de către algoritm.

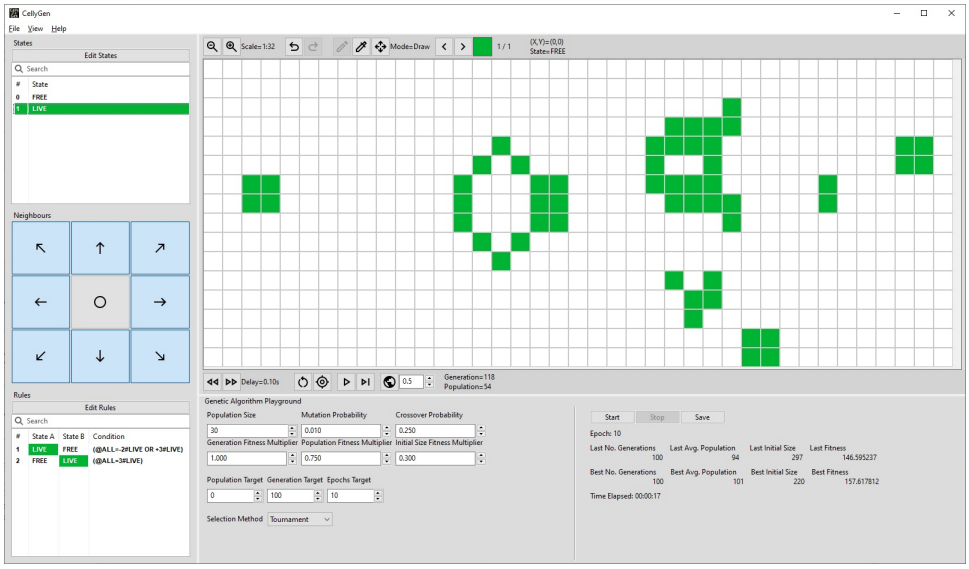

Figura 1: Aplicația CellyGen

Motivație

Una din principalele motivații pe care le-am avut pentru elaborarea acestei lucrări a fost înclinația către aceste două subiecte abordate: automatele celulare și algoritmii genetici. Am rămas plăcut surprins prima oară când am întâlnit conceptul de „automate celulare”, iar în timp, am încercat să implementez câteva modele cunoscute, uneori înregistrând *live* sau publicând propriile tutoriale pe canalul meu de YouTube

(fig. 2), ca mai apoi să urc proiectele pe GitHub[1].

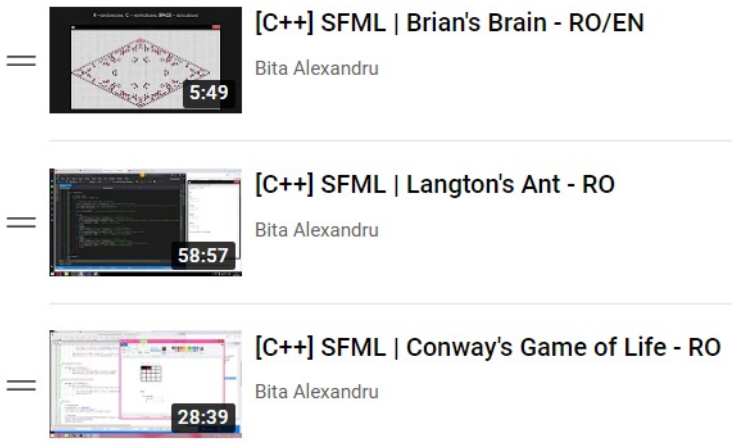

Figura 2: Videoclipuri proprii în care am implementat diverse automate celulare

Cât despre conceptul de „algoritmi genetici", primele interacțiuni care mi-au stârnit interesul au constat în vizionarea pe Youtube a unor prezentări de proiecte ce utilizau această paradigmă. Având oportunitatea de a studia mai profund algoritmii genetici în al doilea an de facultate, nu am ezitat să mă înscriu la acest curs opțional, iar pasiunea pentru acest subiect doar a crescut mai tare.

Astfel, am decis să încorporez aceste două subiecte într-o aplicație: **CellyGen** („Celly" - celule și „Gen" - genetic).

Un alt factor care a reprezentat un motiv important în alegerea acestei lucrări a fost dorința de a realiza un instrument pentru vizualizarea automatelor celulare ușor de folosit chiar și de către utilizatori nefamiliari cu acest concept. La momentul actual, cele mai populare aplicații în acest sens (e.g. Golly[2], CelLab[3]), deși foarte flexibile și performante, pot fi extrem de dificil de utilizat (fig. 3).

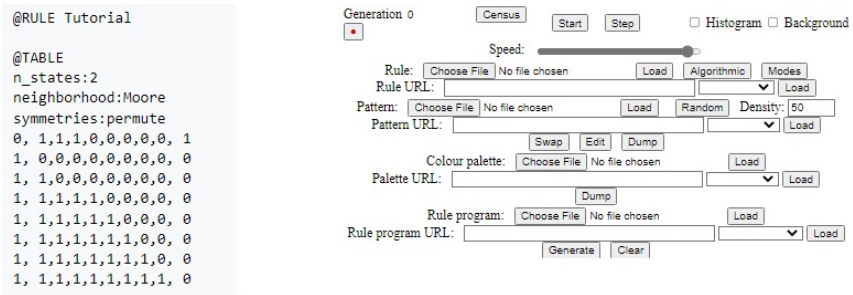

Figura 3: Definirea unei reguli în Golly (stânga) și o parte din interfața CelLab (dreapta)

[1] https://github.com/bita-alexandru/Old-Projects
[2] http://golly.sourceforge.net/
[3] https://www.fourmilab.ch/cellab/webca/

Din acest motiv, mi-am propus ca aplicația CellyGen să fie cât mai accesibilă cu putință, fără a afecta negativ funcțiile de bază, ci chiar să aducă în plus elementul de explorare folosind algoritmi genetici.

Gradul de noutate

La capitolul de „explorare al automatelor celulare", există o mulțime de programe care se raportează strict la această temă. Majoritatea, însă, „suferă" de aceeași problemă descrisă mai sus, și anume că sunt dificil de utilizat și chiar mai dificil de stăpânit. Un utilizator nou care ar dori să experimenteze puțin cu niște automate celulare simple se va regăsi ușor copleșit atât de multitudinea elementelor de interfață puse la dispoziție, cât și de documentația aferentă.

În ceea ce privește însă abordarea ambelor subiecte, cea mai similară lucrare pe care am reușit să o găsesc este cea a unor studenți de la Miami University[4]. Aici, scopul descris este acela de a observa dacă algoritmii evolutivi pot găsi automate celulare cât mai interesante vizual, altfel spus, de „clasă IV" [1]. Ceea ce diferențiază această lucrare de a mea este că studiul lor s-a limitat la un singur automat celular pentru care s-a folosit API-ul pus la dispoziție de Golly, așadar nu dispune de o aplicație grafică proprie în acest sens.

O altă lucrare asemănătoare o reprezintă *MergeLife*, „un algoritm genetic capabil să evolueze automate celulare care generează animații dinamice colorate în conformitate cu specificațiile utilizatorului"[5]. Aplicația prezentată dispune atât de un instrument propriu de vizualizare cât și de parametri configurabili în utilizarea algoritmului genetic. Totuși, o diferență majoră este aceea că MergeLife urmărește explorarea de noi automate celulare similare cu *Conway's Game of Life* (5.1.1), în timp ce CellyGen are ca scop explorarea unui automat celular predefinit.

În concluzie, în urma cercetării efectuate, am constat că la momentul actual nu există o aplicație care să bifeze cerințele setate în conceperea CellyGen, însă lucrările similare precum cele enumerate anterior pot constitui bune puncte de referință din prisma unor direcții viitoare de cercetare.

[4]Carter Hale, Owen Hichens, Eric Schonauer (2021) *Evolving Cellular Automata through Genetic Algorithms*, ECE Miami University

[5]Jeff Heaton (2018) *Evolving continuous cellular automata for aesthetic objectives*

Structura lucrării

Această lucrare este structurată pe cinci capitole, unde fiecare capitol descrie în mod progresiv elementele ce au stat la baza dezvoltării aplicației CellyGen.

Primul capitol, **Subiecte abordate**, introduce cele două concepte: automatele celulare și algoritmii genetici.

Al doilea capitol, **Tehnologii utilizate**, prezintă pe scurt câteva detalii tehnice și motivațiile în alegerea anumitor tehnologii.

Al treilea capitol, **Ghid de utilizare**, oferă o privire de ansamblu asupra elementelor de interfață grafică puse la dispoziție și explică funcționalitatea acestora.

Al patrulea capitol, **Implementare**, descrie în detaliu funcțiile cheie ale aplicației și algoritmii din spatele acestora.

Ultimul capitol, **Demonstrații**, prezintă o serie de teste și comparații privitor la corectitudine, performanță și eficiență.

Contribuții

Așa cum am menționat în capitolul de **Introducere**, ținta acestei lucrări a fost dezvoltarea unei aplicații ușor de înțeles și de folosit, cu elemente de interfață sugestive care să lege conceptul de automate celulare cu paradigma programării genetice. Pentru a împlini aceste cerințe am avut la dispoziție două opțiuni:

1. folosirea unui program *open-source* (e.g. Golly) și modificarea acestuia astfel încât să se potrivească cerințelor menționate

2. realizarea de la zero a unei astfel de soluții

Prima variantă prezenta avantajul că interfața grafică și funcțiile aferente unui automat celular erau deja implementate și extrem de bine optimizate, însă principalul dezavantaj se datora, în primul rând, documentației laborioase și în al doilea rând, faptului că simplificarea utilizării ar fi presupus mai multă muncă decât crearea unei aplicații de la zero.

Un argument împotriva celei de a doua variante a constat în necesitatea implementării unei interfețe grafice, însă cu ajutorul tehnologiilor din ziua de astăzi acest lucru nu a reprezentat un impediment major. În plus, această opțiune are avantajul major de a oferi mai multă flexibilitate din prisma funcționalității, mai exact a integrării algoritmului genetic care să permite utilizatorului să îl parametrizeze după bunul său plac. Acestea sunt motivele pentru care am ales varianta numărul 2.

