Ifrim Andreea-Carmen

Grupa 322CB

**Tema nr. 3**

**I.Introducere**

Pentru tema nr. 3, am avut de implementat un algoritm care simuleaza Conway’s Game of Life. Acesta este un joc în care, setand o configurație initială, utilizatorul poate urmări evoluția unor celule, comportamentul acestora bazandu-se pe următoarele reguli:

* O celulă vie cu mai puțin de doi vecini vii moare
* O celula vie cu mai mult de trei vecini vii moare
* O celulă vie cu doi sau trei vecini traieste si in generația următoare.
* O celulă moartă cu exact trei vecini vii devine o celulă vie.

**II. Decizii de design:**

Pentru implementarea acestui algoritm, am ales paternul Singleton. Acesta este folosit în programarea orientată pe obiecte pentru a asigura faptul că o clasă nu poate avea decât o singură instanță, iar accesarea acesteia se face cu ajutorul unei funcţii capabile să asigure acest lucru.

Ținand cont de cerințele problemei, paternul Singleton a reprezentat o alegere potrivită deoarece:

* Asigură acces controlat la instanța clasei realizat prin încapsulare;
* Instanțele claselor sunt create doar atunci cand este nevoie de ele (instanțe ale clasei Cell sunt create doar în construirea matricei cells, în inițializarea jocului);
* Asigură faptul că o clasă nu poate avea decat o singură instanță (ne dorim o singură instanță a clasei Game deoarece avem un singur joc care va rula și o singură instanță a clasei Frame, deoarece totul se va desfășura într-o singură fereastră);
* Obiectele instanțiate ocupă un spațiu de memorie de mare (instanțele claselor Game și Frame) și vrem să evităm instanțierea necontrolată a acestora;
* Nu se confundă cu variabilele globale;
* Poate fi modificat cu ușurință astfel încat sa permită o instanțiere multiplă acolo unde este nevoie (vom avea nevoie de mai multe celule, deci de mai multe obiecte de tip Cell);
* Implementarea sa nu prezintă un grad de dificultate ridicat;
* Codul rezultat este unul ușor de citit și interpretat.

**III. Implementare:**

Pentru a implementa Conway’s Game of Life, am creat:

* Clasa Cell;
* Clasa Game;
* Clasa Frame;
* Clasa GoL (Game of Life).

**Clasa Cell –** obiectele de tip Cell vor reprezenta celulele jocului. Aceasta va fi instanțiată de mai multe ori în cadrul inițializării jocului. Cu toate acestea, pentru a controla instanțierea sa, aceasta respectă paternul singleton și, în consecință, are o variabila statică Cell cell și o metodă static getCell prin care se vor obține instanțele sale;

De asemenea, aceasta clasă are ca attribute 2 coordonate de tip int, x și y ce vor reprezenta poziția pe harta de joc și vor fi specifice fiecărui obiect de tip cell, o variabilă statică de tip int size, în care se va reține dimensiunea unei celule, o variabilă boolean alive, ce va reprezenta starea celulei în frame-ul curent (true dacă celula este vie, false dacă este moartă) și o variabilă boolean nextRound în care se va reține starea celulei după update.

Metodele clasei:

* Void setAlive(Boolean alive) – setează starea celulei, atribuind o valoare caracteristicii alive, specifică fiecărei celule
* Boolean is Alive () – returnează starea curentă a celulei;
* Void setNextRound(Boolean alive) – setează starea celulei pentru frame-ul următor;
* Void nextRound() – se atribuie variabilei alive valoarea variabilei nextRound; această metodă va fi folosită atunci cand se va face update, iar celula își va schimba starea cu cea determinată în urma inspectării “regulilor de joc”
* Static Cell getCell(int x, int y) – returnează o instanță a clasei Cell
* Void drawCell – desenează celula

**Clasa Game** – în această clasă se va simula jocul, Clasa implementează interfețele MouseListener, MouseMotionListener și KeyListener pentru a permite introducerea datelor, cu ajutorul tastaturii și a mouse-ului, prin interfața grafică

Această clasă respecta principiile paternului Singleton și, prin urmare, are un constructor privat, o variabilă statică Game game și o metodă statică getGame prin care se obține o instanță de tip Game.

In constructor, se construiește matricea de celule cells și se instanțiază fiecare celulă.

În plus, clasa Game și obiectele de acest tip au și următoarele attribute:

* Cells[][] cells – reprezintă matricea de celule ale jocului; aceasta va fi inițializată în constructorul clasei;
* Int width și int height – acestea vor fi dimensiunile matricei, reprezentand numărul de celule din joc (sunt determinate în funcție de mărimea hărții și dimensiunea unei celule)
* Int nrGeneration – reprezintă numărul generației curente
* Int go – reprezintă starea jocului:
  + -1 pauză;
  + 0 jocul rulează;
  + 1 jocul se termină (atunci cand se ajunge într-o configurație de tip block).
* Int button – variabilă ce determină dacă mouse-ul este apăsat sau nu
* Static Boolean border – variabilă specifică jocului ce determină dacă harta acestuia se va implementa cu graniță” sau “fără graniță” (true – cu graniță, false – fără graniță)

Metodele clasei:

* Game getGame() – returnează o instanță a clasei Game
* Void Draw(Graphics g) – desenează “planșa de joc” cu fiecare celulă

Void update() – se realizează un update al stării celulelor; dacă jocul se află în derulare, atunci pentru fiecare celulă se numără vecinii vii și se modifică variabila nextRound specifică fiecăreia în funcție de numărul acestora. Vecinii fiecărei celule se numără diferit în funcție de cele 2 moduri ale jocului (cu graniță și fără graniță). Dacă harta este fără graniță, atunci se consideră marginea din dreapta a acesteia legată de cea din stanga, respectiv marginea superioară de cea inferioară, astfel că o celulă aflată în partea superioară a hărții va putea fi considerată vecină cu una aflată în partea inferior și la fel pentru partea stanga și dreaptă.

Apoi, după ce s-a determinat pentru toate celulele în ce stare trebuie să treacă, se apelează metoda nextRound().

Dacă noua configurație este aceeași cu cea de dinainte înseamnă că s-a ajuns la o generație de tip block și jocul se oprește.

* Metoda keyPressed permite stabilirea stării în care se află jocul prin apăsarea tastei SPACE.
* Metodele mouseMoved, mouseDragged și mousePressed permit stabilirea grafică a configurației inițiale. Prin apăsare unei celule, acesta își modifică starea din vie în moartă, respectiv din moartă în vie. Acest lucru se întamplă doar dacă jocul este pe pauză.

**Clasa Frame** - aceasta extinde clasa JFrame și va reprezenta interfața grafică, construind o ferestră în care se va desfășura jocul. Aceasta va fi maximizată pe tot ecranul și se va trece la următorul frame prin metoda update(float dt) după un timp stabilit.(mai mare decat UPDATETIME). Tot aici se creează 2 ferestre de input prin care se vor seta dimensiunea celulei și modul jocului ( cu graniță sau fără).