Dumitru Mihai-Valentin 325 CB

Tema 3 POO – Conway's „Game of Life”

1. Introducere

Tema implementează în Java, folosind paradigma POO, „jocul” „Game of Life”. Aplicația nu are interfață grafică, ci comunică cu utilizatorul prin interemediul argumentelor din linia de comandă și prin citire/scriere în fișiere.

Tema implementează două modele diferite de joc: una cu margini, în care acțiunea jocului se desfășoară într-un spațiu dreptunghiular definit inițial și care nu poate fi schimbat și o a doua implementare care permite expansiunea spațiului de joc la infinit, în orice direcție.

Proiectul a fost dezvoltat folosind principiile TDD.

2. Design Choices

Design-ul conține două clase majore: clasa „Game” care reprezintă un joc activ și clasa „GameField” care reprezintă spațiul de joc.

Clasa „Game” este implementată folosint design pattern-ul singleton, întrucât este logic să nu poată să existe decât o singură instanță a unui joc în program. Jocul conține un câmp de joc, precum și metode prin care se poate avansa în simulare, computându-se următoarele generații.
Pentru a genera următoarea stare a unui câmp, obiectul „Game” se folosește de un obiect de tip „Generator” care este o clasă ce conține exclusiv metode și membrii statici (deoarece clasa „Generator” nu are stări interne, am considerat o implementare statică de preferat în locul unei implementări singleton). Clasa „Generator” este răspunzătoare pentru producerea noilor generații, fiind scrisă cu design pattern-ul „Strategy” pentru a alege dinamic, în funcție de subtipul câmpului de joc, modul în care creează o generație nouă.

Clasa „GameField” este o clasă abstractă care servește ca punct de reper pentru cele două clase care o extind, anume „BoundedField” (câmpul de joc mărginit) și „BoundlessField” (câmpul de joc expandabil în orice direcție). Clasa conține metoda abstractă „toggle” prin care se poate inversa starea unei anumite celule (prin coordonatele acesteia).

Clasa „BoundedField” reține, în mod intern, suprafața de joc ca o matrice bidimensională de booleene, în care fiecare booleană cu valoarea „adevărat” reprezintă o celulă vie și fiecare booleană cu valoarea „fals” reprezintă o celulă moartă. Clasa are mai mulți constructori pentru a putea primi o configurație inițială sub forma unei matrici bidimensionale de booleene sau ale unei liste de celule vii. Metoda această de implementare este mai eficientă din punct de vedere al generării următorului pas, dar ineficientă din punct de vedere al memoriei, mai ales când raportul celule vii – celule moarte este foarte mic.

Clasa „BoundlessField” reține, în mod intern, o listă cu celulele vii, fără a mai consuma memorie pentru a reține celule moarte. La fel ca clasa „BoundedField” și aceasta conține constructori pentru a putea primi o configurație inițială sub forma unei matrici bidimensionale de booleene sau ale unei liste de celule vii. Spre deosebire de variantă bordată, nu există o restricție a dimensiunii spațiului de joc pe care un obiect al acestei clase poate să-l reprezinte, deci clasa nu are versiuni de constructor pentru a specifica limite fizice. Această reprezentare este foarte eficientă din punct de vedere al memoriei utilizate și mult mai ușor de înțeles și utilizat decât alte reprezentări ale unui teren de joc nemărginit, însă algoritmul de trecere la o nouă generație este mai lent decât în cazul mărginit.

Clasa „Cell” este o clasă utilizată în principal de clasa „BoundlessField” pentru lista internă a celulelor vii, dar este folosită și de constructorul cu lsite al ambelor subclase a „GameField”. O celulă constă în două coordonate bidimensionale și doi constructori: unul care creează o celulă la o poziție anume și altul de copiere.

Clasa principală a programului, „Main” conține o metodă „main” și o metodă auxiliară. Aceasta asociază unui joc un câmp de joc, al cărui tip exact este determinat la rulare prin interemediul parametrilor din linia de comandă cu care a fost apelată metoda „main”. Tot în linia de comandă se specifică fișierul de inpu precum și fișierul de output al programului. Fișierul de input deține toate informațiile necesare unei simulări: o posibilă dimensiune minimă a spațiului de joc, numărul de pași de efectuat și configurația inițială.

Dumitru Mihai-Valentin 325 CB

3. Implementare

Programul a fost dezvoltat folosind principiile TDD. Fiecare etapă începea cu scrierea unor teste pentru o funcționalitate dorită pentru care, mai apoi, se scria codul.

Pșii urmați au fost următorii:

-crearea unui spațiu de joc mărginit care poate primi o configurație inițială prin intermediul mai multor constructori
-crearea unui getter care să returneze reprezentarea spațiului de joc mărginit
-crearea unei clase de joc singleton care să conțină un spațiu de joc
-scrierea unei metode „next” într-o clasa „Game” (care conține un spațiu de joc mărginit) pentru a putea produce următoarea generație
-supraîncărcarea acesteia cu o metodă care produce următoarele n generații
-definirea unei metode „toggle” în interiorul spațiului de joc pentru a putea manipula celule individual
-schimbarea spațiului de joc mărginit încât acesta să extindă o clasă abstractă; schimbarea clasei „Game” încât să conțină o referință la clasa de bază
-extinderea acestei clase abstractă într-un spațiu de joc nemărginit care să poată primi o configurație inițială prin intermediul mai multor constructori (pentru acesta a fost implementată clasa Cell)
-crearea unui getter care să returneze reprezentarea spațiului de joc nemărginit
-definirea unor metode „nextBoundless” și „nextBoundless(int)” în clasa „Game” pentru generearea unor noi etape într-un spațiu de joc nelimitat
-mutarea metodelor de tip „next” și „next(int)” din clasa „Game” într-o clasă statică „Generator” care se ocupa cu computarea noilor generații, folosind o strategie pentru a decide ce algoritm să ruleze
-scrierea unei clase principale care să decidă prin intermediul liniei de comandă tipul de spațiu de joc, a cărui configurație inițială e citită dintr-un fișier de input și a cărui configurație după un număr dorit de pași e scrisă într-un fișier de output

Pentru fiecare din acești pași au fost scrise teste corespunzătoare în prealabil, folsoindu-se framework-ul Junit. Unele teste au fost modificate ulterior, odată cu refactorizarea codului pentru a reflecta schimbările în design pe măsură ce proiectul s-a dezvoltat.

În surse există următoarea structură:
-pachetul „gameoflife” conține clasele publice „Game”, „GameField” din care sunt extinse clasele „BoundedField” și „BoundlessField”, clasa „Cell” și clasa package-private „Generator”.
-pachetul default conține clasa „Main”.

În teste există următoarea structură:
-pachetul „fieldtests” conține testele realizate pe câmpuri de joc și e alcătuit din clasele publice „BoundedFieldConstructorTest”, „BoundedFieldTest”, „BoundlessFieldConstructorTest”, „BoundlessFieldTest”.
-pachetul „gametests” conține testele asupra modului de funcționare propriu-zisă a jocului și e alcătuit din clasele publice: „GameTestBasic”, „GameTestBounded”, „GameTestBoundless”.
-printre teste există și pachetul ajutător „testextra” care conține clase utile pentru teste: clasa „TestUtil” care încorporează câteva metode statice și clasa „CellComparator” care implementează un comparator pentru celule.