Pentru încorporarea algoritmului genetic am avut de ales fie utilizarea unei biblioteci de programare genetică *open-source* (e.g. *openGA*[6], *GAlib*[7]), fie implementarea de la zero. Din considerente legate de flexibilitate, resurse utilizate, dar și din propria dorință de a pune în aplicare ce am învățat în facultate am ales să merg pe ultima variantă.

[6] `https://github.com/Arash-codedev/openGA`
[7] `http://lancet.mit.edu/ga/dist/`

Capitolul 1

Subiecte abordate

În secțiunile următoare voi prezenta pe câteva noțiuni de bază referitoare la subiecte care compun această lucrare fără a intra în detalii irelevante.

1.1 Automatele celulare

1.1.1 Prezentare generală

O bună definiție informală spune că un automat celular (prescurtat **AC**) poate fi privit ca un univers stilizat, unde spațiul este reprezentat printr-o grilă uniformă al cărei fiecare celule conține câteva bucăți de informație; timpul avansează în unități discrete, iar legile universului sunt exprimate printr-un tabel de căutare pe baza căruia, la fiecare pas, toate celulele își calculează noua stare în funcție de vecinii lor [2].

Pentru a înțelege mai bine, vom lua un exemplu simplu în care *universul* este reprezentat printr-un tabel de 2 rânduri și 2 coloane, iar *celulele* sunt pătrățelele care o formează. Fiecare celulă se află într-o anumită *stare*, lucru indicat de culoarea pe care o are. În acest exemplu vom avea două stări, starea **A** (desemnată prin culoarea neagră) și starea **B** (culoarea gri). *Vecinătatea* va fi alcătuită din celulele imediat adiacente de pe cele 4 direcții cardinale (nord, est, sud și vest), însă deoarece exemplul nostru este restricționat de dimensiunile mici pe care i le-am atribuit, fiecare celulă va avea doar 3 vecini, mai exact celelalte celule. Legile, sau mai bine spus *regulile* acestui univers sunt descrise tabelul 1.1.

Regula	Condiția
o celulă în starea **A** va trece în starea **B**	vecinătatea ei conține 2 celule în starea **B**
o celulă în starea **B** va trece în starea **A**	vecinătatea ei conține 2 celule în starea **A**

Tabela 1.1: Tabelul de reguli pentru AC-ul propus ca exemplu

Acum putem considera o configurație inițială (la timpul *t=0*) și rezultatul iterației următoare (*t=1*) precum în figura 1.1.

Figura 1.1: Configurația inițială (t=0) a AC-ului oferit ca exemplu (stânga) și rezultatul obținut la iterația imediat următoare (t=1) (dreapta)

Aplicând din nou regulile pentru următoarea iterație (t=2), vom ajunge la aceeași configurație precum cea inițială și observăm că acest AC va oscila între configurațiile descrise în figura precedentă. Schimbând dimensiunile tablei (sau uneori chiar și planul dimensional), lista de stări, reguli și vecini, AC-urile pot duce la comportamente extrem de complexe și interesante vizual.

1.1.2 Utilizări

Luând în calcul infinitatea de AC-uri posibile, nu este greu de închipuit că acestea pot fi folosite ca o „modalitate naturală de a studia sisteme fizice complexe” [2]. Un exemplu de astfel de sistem îl reprezintă „pigmentația scoicilor” [4] (fig. 1.2).

Figura 1.2: „*Conus textile*, prezintă un model de AC pe carcasă”[4]

1.2 Algoritmii genetici

1.2.1 Prezentare generală

Cel mai ușor mod de a înțelege felul în care funcționează algoritmii genetici (prescurtat **AG**) este să ne gândim la teoria evoluționistă a lui Charles Darwin[5] care menționează că având la dispoziție un mediu care poate adăposti un număr limitat de indivizi, iar instinctul primar al acestor indivizi este de a se reproduce, necesitatea unui proces de selecție devine inevitabilă. Selecția naturală favorizează indivizii care concurează cel mai eficient pentru resursele disponibile, altfel spus, cei mai adaptați în raport cu condițiile impuse. Acest fenomen poartă numele de *supraviețuirea celui mai adaptat* [6].

De aici putem extrage următoarele cuvinte cheie:

- *mediu* = cerința problemei
- *indivizi* = soluții candidat, de obicei reprezentați prin șiruri de biți (*gene*)
- *selecție* = procesul în care indivizi din *generația* actuală sunt selectați pentru reproducere
- *reproducere* = procesul în care indivizii selectați anterior (*părinți*) sunt cuplați și produc noi indivizi (*copii*) ce conțin gene de la ambii părinți
- *adaptare* = procesul în care un individ este supus diverșilor operatori genetici (selecția, reproducerea și *mutația*) în scopul de a produce rezultate cât mai bune; poate fi măsurată prin *fitness*-ul său
- *mutația* = fenomenul de alterare al genelor
- *fitness-ul* = valoarea numerică ce apreciază cât bine s-a adaptat un individ în generația actuală
- *generație* = o iterație a unui AG

1.2.2 Utilizări

Cea mai des întâlnită aplicație a algoritmilor genetici o reprezintă problemele de optimizare [7]. O privire de ansamblu asupra unor astfel de probleme rezolvate cu AG

ne arată că o parte din domeniile care utilizează cel mai bine AG sunt într-o strânsă legătură cu: biologia, informatica, ingineria, fizica și chiar și științele sociale [8].

1.3 Concluzii

În urma acestor prezentări, putem observa că ambele subiecte dispun într-o anumită măsură de o tematică a naturii: automatele celulare capabile să modeleze fenomene din natură [9] și algoritmii genetici care își au întreaga fundație în natura evolutivă a tuturor speciilor [10].

Tocmai acest simplu, dar în același timp complex fapt sugerează ipoteza unei combinații extrem de interesante între cele două și reprezintă un factor în plus în motivarea dezvoltării acestei lucrări.

Capitolul 2

Tehnologii utilizate

În cele ce urmează, voi enumera tehnologiile pe care le-am folosit în realizarea programului CellyGen și voi argumenta în mod concis motivația alegerilor făcute.

2.1 Mediul de dezvoltare

Am ales să folosesc Visual Studio[1], motivele principale fiind atât simplitatea cu care se configurează *framework-ul* cât și familiaritatea pe care o aveam deja cu acest IDE. Totodată, Visual Studio dispune de o gamă largă de funcții utile în eficientizarea scrierii de cod, precum *IntelliSense*[2].

2.2 Limbajul de programare și *framework*-ul

În alegerea limbajului de programare, am avut două cerințe majore:

- să fie un limbaj *high-level* compilat pentru a scrie cod în mod eficient care să ruleze cât mai rapid
- să existe un *framework* popular și relativ ușor de folosit care să permită lucrul cu acest limbaj de programare

Drept urmare, am ales să folosesc C++ împreună cu *wxWidgets*, un *framework* care permite programatorilor care permite dezvoltarea de aplicații grafice pentru diverse platforme desktop (Windows, Linux, macOS) folosind cod nativ[3].

[1] `https://visualstudio.microsoft.com/`
[2] `https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense`
[3] `https://www.wxwidgets.org/`

2.3 Alte tehnologii

Pentru crearea logoului (fig. 2.1) CellyGen am utilizat MS Paint, în realizarea manualului de utilizare am creat pagini web folosind HTML, iar ca instrument de control al versiunilor proiectului am folosit GitHub Desktop[4].

Figura 2.1: Logo CellyGen

2.4 Concluzii

Din ceea ce am descris în secțiunile de mai sus, pot spune că principalele mele motive în alegerile tehnologiilor utilizate se rezumă la:

- maturitate - în sensul în care o tehnologie are o vechime destul de mare și o comunitate de utilizatori remarcabilă
- familiaritate - am ales să lucrez cu tehnologii cu care mai lucrasem și înainte, ori cel puțin care să integreze astfel de tehnologii
- eficiență - un astfel de proiect complex trebuie să fie bine optimizat și gestionat

[4] https://docs.github.com/en/desktop

Capitolul 3

Ghid de utilizare

În acest capitol voi prezenta principalele elemente de interfață grafică pe care aplicația le pune la dispoziție și voi descrie sumar funcțiile pe care le îndeplinesc.

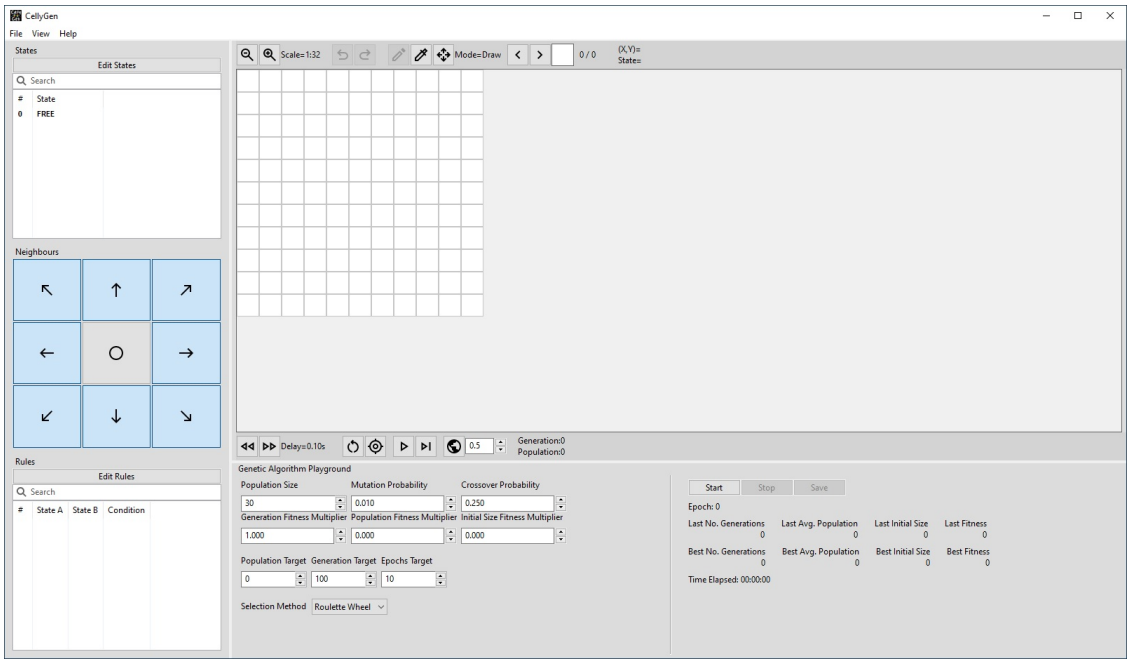

Figura 3.1: Interfața principală a aplicației CellyGen

3.1 Meniul principal

Meniul principal se poate observa în partea de sus a figurii 3.1 și este alcătuit din 3 meniuri:

