10.01.2015

Temă 3 POO

Boțârleanu Robert-Mihai

321CB

Reviewer	Owner	Version
	Boțârleanu Robert-Mihai	1.0

Cuprins

1. Introducere
2. Design choices
3. Implementare
4. Concluzii
5. Diagramă

1. Introducere

Scopul temei este acela de a simula “The Game of Life” gândit de John Conway. Abordarea temei a fost centrată pe crearea unei interfețe grafice pentru utilizator astfel încât acesta să poată vizualiza evoluția jocului.

Am încercat să creez o interfață intuitivă și care să permită utilizatorului abilitatea de a defini ușor configurația inițială a jocului, să poată opri în orice moment pentru a vizualiza starea plănșii la orice generație, dar și abilitatea de a crește/scădea viteza simulării și să selecteze între cele două tipuri de simulări Border și Borderless permițând abilitatea de a comuta rapid între aceste doua abordări.

2. Design choices

În realizarea temei am ales o abordare centrată pe interfața grafică. Pentru a simplifica construcția aplicației, am ales o abordare “Top-down”. Prima clasă creată a fost cea pentru World, ea moștenind superclasa JFrame. Acest frame este impărțit în două panele: unul pentru celule, care va expune simularea și altul pentru butoanele cu care utilizatorul poate interacționa cu aplicația.

Pentru a putea reține parametrii jocului am utilizat o clasa de configuratie care foloseste pattern-ul de Singleton: “Configuration”. Instanța acestei clase va conține informații referitoare la viteza de derulare a simulării, abordarea pentru border, cât și configuratia celulelor pentru care se face simularea.

Am decis ca obiectul de tip “GameEngine” folosit de clasa Game pentru a parcurge toate celulele și a le actualiza starea pentru o generație următoare să analizeze doar celulele din interiorul matricei, fără cele de pe graniță. Pentru cele de la margini, am implementat un pattern de Strategy pentru a alege între tipul Border și cel Borderless.

Când am abordat problema unui joc “fără granițe” am decis să consider matricea de celule ca fiind toroidală. Astfel, pentru celulele de la granițe am analizat corespunzător numărul de vecini.

3. Implementare

Pentru celule, am creat o clasă “Cell” care să conțină un flag pentru a determina dacă acea celulă este vie la orice moment, flag-ul fiind numit “alive” și având tipul boolean.

Am considerat ca simularea să fie prezentată pe un Panel ce conține un GridLayout de celule, obiecte ale clasei “Cell”. Butoanele si ComboBox-urile au fost adăugate în border-ul de sud al frame-ului.

Pentru a reține configurația simulării am folosit o matrice pătratică formată din obiecte de tip Cell.

Atunci când utilizatorul apasă butonul “Start” aplicația va genera o instanță a clasei “GameThread” care extinde Thread, iar acesta va rula într-un loop până la apelarea metodei “stopThread” atunci când butonul de “Stop” este apăsat.

Pentru Strategy am creat o interfață „BorderStrategyInterface” care este implementată de două clase: BorderStrategy și BorderlessStrategy. Acestea au în comun o metodă “resolveCollision” care primește ca parametrii starea actuală a simulării, o matrice de aceeasi dimensiuni și tip care reprezintă starea viitoare a simulării după o generație, cât și o poziție a celulei de analizat determinată de indicii pentru linie și coloană.

Pentru a determina viteza jocului am scris un enumeration “Speed” care conține trei valori: SLOW, MEDIUM, FAST și VERY FAST, fiecare reprezentând un număr întreg folosit pentru a determina de câte ori pe secundă se va actualize simularea pe ecran.

Clasa “GameEngine” va parcurge toate celulele din matrice și, pentru cele care nu se află pe graniță va determina la fiecare numărul de vecini vii și, în funcție de acesta și de starea celulei, va seta starea sa în generația următoare. Pentru cele care se află pe border, se va apela pentru fiecare „resolveCollision”, metodă specifică claselor ce implementează interfața BorderStrategyInterface. În funcție de tipul de abordare ales la momentul respectiv, celulei respective i se va determina soarta considerând matricea ca fiind fie finită, fie toroidală.

Clasele BorderedStrategy și BorderlessStrategy vor urma un principiu comun pentru a determina dacă o celulă va supraviețui în generația următoare: vor determina numărul de vecini vii și în funcție de acesta se va modifica starea celulei. Diferența este prin felul în care vecinii sunt numărați: BorderedStrategy va ignora orice vecin care nu se află în interiorul matricei, în timp ce Borderless va determina vecinul din matrice ca și când aceasta ar fi toroidală.

După ce tuturor celulelor le-a fost analizată situația, la final GameEngine va actualiza fiecare celulă cu valoarea din generația următoare și, dacă valoarea este diferită față de cea din generația actuală aceasta va fi reafișată pe ecran.

4. Concluzii

În realizarea acestei teme am folosit o abordare “Top-Bottom” centrată pe o interfață grafică care să permită vizualizarea ușoară a simulării și o testare automata mai simplă prin abilitatea utilizatorului de a folosi mouse-ul pentru a seta configurația initială și a controla decurgerea simulării.

În implementarea temei am ales să folosesc două design patterns: Strategy și Singleton. Acestea au fost gândite pentru a face codul mai lizibil și să creeze o structură ușor de înțeles și, dacă este necesar, de modificat. Pattern-ul de Strategy asigură că se pot adăuga noi metode de abordare a problemei “borderless” și conduce spre un cod în care se pot interschimba algoritmi ușor.

Înainte de realizarea temei, am gândit structura generală a claselor și, la fiecare pas am creat teste care să acopere toate posibilele cazuri. Acest lucru a asigurat că algoritmii sunt minimali și eficienți și că nu există situații care să cauzeze probleme.

5. Diagramă

Se regăsește în diagrama.pdf