Marele vrăjitor - Salvează lumea

Diaconescu Bogdan Florin

CR 3.2A

Technical report

1 Structura aplicatiei

Pentru rezolvarea problemei am creat urmatoarele clase:

- **Demon**: reprezinta un demon implementat ca si un thread.

  - Fiecare demon este identificat prin intermediul variabilei *number*.
  - Un demon este reprezentat in interiroul unui cuib printr-o pereche de coordonate (X, Y).
  - Fiecare demon are un nivel de abilitati sociale care poate fi crescut sau scazut si este reprezentat prin variabila *socialLevel*.
  - Pentru contorizarea loviturilor am folosit o variabila contor *hits* .
  - Am utilizat un flag *hitsFlag* pentru cazul in care un demon se loveste de zid si este pus sa astepte 2 ture.
  - Fiecare demon contine o instanta *coven* a cuibului in care se afla .
  - Metoda *run* - demonul va executa la infinit urmatoarele actiuni:
    - crearea unui ingredient
    - miscarea in interiorul cuibului
    - odihna pentru 30 milisecunde
    - retragerea din cuib
  - Metoda *changePosition* - schimba coordonatele demonului in timpul miscarii in cuib.
  - Metoda *stopWork* - il face pe demon sa se opreasca pentru un timp cuprins intre 10 si 50 de milisecunde.
  - Metoda *reportPosition* - afiseaza pozitia curenta a demonului in cuib.

- **DemonRetirement** - reprezinta un thread utilizat pentru retragerea unui demon. La fiecare 50 de milisecunde se cere permisunea semaforului *demonRetireSemaphore* si daca se primeste se retrage un demon.

- **DemonSpawner** - reprezinta un thread folosit pentru crearea demonilor in fiecare cuib. Clasa contine o variabila *coven* ce indica cuibul in care vor fi creati demonii, dupa care se apeleaza functia *sleep()*, ce adoarme threadul pentru o perioada cuprinsa intre 500 si 1000 de milisecunde.

  - Metoda *spawnAnElf*:

    * primeste zavorul pentru cuib si-l blocheaza
    * verifica daca numarul demonilor este mai mic decat N/2, caz in care ceaaza un nou demon
    * blocheaza zavorul pentru lista demonilor pentru ca numarul demonilor sa nu poata fi modificat
    * adauga demonul in cuib si creste numarul acestora
    * elibereaza zavorul pentru contorul demonilor
    * elibereaza zavorul pentru matricea cuibului

- **Witch** - clasa ce reprezinta o vrajitoare prin intermediul unui thread.
  - Fiecare vrajitoare este identificata prin intermediul variabilei *number*.
  - O vrajitoare poate citi intotdeauna numarul de ingrediente din cuiburi, de aceea este nevoie de un vector *covens* ce contine toate cuiburile.
  - *potionQueue* reprezinta coada in care vrajitoarele vor pune potiunile.
  - Vrajitaorele creeaza potiunile in functie de retete, de aceea este nevoie de un ArrayList *potionList* ce contine retele stiute.
  - Ingredientele detinute de o vrajitoare sunt stocate in vectorul *availableIngredients*. Deoarece exista 10 tipuri de ingrediente, acestea vor fi sortate in vector astfel: pe pozitia 0 va fi ingredientul 0, pe pozitia 1 ingredientul 1 samd.
  - Potiunile sunt create prin intermediul functiei *makePotion*. Pentru ficare reteta se verifica daca aceasta se poate face cu ingredientele disponibile prin intermediul functiei *checkReceipt* din clasa *PotionReceipt*.
  - *run* - threadul va executa la infinit urmatoarele actiuni:
    * primeste un ingredient din cuib pe care il pune in vectorul *newIngredients*
    * ingredientele noi sunt puse apoi in vectorul *availableIngredients* in functie de tipul lor.
    * este creata potiunea prin intermediul functiei *makePotion*
    * potiunea este trimisa Marelui Vrajitor prin intermediul cozii *potionQueue*
    * este oprit threadul pentru o perioada cuprinsa intre 10 si 30 de milisecunde.
  - *getIngredientFromCoven* - intra intr-un cuib la intamplare si ia un ingredient

- **PotionTransfer** - clasa ce reprezinta o coada prin intermediul careia se realizeaza transferul potiunilor de la vrajitoare catre Marele Vrajitor.
  - Contine 2 variabile *head* si *tail* ce reprezinta capul si coada cozii.

  - De asemnea, contine si un vector *potions* de tipul int cu dimensiunea 10 pentru stocarea potiunilor.
  - Potiunile sunt introduse in coada de catre vrajitoare prin functia
  - *giveGift* si sunt primite de Marele Vrajitor prin intermediul functiei
  - *receiveGift*.