- *File* ce are în vedere lucrul cu fișiere, respectiv închiderea aplicației:

– *Import* pentru a seta un AC conținut într-un fișier în format special creat de către funcția de *Export*

– *Export* pentru a salva local un fișier într-un format special ce conține AC-ul setat la acel moment

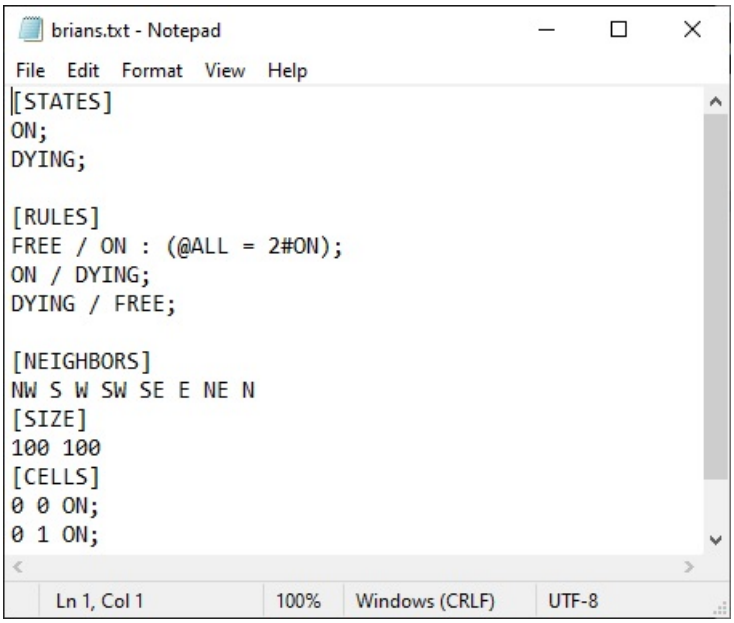

Figura 3.2: Cum arată un fișier creat prin funcția de Export

– *Exit*

- *View* unde se pot accesa editoarele, setările dimensiunii grilei sau se poate modifica aspectul interfeței:

  – *Change Dimensions* permite setarea dimensiunilor grilei

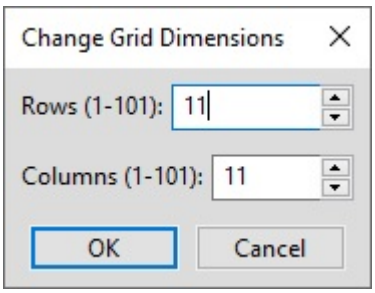

Figura 3.3: Setarea dimensiunilor grilei

  – *States Editor* deschide editorul de stări

  – *Rules Editor* deschide editorul de reguli

  – *Default Perspective* resetează aspectul interfeței

  - *Grid Perspective* schimbă aspectul interfeței astfel încât panoul de grilă să acopere majoritatea ferestrei
  - *Full Screen*

- *Help* ce pune la dispoziție manualul de utilizare redactat în limba engleză
  - *User Manual*

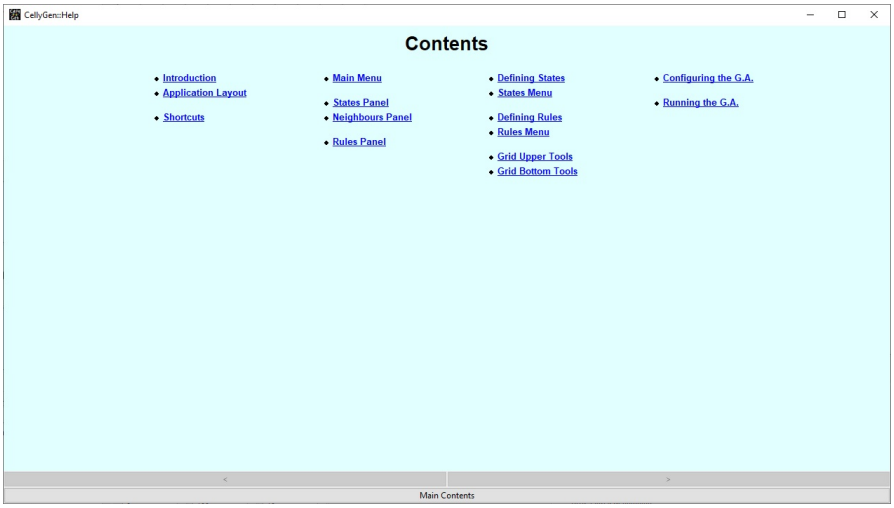

Figura 3.4: Vizualizarea manualului de utilizare în format HTML, redactat în limba engleză - inspirat după manualul de utilizare al aplicației Golly

3.2 Panoul de configurare al automatului celular

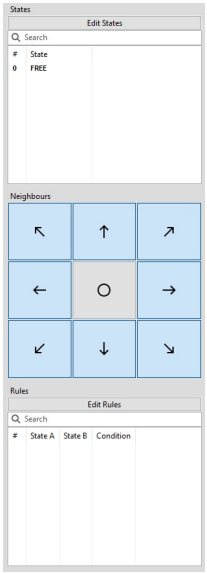

Figura 3.5: Panoul de configurare al automatului celular

Se situează în partea din stânga și, la rândul lui, este compus din trei panouri mai mici cu utilizări diferite (fig. 3.5).

3.2.1 Panoul de *stări*

Este primul și cel mai de sus panou (fig. 3.5) și permite definirea, ștergerea și selectarea stărilor unui AC.

3.2.1.1 Editorul de *stări*

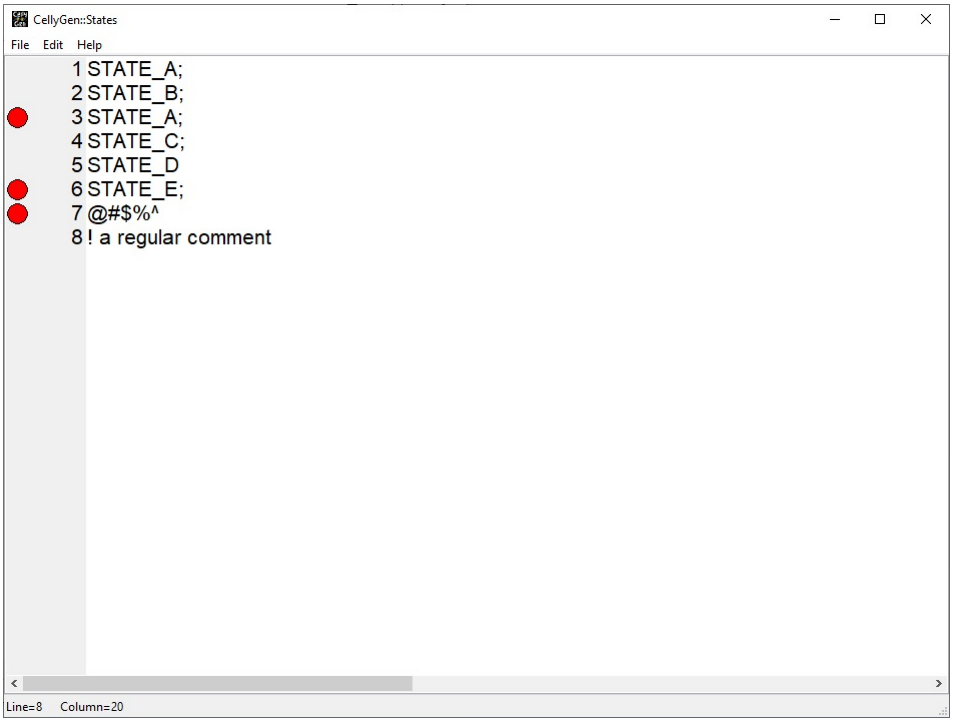

Figura 3.6: Editorul de stări

Pentru a putea defini stările unui AC, utilizatorul este nevoit să completeze caseta de text (e.g. fig. 3.6) folosind următoarea sintaxă:

```
<NUME_STARE_1>; <NUME_STARE_2>; . . . ; <NUME_STARE_N>;
```

Meniul prezintă următoarele funcții:

- *File*
  - *Import*, respectiv *Export*
  - *Save*, respectiv *Save & Close*
  - *Close*

- *Edit*
  - *Find*, respectiv *Replace*
  - *Next Mark*, respectiv *Previous Mark* care ghidează către stările marcate ca fiind invalide (cele marcate cu roșu în figura 3.6)
  - *Format* pentru a șterge spațiile și rândurile libere și pentru a trece caracterele în majusculă
- *Help*
  - *Defining states* pentru a deschide manualul de utilizare la capitolul care prezintă în detaliu modul în care se definesc stările unui AC

3.2.1.2 Lista de *stări*

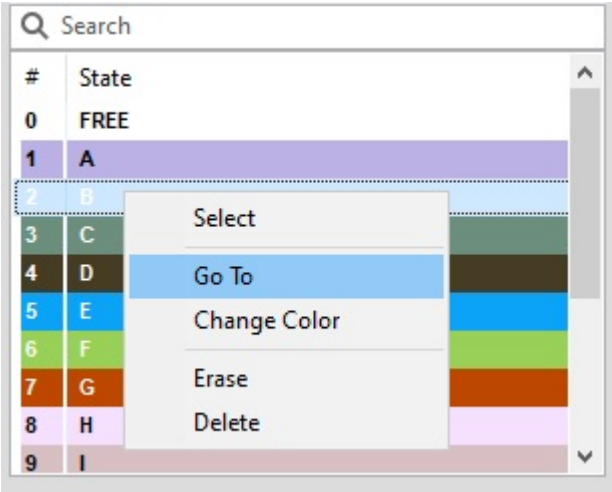

Figura 3.7: Lista de stări

Cuprinde stările definite în editorul de stări și oferă, în plus, următoarele opțiuni (fig. 3.7):

- Bara de căutare
- Un meniu contextual care apare atunci când se efectuează un click-dreapta:
  - *Select* setează starea pentru desenul pe grila
  - *Go To* deschide editorul de stări către linia cu starea selectată
  - *Change Color*
  - *Erase* șterge celulele aflate în această stare pe grila
  - *Delete* șterge starea din listă și celulele aflate în această stare pe grila

3.2.2 Panoul de *vecini*

Panoul de vecini este cel din mijlocul figurii 3.5 și este compus din nouă butoane, fiecare buton simbolizând o direcție cardinală (nord, nord-est, est, sud-est, sud, sud-vest, vest și nord-vest) sau celula în sine (centru). Când un buton este activat, direcția afișată de acesta este inclusă în definiția vecinătății unui AC.

3.2.3 Panoul de *reguli*

Ultimul panou și cel mai de jos din figura 3.1 permite definirea, modificarea și ștergerea regulilor unui AC.

3.2.3.1 Editorul de *reguli*

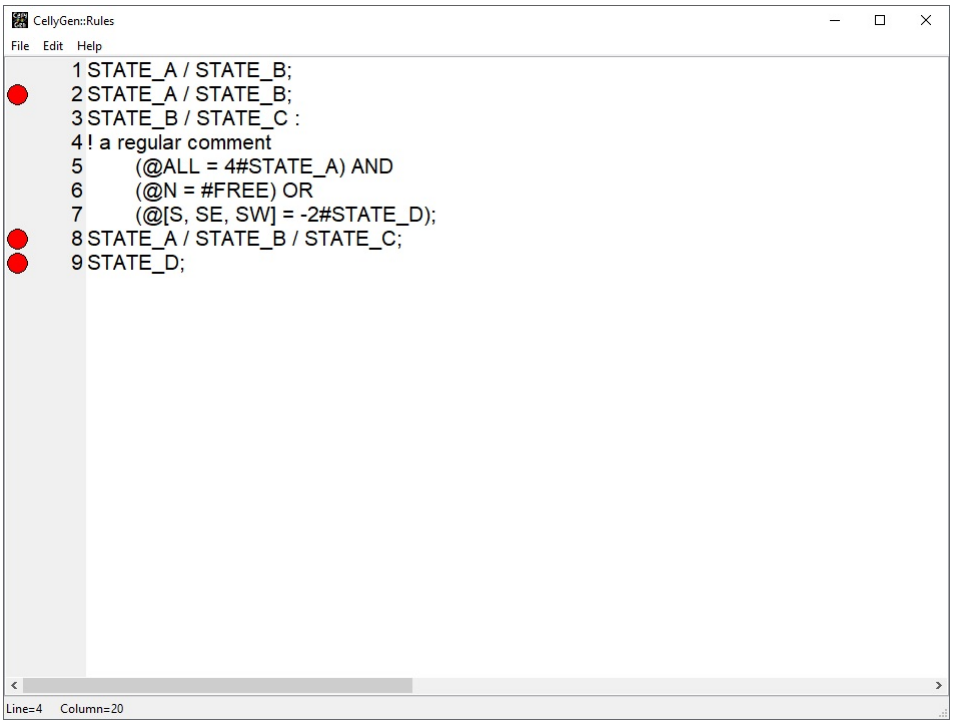

Figura 3.8: Editorul de reguli

Similar cu editorul de stări, însă diferit din punct de vedere al sintaxei necesare definirii regulilor unui AC, astfel că sintaxa permite următoarele variante:

- A / B;

  „O celulă în stare A va trece în stare B”

- A / B : **condiție**;

„O celulă în stare A va trece în stare B în cazul în care condiția **condiție** este împlinită”

O **condiție** este alcătuită din condiții mai mici:

- (**@vecinătate** = **semn număr-celule # stare-celulă**)
  - când **semn** lipsește, înseamnă că vecinătatea celulei evaluate trebuie să aibă exact **număr-celule** celule în stare **stare-celulă**) pentru a valida condiția
  - când **semn** = **+**, înseamnă că vecinătatea celulei evaluate trebuie să aibă mai multe de **număr-celule** celule în stare **stare-celulă**) pentru a valida condiția
  - când **semn** = **-**, înseamnă că vecinătatea celulei evaluate trebuie să aibă mai puțin de **număr-celule** celule în stare **stare-celulă**) pentru a valida condiția

O **vecinătate** poate fi definită astfel:

- printr-un singur vecin constituind o direcție cardinală (N, NE, E, SE, S, SW, W, NW) sau prin litera C, referindu-se la propria entitate
- printr-o mulțime de celule ilustrate în același mod dar respectând sintaxa: [**vecin-1**, **vecin-2**, ..., **vecin-n**]
- prin cuvântul **ALL** ce face referire la întreaga vecinătate definită în panoul de vecini

Aceste condiții pot fi înlănțuite folosind fie simbolul **AND**, fie **OR**.

Meniul principal ilustrează aceeași funcționalitate ca și cel prezent în editorul de stări.

3.2.3.2 Lista de *reguli*

Cuprinde regulile definite în editorul de reguli și oferă aceleași opțiuni (fig. 3.9) ca și cele prezentate în lista de stări:

- Bara de căutare
- Meniul contextual

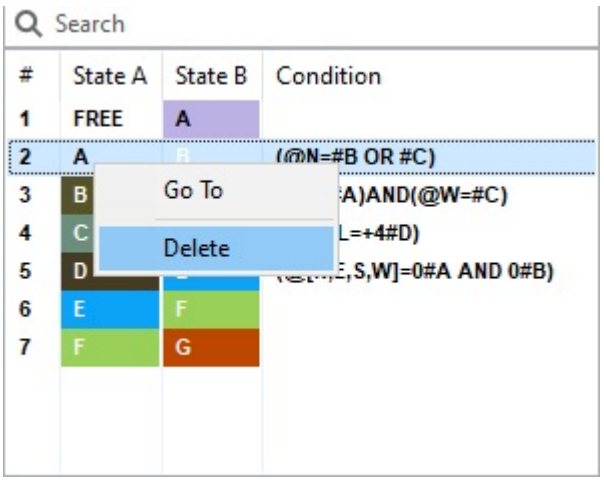

Figura 3.9: Lista de reguli

3.3 Panoul de *grilă*

Acest panou (fig. 3.3) conține grila grafică și instrumentele necesare manipulării acesteia.

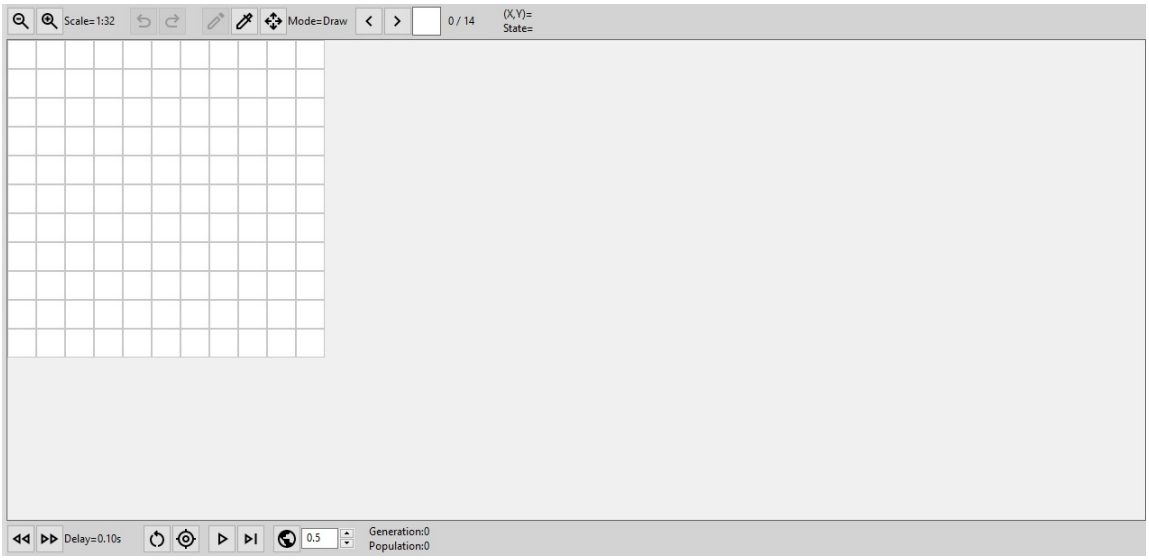

Figura 3.10: Panoul de grilă

3.3.1 Bara superioară de instrumente

Cu următoarele elemente, de la stânga spre dreapta:

- Butoanele de *Zoom In*, respectiv *Zoom Out* dedicate modificării mărimii celulelor pe grilă

- Butoanele de *Undo*, respectiv *Redo* ce permit utilizatorului să anuleze până la 50 de modificări consecutive făcute recent pe grilă
- Butoanele de *Draw*, *Pick*, respectiv *Move* pentru schimbarea modului de manipulare a grilei
- Butoanele de *Previous State*, respectiv *Next State* pentru selectarea stărilor din lista de stări

3.3.2 Grila

Grila reprezintă „universul" în care se desfășoară AC-ul. Utilizatorul poate desena sau șterge celule utilizând instrumentele din bara superioară și apoi vizualiza efectele aplicate folosind instrumentele din bara inferioară.

3.3.3 Bara inferioară de instrumente

Cu următoarele elemente, de la stânga spre dreapta:

- Butoanele de *Decrease Delay*, respectiv *Increase Delay* dedicate ajustării timpului de așteptare între iterațiile de actualizare a unui AC
- Butoanele de *Reset*, respectiv *Go To Center*
- Butoanele de *Play*, respectiv *Next Generation*
- Butonul de *Populate Randomly* și caseta aferentă pentru modificarea factorului de populare arbitrară

3.4 Panoul de utilizare al algoritmului genetic

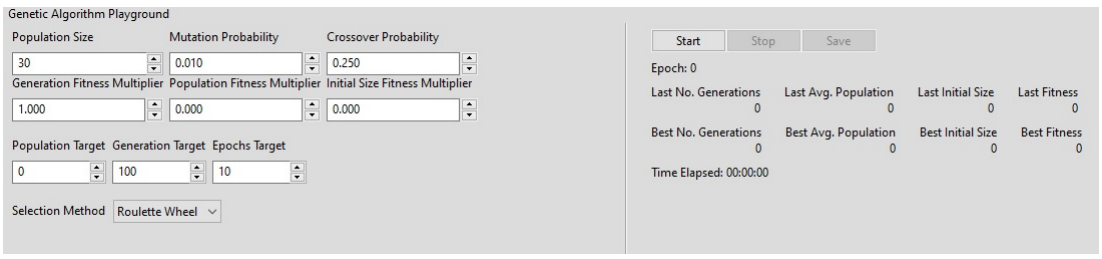

Figura 3.11: Panoul de utilizare al algoritmului genetic

Ultimul panou (fig. 3.11), oferă utilizatorului o serie de elemente de interfață pentru a configura parametrii algoritmului genetic și pentru a-l rula și a salva rezultatele.