- **PotionReceipt** - clasa ce reprezinta o reteta de potiuni.
  - Fiecare potiune este identificata prin variabila *number*.
  - Ingredientele necesare pentru crearea potiunii sunt stocate intr-un ArrayList *necessaryIngredients*.
  - Fiecare reteta necesita un timp *time* pentru a fi creata.
  - Tipul de ingrediente necesare, precum si timpul sunt generate in mod random in interiorul constructorului.
  - Cu metoda *checkReceipt int[] availableIngredients* se parcurge vectorul de ingrediente necesare si se verfica daca acestea se gasesc in vectorul *availableIngredients* dat ca si parametru functiei. Functia returneaza *true* in cazul in care potiunea se poate crea, respectiv *false* in caz contrat.

- **MainClass** - contine metoda main:
  - *main* :
    * crearea cozii *potionQueue*
    * crearea Marelui Vrajitor
    * crarea Cercului Marelui Vrajitor
    * Cercului Marelui Vrajitor incepe crearea cuiburilor
    * Marele Vrajitor primeste potiuni de la vrajitoare

- **Grand Sorcerer** - reprezinta Marele Vrajitor si extinde clasa *Thread*.
  - Contine o instanta a clasei PotionTransfer, *giftQueue* utilizata pentru primirea potiunilor
  - Marele Vrajitor va primi incontinuu potiuni

- **Zombie** - clasa ce reprezinta un strigoi prin intermediul unui thread.
  - Fiecare zombie este identificat prin variabila *number*.
  - Un zombie poate ataca orice cuib de aceea este nevoie de un vector *covens[]*.
  - Metoda *run* - strigoiul va executa la infinit urmatoarele actiuni:
    * genereaza un numar random *covenNumber* ce reprezinta cuibul atacat.

        * genereaza un numar random *demonKillNumber* intre 5 si 10 ce reprezinta numarul de demoni ce vor fi omorati.
        * daca numarul demonilor din cuibul ales random este mai mare de 5 se apeleaza functia *killDemon()*.
        * se asteapta un numar de secunde cuprins intre 500 si 1000 de secunde.

- **Coven** - clasa ce reprezinta cuibul prin intermediul unui thread.
    - *number* - numarul prin care este identificat cuibul
    - *N* - dimensiunea matricei cuibului
    - *demons* - ArrayList ce contine demonii existenti in cuib
    - *ingredients* - vector de tip int ce contine ingredientele existente in cuib
    - *covenLock* - zavor folosit pentru accesul la matricea cuibului
    - *demonsListLock* - zavor folosit pentru accesul la lista de demoni
    - *witchSemaphore* - semafor pentru numarul maxim de vrajitoare dintr-un cuib(10)
    - *ingredientsLock* - zavor pentru accesul la lista de ingrediente
    - *zombieSemaphore* - semafor pentru numarul maxim de strigoi ce pot ataca un cuib(am considerat numarul maxim 10 la fel ca la vrajitoare)
    - *run* - threadul va executa la infinit urmatoarele actiuni
        * cere tuturor demonilor sa raporteze pozitia
        * se apeleaza functia slepp pentru 3000 milisecunde
    - *moveDemon* - muta un demon in cuib:
        * blocheaza zavorul *covenLock* pentru matricea cuibului
        * incearca sa mute un demon in orice directie, in caz contrat acesta este blocat
        * mutarea in orice directie presupune schimbarea coordonatelor in matrice, crearea unui ingredient (daca este indeplinita conditia *hitsFlag == 0*, adica daca demonul nu s-a lovit de zic si nu trebuie sa astepte 2 ture), cresterea nivelului social al demonului(daca sunt indeplinite conditiile), modificarea pozitiei curente a demonului si raportarea pozitiei tuturor demonilor
        * deblocarea zavorului pentru matricea cuibului
    - Prin intermediul metodelor *canMoveUp*, *canMoveDown*, *canMoveRight*, *canMoveLeft* sunt verificate mutarile posibile ale unui demon
    - *addDemon* - adauga un nou demon in cuib:
        * blocheza zavorul listei de demoni
        * daca nu exista deja un demon in pozitia generata random, se adauga demonul in lista de demoni, toti demonii raporteaza pozitia curenta si se elibereaza zavorul

- *askDemonsForPosition* - functie prin care este afisata pozitia tuturor demonilor existenti in cuibul respectiv
  - blocheaza zavorul matricei cuibului
  - blocheaza zavorul listei de demoni
  - blocheaza zavorul listei de ingrediente
  - este afisata pozitia tuturor demonii din lista de demoni existenti
  - deblocarea celor 3 zavoare
- *getIngredient* - metoda utilizata de vrajitoare pentru a primi ingredientele din cuib
- *createIngredient* - adauga un ingredient in lista de ingrediente
- *retireDemon* - retrage un demon din cuib
- *killDemon* - metoda folosita de strigoi pentru a omora un demon
  - asteapta permisiunea semaforului *zombieSemaphore*
  - omora un demon random din cuib
  - blocheaza zavorul listei de ingrediente
  - sterge toate ingredientele existente in cuib
  - deblocheaza cele 2 zavoare

- **GrandSorcererCircle** - Cercul Marelui Vrajitor
  - *nrCovens* - numarul de cuiburi existente
  - *covens* - vector ce contine toate cuiburile existente
  - *spawners* - vector ce contine toate thread-urile spawner
  - *nrTotalDemons* - numarul total de demoni existenti
  - *demonsCounterLock* - zavor pentru numarul total de demoni existenti
  - *witches* - vector ce contine toate vrajitoarele existente
  - *potionQueue* - coada pentru transferul potiunilor de la vrajitoare la Marele Vrajitor
  - *demonfRetireSemaphore* - semafor pentru retragerea demonilor
  - *demonRetire* - thread pentru retragerea unui demon
  - *getDemonsCounterLock* - returneaza zavorul pentru numarul de demoni
  - *createCovens* - creaza toate cuiburile, spawn de demon, vrajitoarele si incepe executia threadurilor.