3.4.1 Panoul de configurare

Aici se pot schimba următorii parametri:

- *Population Size*, numărul de cromozomi ce alcătuiesc o populație
- *Mutation Probability*, șansa ca o genă aparținând unui cromozom să fie alterată
- *Crossover Probability*, șansa ca un cuplu de cromozomi selectați pentru *crossover* să se reproducă
- *Generation Fitness Multipler*, cu cât acest număr este mai mare, cu atât mai mult AG-ul va prioritiza ca un cromozom să atingă un număr mare de generații într-un AC
- *Population Fitness Multiplier*, cu cât acest număr este mai mare, cu atât mai mult AG-ul va prioritiza ca un cromozom să atingă o medie a populației mare într-un AC
- *Initial Size Fitness Multiplier*, cu cât acest număr este mai mare, cu atât mai mult AG-ul va prioritiza ca un cromozom să aibă o configurație inițială cât mai mică într-un AC
- *Population Target*, mărimea populației la care un cromozom își poate opri simularea AC-ului
- *Generation Target*, numărul de generații într-un AC cromozom își poate opri simularea
- *Epoch Target*, numărul de iterații al AG-ului după care rularea va înceta
- *Selection Method*, funcția de selecție a cromozomilor pentru *crossover*

3.4.2 Panoul de rulare

În acest panou se pot observa datele cu privire la evoluția AG-ului (fig. 3.12), și anume:

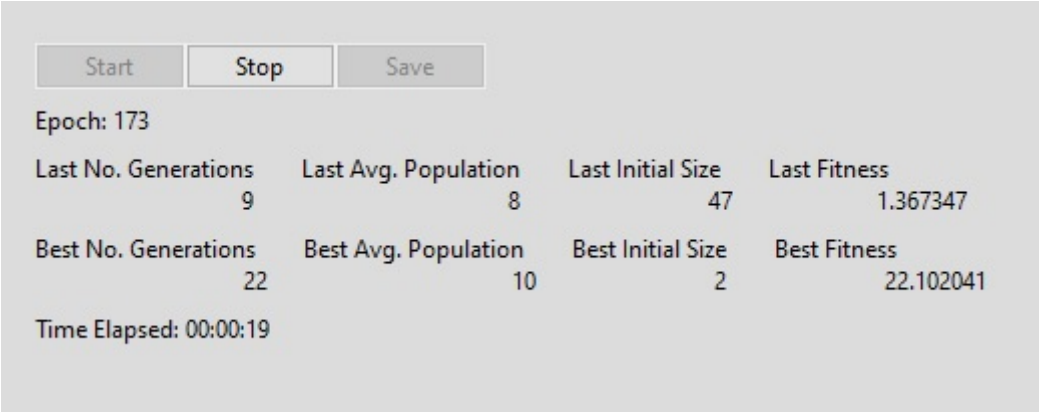

Figura 3.12: Panoul de rulare al algoritmului genetic în timpul unei execuții

- Butoanele de control
- *Epoch*, numărul iterației (al generației curente)
- Rezultatele celui mai bun cromozom al generației trecute
- Rezultatele celui mai bun cromozom din toate generațiile
- Timpul trecut de la apăsarea butonului de start

3.5 Concluzii

Este ușor de observat că, deși nu foarte modernă, interfața grafică a aplicației CellyGen este foarte sugestivă și bine pusă la punct. Mai mult decât atât, modul de definire a regulilor înfățișează un limbaj prietenos cu începătorii și ușor de învățat încă de la primele exemple din manualul de utilizare.

Din acest punct de vedere, consider că o bună parte din primul obiectiv setat a fost atins.

Capitolul 4

Implementare

Aici voi vorbi puțin mai tehnic despre unele detalii de implementare din cadrul dezvoltării CellyGen.

4.1 Automatul celular

4.1.1 Reprezentare

În reprezentarea automatului celular, cea mai dificilă provocare a fost arhitecturarea unei structuri de date pentru stocarea regulilor de tranziție. Soluția la care am convenit a fost un vector cu 4 dimensiuni:

1. o dimensiune pentru mini-condițiile înlănțuite prin operatorul AND
2. o dimensiune pentru mini-condițiile înlănțuite prin operatorul OR ce cuprinde lanțul de condiții anterioare
3. o dimensiune pentru condițiile înlănțuite prin operatorul AND ce cuprinde lanțul de condiții anterioare
4. o dimensiune pentru condițiile înlănțuite prin operatorul OR ce cuprinde lanțul de condiții anterioare

O astfel de mini-condiție este implementată sub forma unui vector de forma **vector<pair<pair<int, int>, string**, unde:

- **pair<int, int>** reprezintă perechea (număr de celule, semn), iar
- șirul de la final este numele stării din mini-condiție

La condițiile mari se mai ține cont de o valoare, și anume vecinătatea.

Mulțimea de celule este reprezentată mai simplu, printr-o structură de date de tip *cheie-valoare*, unde cheia reprezintă coordonatele (x, y) ale unei celule iar valoarea reprezinta perechea (stare, culoare), iar mulțimile de stări și vecini sunt doar reprezentate ca mulțimi neordonate de șiruri.

4.1.2 Rularea unui automat celular

Așadar, ținând cont de cele descrise mai sus, rularea unui automat celular poate fi sumarizat prin următorul pseudocod[1]:

```
while !no_updates_to_be_made() and !paused():
 for rule in rules:
  new_state = rule.new_state

   for cell in cells:
    if cell.state != rule.old_state: continue
    cell_nghbrs = get_nghbrs(cell)

    for cond_or in rule.conds_or:
     for cond_and in conds_or.conds_and:
      cond_nghbrs = cond_and.cond_nghbrs

       for mini_cond_or in conds_and.mini_conds_or:
        for mini_cond_and in mini_conds_or.mini_conds_and:
         cond_nr = mini_cond_and.cond_nr
         cond_sign = mini_cond_and.sign
         cond_state = mini_cond_and.state
         nghbrs = intersect(cell_nghbrs, cond_nghbrs)

         if valid(cell,nghbrs,cond_nr,cond_sign,cond_state):
          update_on_next_iteration(cell,new_state)
          skip_to_next_rule()
```

[1]functia **valid()** returnează *true* dacă celula dată ca parametru cuprinde în vecinătatea cerută exact atâtea celule în starea menționată de parametrii condiției, altfel *false*

4.2 Algoritmul genetic

4.2.1 Prezentare generală

Implementarea algoritmului genetic urmărește diagrama din figura 4.1, după schema prezentată în [11].

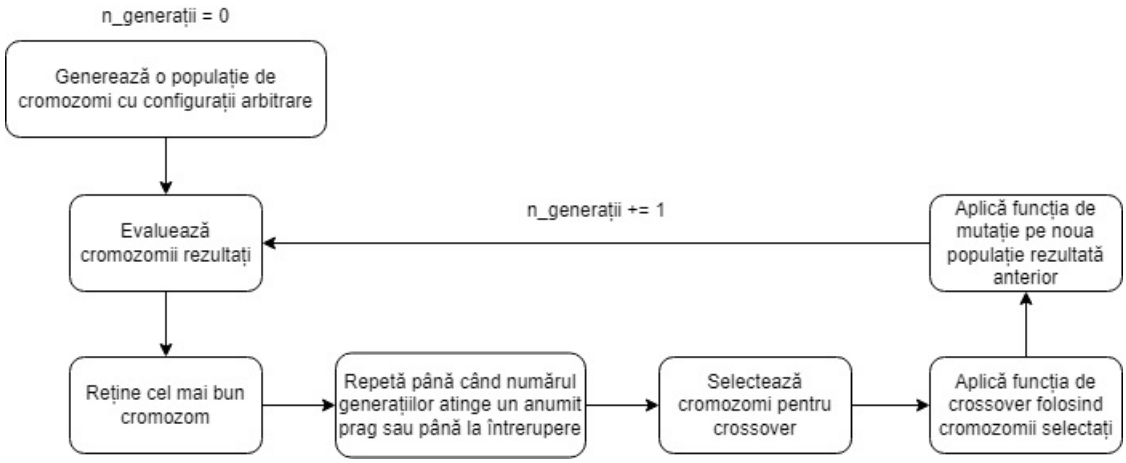

Figura 4.1: Diagrama algoritmului genetic

Pentru a reprezenta un cromozom, am folosit un vector unidimensional de numere de la 0 la N, unde N reprezintă numărul total de celule pe grilă. Fiecare element din vector poate avea valori întregi în intervalul $[0, S)$, unde S simbolizează numărul total de stări. O astfel de valoare va reflecta, de fapt, indexul unei stări în lista de stări.

În acest fel, pentru o grilă cu 2 rânduri și 3 coloane și un AC cu 4 stări, un cromozom cu valorile $[1, 0, 0, 3, 2, 0]$ va însemna în realitate, o matrice $[[1, 0, 0], [3, 2, 0]]$ care poate fi transpusă pe grilă prin simpla înlocuire a indecșilor cu numele stărilor corespunzătoare.

Așadar, pentru a evalua un cromozom, algoritmul va rula un AC care pornește de la configurația inițiala descrisă în cromozom[2], folosind metoda prezentată mai sus. În urma unei simulări complete sau în cazul în care algoritmul a atins un prag țintă ori a fost întrerupt, „calificativul" acordat cromozomului este exprimat prin funcția:

f(nOfGenerations, avgPopulation, initialSize) = 1 + (GFM*nOfGenerations + PFM*avgPopulation) * (1 - ISFM*initialSize/N), unde GFM, PFM și ISFM sunt parametrii de multiplicare setați în panoul de configurare al AG-ului și N reprezintă numărul tot de celule care alcătuiesc grila.

[2]O observație importantă este aceea că informația unui cromozom nu trebuie modificată la acest pas.

4.2.2 Funcția de selecție

Interfața grafică dispune de o serie de opțiuni când vine vorba de alegerea unei funcții de selecție:

1. *Roata norocului*

   O metodă prin care șansa unui cromozom de a fi selectat este proporțională cu fitness-ul său.

2. *Selecția bazată pe rang*

   Asemănătoare cu roata norocului, însă șansa unui cromozom de a fi selectat este proporțional cu poziția pe care s-a clasat ca fitness[3]. Acest lucru reduce o convergență prea rapidă prin faptul că se încearcă evitarea selectării predominante a celor mai buni cromozomi [12].

3. *Selecția turneu*

   Se organizează grupuri a câte 2 cromozomi diferiți, aleși arbitrar, iar cel cu scorul de fitness mai bun are o șansă mai mare de a fi ales.

4. *Selecția Steady-State*

   Această selecție presupune înlocuirea doar a unor indivizi[4], de regulă cei mai slab adaptați [13].

5. *Elitismul*

   Elitismul presupune conservarea celor mai bine adaptați 2 cromozomi. Asta înseamnă că nu vor suferi mutații în urma operatorilor genetici, deci cu fiecare generație soluția poate fi doar cel puțin la fel de bună.

6. *Selecția pur întâmplătoare*

   Selecția se face în totalitate arbitrat, fără a se ține cont de vreun factor. Motivul introducerii acestui mod de selecție s-a datorat interesului de a observa cum se descurcă un astfel de algoritm haotic într-un mediu care poate fi la rândul lui, haotic.

În cele ce urmează voi prezenta fie codul folosit, fie un pseudocod sumar în implementarea unor metode din cele enumerate mai sus.

[3] în urma unei sortări

[4] cei mai slabi 10% în cadrul acestei implementări

4.2.2.1 Roata norocului

```
vector<Chromosome> AlgorithmOutput::RouletteWheelSelection(vector<Chromosome>& population)
{
    double totalFitness = 0.0;
    for (int i = 0; i < popSize; i++) totalFitness += population[i].fitness;

    // calculate selection probability and cumulative selection probability
    // the probability of selection is proportionate to the fitness
    vector<double> q(popSize + 1);
    for (int i = 0; i < popSize && m_Running; i++)
    {
        double p = population[i].fitness / totalFitness;
        q[i + 1] = q[i] + p;
    }
```

Figura 4.2: Calculul probabilităților individuale și cumulative în selecția bazată pe *Roata norocului*

```
uniform_real_distribution<double> r01(0, 1);
vector<Chromosome> newPopulation;
int j = 0;
// "spin" the wheel until we have selected enough parents
while (j != popSize && m_Running)
{
    // iterate through each chromosome
    for (int i = 0; i < popSize && m_Running; i++)
    {
        double p = r01(generator);
        // select for the next generation
        if (q[j] < p && p <= q[j + 1])
        {
            Chromosome chromosome = population[i];
            chromosome.id = j++;

            newPopulation.push_back(chromosome);

            if (j == popSize) break;
        }
    }
}

return newPopulation;
```

Figura 4.3: „Învârtirea roții” și selectarea cromozomului rezultat pentru *crossover*

4.2.2.2 Selecția turneu

Poate fi rezumată prin următorul pseudocod:

```
selections = list()
n_selected = 0

while n_selected < population_size:
```

```
    # formam un turneu din 2 cromozomi alesi arbitrar
    chromosome_a = get_rand_chromosome(population)
    chromosome_b = get_rand_chromosome(population)

    # acelasi cromosom?
    while chromosome_a == chromosome_b:
      chromosome_b = get_rand_chromosome(population)

    # cromozomul mai "fit" are sanse mai mari sa fie ales
    p = real.random(0, 1)
    if p < 0.75:
      selection = get_best_of(chromosome_a, chromosome_b)
    # dar si cel mai slab are o mica sansa
    # pentru ca algoritmul sa nu convearga prea devreme
    else:
      selection = get_worst_of(chromosome_a, chromosome_b)

    parents.push_back(selection)
    n_selected += 1

return selections
```

4.2.3 Funcția de *crossover*

Folosind populația de cromozomi obținută la pasul de selecție, algoritmul va căuta să cupleze „părinți" 2 câte 2 și să producă noi indivizi. Probabilitatea ca doi părinți să se reproducă este p_c, definită de către utilizator în panoul de configurare al AG-ului.

În cazul în care nu are loc *crossover*-ul a 2 părinți, atunci ei vor fi copiați așa cum sunt, pentru a alcătui noua generație, altfel se va proceda astfel:

1. se creează un cromozom[5] cu informația copiată după primul părinte și un cromozom[6] cu informația copiată după al doilea părinte

2. se generează arbitrar 2 „puncte de tăiere"

[5]primul copil
[6]al doilea copil

3. genele primului copil, poziționate între cele 2 puncte generate anterior, vor fi înlocuite cu genele celui de-al doilea părinte

4. genele celui de-al doilea copil, poziționate între cele 2 puncte generate anterior, vor fi înlocuite cu genele primului părinte

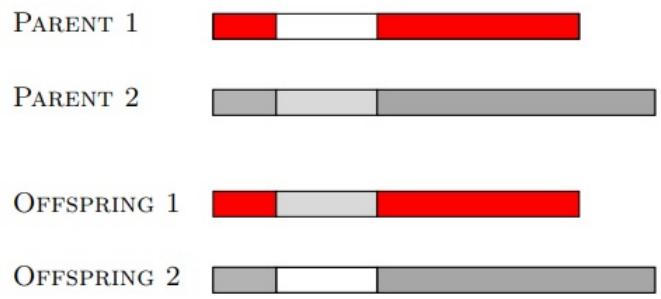

Figura 4.4: Cromozomii rezultați în urma unui *crossover* cu 2 puncte de tăiere, imagine preluată din [14]

Următorul pseudocod descrie implementarea funcției de *crossover*:[7]

```
new_population = list()
couple = pair()
for parent in parents:
    if couple.size() < 2:
      couple.push_back(parent)
    else:
      p = real.random(0, 1)

      if p < pc:
        # aplica crossover
        offsprings = couple
        x1, x2 = generate_cutpoints()
        for i in (x1, x2):
          swap(offsprings.first[i], offsprings.second[i])

        new_population.push_back(offsprings.first)
        new_population.push_back(offsprings.second)
```

[7] în corcondanță cu explicațiile anterioare și cu figura 4.4

```
        else:
            # copiaza parintii
            new_population.push_back(couple.first)
            new_population.push_back(couple.second)

        couple.clear()

if parents.size() % 2 == 1:
    # copiaza ultimul cromozom, ramas fara pereche
    new_population.push_back(parents.last())

return new_population
```

4.2.4 Funcția de mutație

Alterează cu o anumită probabilitate p_m, definită de către utilizator în panoul de configurare al AG-ului, genele cromozomilor, având grijă să nu modifice cromozomii elită.

```
void AlgorithmOutput::DoMutatiton(vector<Chromosome>& population)
{
    // select and alter genes to increase variety

    // keep count of the elites in this generation
    if (selectionMethod == "Elitism") SetEliteChromosomes(population);

    const int N = rows * cols;
    uniform_int_distribution<int> i0n(0, m_States.size() - 1);
    uniform_real_distribution<double> r01(0, 1);
    for (int i = 0; i < popSize && m_Running; i++)
    {
        // ignore chromosome if it's one of the elites
        if (eliteChromosomes.find(population[i].id) != eliteChromosomes.end())
            continue;

        // iterate through the chromosome's genes
        for (int j = 0; j < N && m_Running; j++)
        {
            double p = r01(generator);
            if (p <= pm)
            {
                // modify this gene
                int cellType = i0n(generator);
                population[i].initialPattern[j] = cellType;
            }
        }
    }
}
```

Figura 4.5: Alterarea genelor cu probabilitate p_m

4.3 Concluzii

O primă și nedisputabilă concluzie este: există loc de îmbunătățire în ceea ce privește arhitectura și performanța anumitor algoritmi. Rularea automatului celular are ca și complexitate de timp

$$O(k * n_1 * n_2 * n_3 * n_4)$$

unde k reprezintă numărul de celule peste care se va aplica o anumită regulă, n_1, n_2, n_3, n_4 marchează numărul de reguli imbricate prin operatorul **or**, apoi prin operatorul **and**, pe urmă condițiile interioare imbricate prin operatorul **or**, respectiv **and**, Acest lucru poate constitui un subiect interesant în îmbunătățirea performanței aplicației.

Tot legat de performanță, algoritmii prezentați în cadrul AG-ului pot fi optimizați la rândul lor, spre exemplu unele metode de selecție pot fi paralelizate[8], iar unele nu necesită evaluările tuturor cromozomilor decât strict la momentul selecției[9], sau nici măcar atunci[10]. Cu toate acestea, aplicația rulează bine în condiții normale și suficient de bine în condiții de stres.

Un alt lucru care poate fi menționat este faptul că, deși AG-ul poate fi personalizat în privința anumitor parametri generali, funcțiile de selecție sunt implementate cu valori rigide.

[8]Selecția Turneu

[9]Selecția Steady-State

[10]Selecția pur aleatorie

Capitolul 5

Demonstrații

În acest ultim capitol testa corectitudinea aplicației CellyGen din prisma rulării unei anumite configurații de AC, generată aleatoriu, în comparație cu Golly, iar mai apoi voi propune niște probleme pentru care AG-ul va trebui să găsească o soluție cât mai potrivită.

5.1 Rularea automatelor celulare

Am decis ca pentru aceste teste să folosesc AC-urile mele preferate: Conway's Game of Life, Brian's Brain și Langton's Ant.

5.1.1 *Conway's Game of Life*

5.1.1.1 Prezentare generală

Probabil cel mai cunoscut AC și cel care mi-a trezit interesul pentru automatele celulare, *Conway's Game of Life* [15] poate fi văzut ca o populație de organisme stilizate ce se dezvoltă, în timp, sub efectul constant al contracarării tendințelor de propagare și extincție [16].

5.1.1.2 Rezultate

Privind figurile 5.1 (**alb**=*moartă*, **gri**=*vie*) și 5.2 (**alb**=*vie*, **negru**=*moartă*), putem observa că rezultatele sunt identice.