2 Implementarea taskurilor

2.1 Metode de sincronizare

Pentru rezolvarea problemei si sincronizare am utilizat 3 zavoare:

- Un zavor *covenLock* care este folosit pentru accesul la cuib ce este reprezentat printr-o matrice, atunci cand un demon se muta sau raporteaza pozitia(2 demoni nu se pot misca simultan si nu se pot misca cand raporteaza pozitia).
- Un zavor *demonsListLock* care este folosit pentru a controla accesul la lista de demoni(2 demoni nu pot fi adaugati simultan).
- Un zavor *ingredientsLock* care gestioneaza accesul la lista de ingrediente atunci cand un demon este intrebat de pozitia sa(o vrajitoare nu poate primi ingrediente cand demonii raporteaza poxitia), cand vrajitoarele primesc ingrediente din cuiburi(2 vrajitoare nu pot primi acelasi ingredient) sau cand este creat un nou ingredient(modificarea listei cu ingrediente).

Pentru a sincroniza accesul vrajitoarelor la cuiburi am folosit un semafor *witchSemaphore* cu 10 permisiuni pentru fiecare cuib deoarece maxim 10 vrajitoare pot accesa un cuib in acelasi timp. Acest semafor este incrementat dupa ce o vrajitoare ia un ingredient dintr-un cuib.

Deoarece demonii pot crea 10 tipuri de de ingrediente diferite, numerotate de la 1 la 10, am utilizat vectori pentru a stoca ingredientele si pentru a le sorta. Astfel,indicele fiecarui element din vector va reprezenta si tipul de ingredient.

Demonii sunt identificati prin numar, iar pentru evita cazul in care exista 2 demoni cu acelasi numar in cuiburi diferite am utilizat un zavor pentru accesul la lista de demoni.

Vrajitoarele creaza potiuni dupa o anumita reteta. Tipul de ingredientele necesare, precum si timpul necesar sunt generate random. O reteta este reprezentata de clasa *PotionReceipt*.

Dupa ce o potiune este creata, aceasta este transferata catre Marele Vrajitor prin intermediul unei cozi. Metodele pentru punerea si scoaterea elementelor din coada au fost create utilizand metoda synchronized.

2.2 Sarcini de indeplinit:

Puteți retrage un demon

Pentru retragerea unui demon am creat o noua clasa *DemonRetirement* reprezentata printr-un thread care la fiecare 50 de milisecunde va retrage un demon. Fiecare demon va fi creat intr-un cuib, apoi va incerca sa obtina permisunea de a se retrage. Cand un demon este retras, acesta este sters din

matricea cuibului si din lista ce contine toti demonii existenti in cuibul respectiv.

Demoni adormiti - semaphores

Cand un demon ajunge pe diagonala principala, acesta va incerca sa primeasca permisiunea semaforului *counterSemaphore* pentru a modifica valoarea contorului *counter* ce indica numarul demonilor ce asteapta la bariera pana cand valoarea variabilei counter este mai mica decat N. (Numarul maxim de demoni dintr-un cuib a fost modficat de la N/2 la N).

Demoni adormiti - cyclic barrier

Cand un demon ajunge pe diagonala principala, acesta va astepta la bariera pana cand numarul demonilor va ajunge la N, dupa care isi va continua deplasarea in cuib.

Propria bariera ciclica a fost creata in clasa *CyclicBarrier*. Metoda *await()* foloseste un zavor *counterLock* pentru a modifica variabila contor a demonilor ce asteapta la bariera cat timp counter este mai mic decat N.

3 Observatii:

- Pentru ca strigoii sa nu atace prea repede cuiburile iar algoritmul sa dea eroare atunci cand un numar de demoni este omorat, deoarece nu exista atatia demoni in cuib, am decis ca strigoii sa atace doar cand numarul de demoni existenti intr-un cuib este mai mare decat 10.
- Pentru rezolvarea taskurilor am redus dimensiunea matricelor la 30x30 pentru a se indeplini mai repede conditiile ca toti demonii sa se afle pe diagonala principala.
- Deoarece demonii sunt foarte rapizi in crearea ingredientelor(au nevoie doar de 30 de milisecunde de odihna), vrajitoarele trebuie sa fie si ele foarte rapide in preluarea ingredientelor(vor avea un timp de odihna cuprins intre 10 si 30 de milisecunde).
- Mai multe vrajitoare pot trimite potiuni catre Marele Vrajitor, de aceea acesta nu trebuie sa astepte intre primirea potiunilor(nu are nevoie de odihna).