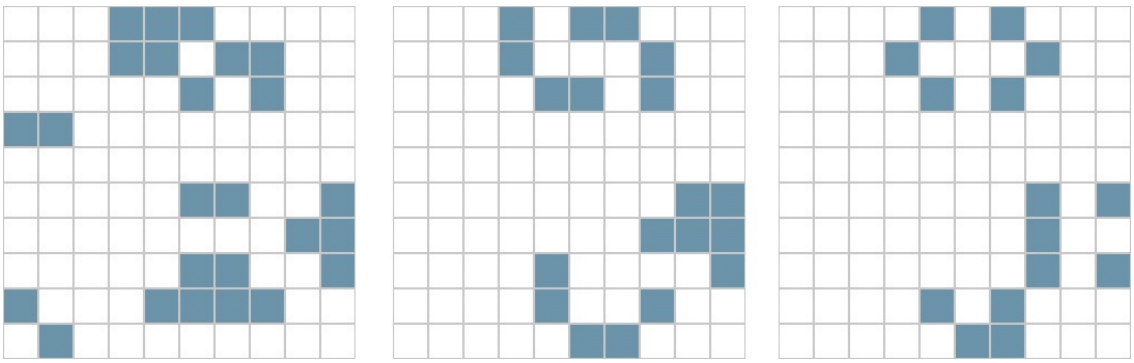

Figura 5.1: Evoluția iterațiilor unei configurații de Conway's Game of Life rulate pe CellyGen

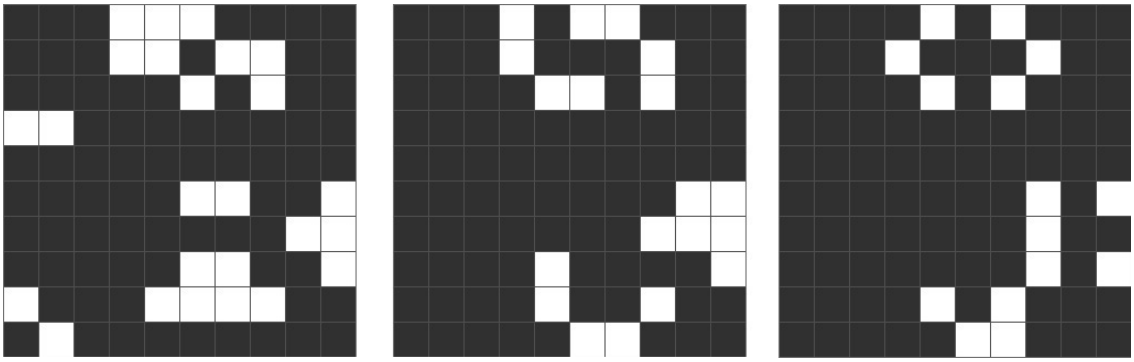

Figura 5.2: Evoluția iterațiilor unei configurații de Conway's Game of Life rulate pe Golly

5.1.2 *Brian's Brain*

5.1.2.1 Prezentare generală

Brian's Brain[1] este un AC similar cu cel prezentat anterior, însă pe lângă cele două stări definitorii (PORNIT și OPRIT), mai prezintă și o stare intermediară (PE MOARTE) care schimbă dramatic situația lucrurilor.

5.1.2.2 Rezultate

Privind figurile 5.3 (**alb**=*oprită*, **albastru**=*pornită*, **mov**=*pe moarte*) și 5.4 (**negru**=*oprită*, **roșu**=*pornită*, **galben**=*pe moarte*), putem observa că rezultatele sunt identice.

[1] `http://www.msevans.com/automata/briansbrain.html`

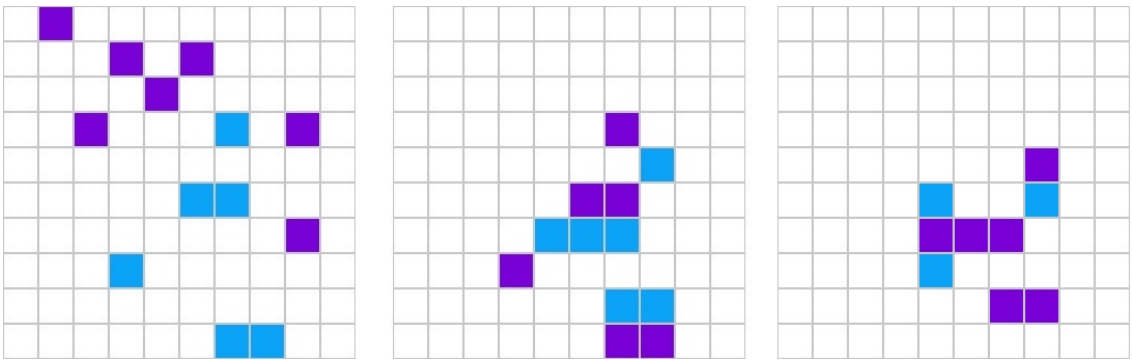

Figura 5.3: Evoluția iterațiilor unei configurații de Brian's Brain rulate pe CellyGen

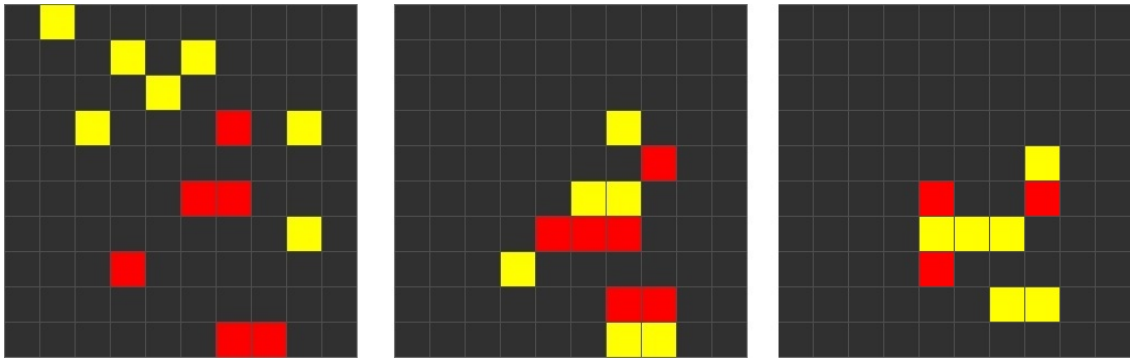

Figura 5.4: Evoluția iterațiilor unei configurații de Brian's Brain rulate pe Golly

5.1.3 *Langton's Ant*

5.1.3.1 Prezentare generală

Langton's Ant este un AC mai diferit față de anterioarele și „urmărește evoluția unei colonii de insecte" [17].

5.1.3.2 Rezultate

Privind figurile 5.5 (**alb**=*căsuță albă*, **mov**=*căsuță neagră*, **gri**=*furnică albă cu fața spre est*, **galben**=*furnică neagră cu fața spre nord*, **verde**=*furnică neagră cu fața spre est*) și 5.6 (aici, reprezentările folosite sunt destul destul de sugestive), putem observa că rezultatele sunt identice.

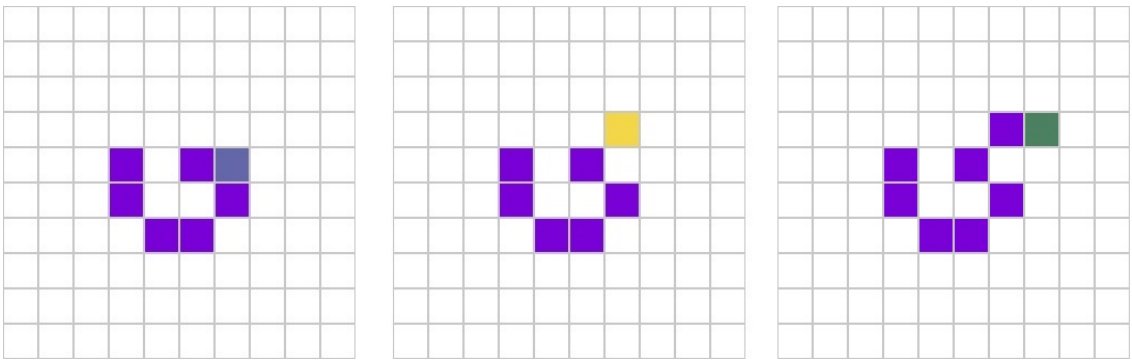

Figura 5.5: Evoluția iterațiilor unei configurații de Langton's Ant rulate pe CellyGen

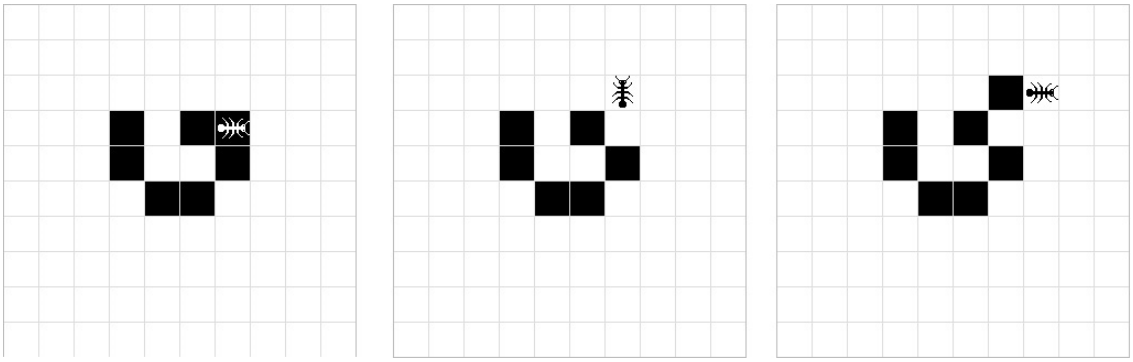

Figura 5.6: Evoluția iterațiilor unei configurații de Langton's Ant rulate pe Golly

5.1.4 Concluzii

Comparând figurile de mai sus și analizând rezultatele, putem concluziona că implementarea AC-ului este corectă și flexibilă din punct de vedere funcțional.

5.2 Rularea algoritmului genetic

În secțiunile ce urmează, voi enunța o problemă în funcție de AC, voi rula AG-ul conform constrângerilor descrise și la final, voi afișa și analiza pe scurt rezultatele obținute.

AG-ul va rula de 3 de ori pe fiecare metodă de selecție, cu următorii parametri:

- Dimensiunea grilei: 7×7
- Mărimea populației: 100
- p_m: 0.01
- p_c: 0.25
- *Numărul de iterații*: 100

5.2.1 *Conway's Game of Life*

5.2.1.1 Context

Cerința acestei probleme va consta în găsirea unei configurații inițiale care să ducă la un număr de generații și o medie a populației cât mai mari, fără mențiuni în ce privește numărul de celule inițiale.

5.2.1.2 Rezultate

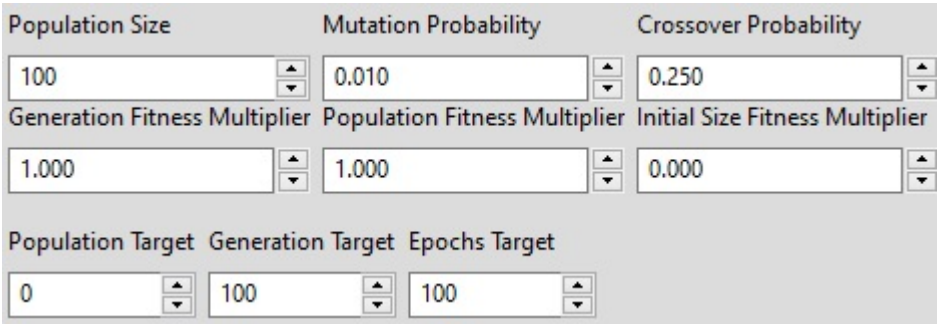

Figura 5.7: Parametrii AG-ului în problema cu Conway's Game of Life

Funcția de selecție	Fitness minim	Fitness mediu	Fitness maxim	Timp mediu
Roulette Wheel	115.69	116.1	116.77	26.3s
Rank	115.81	116.08	116.53	29.6
Steady-State	115.59	115.94	116.63	21.3s
Tournament	115.48	115.69	115.85	4s
Elitism	116	116.25	116.51	42s
Random	115.94	117.43	120.32	24s

Tabela 5.1: Tabelul de rezultate obținute de AG pe Conway's Game of Life

5.2.2 *Brian's Brain*

5.2.2.1 Context

Cerința acestei probleme va consta în găsirea unei configurații inițiale care să ducă la un număr de generații cât mai mari, dar cu numărul de celule inițiale cât mai mic.

5.2.2.2 Rezultate

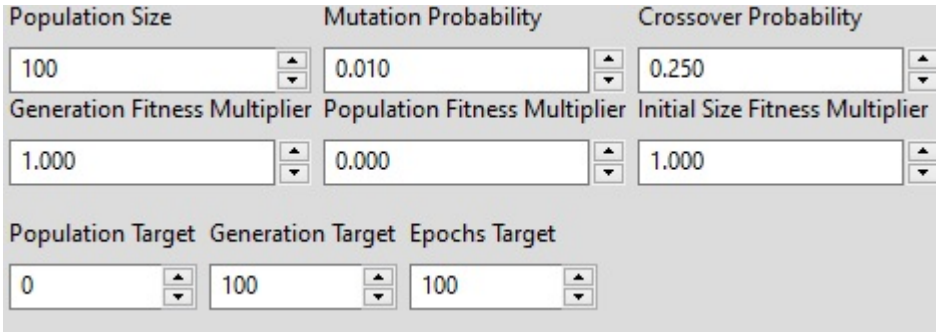

Figura 5.8: Parametrii AG-ului în problema cu Brian's Brain

Funcția de selecție	**Fitness minim**	**Fitness mediu**	**Fitness maxim**	**Timp mediu**
Roulette Wheel	45.89	54.74	60.18	3.3s
Rank	18.36	39.44	80.59	4.3s
Steady-State	15.91	17.96	19.34	3.3s
Tournament	14.71	15.89	17.73	1.6s
Elitism	70.38	76.50	86.71	9s
Random	18.06	51.58	68.34	3.6s

Tabela 5.2: Tabelul de rezultate obținute de AG pe Brian's Brain

5.2.3 *Langton's Ant*

5.2.3.1 Context

Cerința acestei probleme va consta în găsirea unei configurații inițiale care să ducă la un număr de generații cât mai mare, fără alte mențiuni.

5.2.3.2 Rezultate

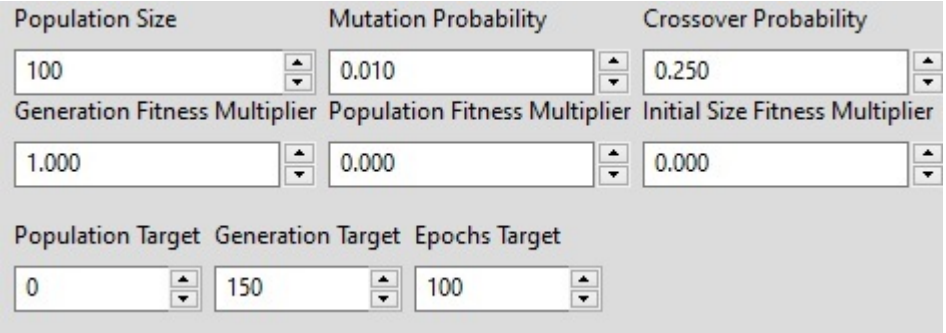

Figura 5.9: Parametrii AG-ului în problema cu Langton's Ant

Funcția de selecție	**Fitness minim**	**Fitness mediu**	**Fitness maxim**	**Timp mediu**
Roulette Wheel	91	95.33	100	15s
Rank	97	108.66	121	10s
Steady-State	87	93.33	104	10.3s
Tournament	95	96.33	112	9.6s
Elitism	114	125.33	148	17.3s
Random	93	97.66	102	14s

Tabela 5.3: Tabelul de rezultate obținute de AG pe Langton's Ant

5.2.4 Concluzii

Încă de la prima vedere a rezultatelor afișate mai sus, putem sesiza polii opuși ai consistenței dintre funcția de selecție arbitrară și cea bazată pe elitism, cum de altfel era de așteptat. Deși cea dintâi a reușit în prima problema să obțină cel mai bun rezultat (tab. 5.1), cea din urmă a oferit cele mai bune rezultate per întreg ansamblul de probleme (primele locuri în tab. 5.2 și 5.3), în schimbul performanței ca timp. Tot la capitolul performanță, observăm cum selecția turneu a oferit cele mai rapide soluții în fiecare dintre teste, însă nu și cele mai bune (în special în tab. 5.2). Restul funcțiilor au avut randamente oarecum similare între ele și nu au prezentat nimic notabil, cel puțin nu în cadrul acestor probleme.

În mod evident, rezultatele ar putea arăta cu totul diferit dacă am rula aceleași teste dar la parametri diferiți, cu atât mai mult dacă am schimba întru totul enunțurile

problemelor. Cert este că oricum ar arăta o problemă în acest sens și orice automat celular am dori să explorăm, o vom putea face simplu, utilizând acest instrument.

Concluzii

Așadar, în urma celor prezentate, putem concluziona că aplicația CellyGen este întocmai ce a promis să fie în partea de **Introducere**:

1. un program de vizualizare al automatelor celulare ușor de folosit, cu o interfață grafică *user-friendly*, performant și flexibil

2. un instrument pentru explorarea automatelor celulare folosind algoritmi genetici, cu un mecanism de setare al parametrilor și al funcției de selecție

Există o mulțime de direcții de îmbunătățire pe care aș dori să le fac într-o bună eventualitate, spre exemplu, referindu-mă înapoi la lucrările asemănătoare prezentate în secțiunea **Grad de noutate**, ar fi foarte interesantă implementarea unei funcționalități de explorare de noi reguli și nu doar de configurații de celule. Chiar și algoritmul genetic ar putea pune la dispoziție mai mulți parametri, sau mai important, de o funcție de fitness care să poată fi personalizată mai mult decât doar prin unor casete de numere. Alte îmbunătățiri ar consta în principiu în aspectele ce țin de optimizare.

În final, pot spune cu sinceritate că lucrarea pe care am abordat-o a fost una extrem de ambițioasă și plină de provocări, însă mă declar mai mult decât mulțumit de rodul muncii pe care am prestat-o. Faptul că am reușit să combin două concepte de o așa natură complexă precum automatele celulare și algoritmii genetici și să le încorporez într-o aplicație *desktop* concepută în așa fel încât să poată fi folosită de oricine, pentru mine este o realizare satisfăcătoare.

Bibliografie

[1] Stephen Wolfram (2002) *A New Kind of Science*, pp. 231-249

[2] Tullio Ceccherini-Silberstein, Michel Coornaert (2010). *Cellular Automata and Groups*, p. 6

[3] Tullio Ceccherini-Silberstein, Michel Coornaert (2010). *Cellular Automata and Groups*

[4] Stephen Coombes (2009) *The Geometry and Pigmentation of Seashell*, pp. 3-4

[5] Charles Darwin (1859) *On the Origin of Species*

[6] A.E. Eiben, J.E. Smith (2015) *Introduction to Evolutionary Computing*, pp. 15-16

[7] Melanie Mitchell (1996) *An Introduction to Genetic Algorithms*, p. 7

[8] David E. Goldberg (1989) *Genetic Algorithms in Search Optimization & Machine Learning*, pp. 125-129

[9] Stephen Wolfram (2002) „Implications for Everyday Systems" in *A New Kind of Science*, pp. 363-432

[10] David E. Goldberg (1989) *Genetic Algorithms in Search Optimization & Machine Learning*, pp. 1-2

[11] Melanie Mitchell (1996) *An Introduction to Genetic Algorithms*, pp. 8-9

[12] Melanie Mitchell (1996) *An Introduction to Genetic Algorithms*, p. 127

[13] A.E. Eiben, J.E. Smith (2015) *Introduction to Evolutionary Computing*, p. 80

[14] Riccardo Poli, William B. Langdon, Nicholas Freitag McPhee (2008) *A Field Guide to Genetic Programming*, p. 65

[15] Martin Gardner (1970) *The fantastic combinations of John Conway's new solitaire game "life"*

[16] Tommaso Toffoli, Norman Margolus (1987) *Cellular Automata Machines: A New Environment for Modeling*, p. 20

[17] Christopher G. Langton (1986) „Studying artificial life with cellular automata" in *Physica D: Nonlinear Phenomen*, pp. 134-136