Labirintul magic

Tema 3

Responsabili: Ștefania Budulan și Alexandru Mustață

Dată publicare temă: 11 dec. 2017
Hard Deadline: 10 ian. 2018, ora 10 PM

1. Introducere

Scopul temei este acela de a exersa modelarea obiectelor în Java, precum și proiectarea și implementarea structurii logice a proiectului, a fiecărei clase în parte, a dependențelor dintre acestea, în vederea obținerii unui cod puternic reutilizat și reutilizabil. De asemenea, aveți ocazia să testați îmbinarea dintre concepte POO, structuri de date și algoritmi, similară cu cea din practica de zi cu zi a oricărui programator.

2. Descrierea temei

Eroul nostru, căruia îi vom spune Dubluve, din dorința de a-i menține ascunsă identitatea, dorește să devină mai bun în a găsi ieșirea dintr-un labirint. Acest labirint nu este ca oricare altul; el este magic: intrarea și ieșirea sunt, de fapt, niște portaluri magice care îi permit lui Dubluve să călătorească în timp și spațiu. Din acest motiv, ele se găsesc plasate oriunde în interiorul labirintului, oricât de aproape sau departe unul față de celălalt, însa eroul știe sigur că există cel puțin un drum de la portalul de intrare, la portalul de ieșire, prin labirint.

Fiecare antrenament începe prin propulsarea eroului prin portalul de intrare.

1. Primul tip de antrenament (Task1)

Dubluve cunoaște înălțimea și lățimea fiecarui labirint din acest tip de antrenament, însă nu are o hartă care să il ghideze către portalul de ieșire. Cerand sfaturi către cercul lui de prieteni, a aflat că o strategie bună pentru a încerca să găsească ieșirea din labirint este să urmeze mereu calea din dreapta sa.
Problema este, însa, ca aceasta strategie nu funcționeaza pentru labirinturile magice, care pot avea portalul de ieșire chiar în mijlocul lor.
Dupa câteva zile de gândire, lui Dubluve îi vine o idee fantastică: „ce-ar fi dacă aș marca drumul pe care am mai fost?”. Așadar, Dubluve decide să pună câte o pietricică pe fiecare celulă prin care trece, deplasându-de așa, celulă cu celulă, și dezvoltă urmatoarea strategie pentru momentul în care trebuie să aleagă pasul următor, la o răscruce de drumuri:

- Dintre celulele următoare, o va alege mereu pe cea mai din dreapta (față de direcția de orientare a eroului, desigur, in ordinea R, F, L, B – explicații mai jos), dar asta doar dacă a trecut prin toate celulele de același număr (minim) de ori.
- Altfel, o va alege pe cea care a fost vizitată de cele mai puține ori.

Explicații suplimentare

Eroul este orientat inițial către nordul labirintului, precum în exemplul de mai jos.

Ordinea direcțiilor in care Dubluve incearcă explorarea:

- R – right (celula din dreapta eroului, față de orientarea curentă a acestuia);
- F – front;
- L – left;
- B – back.

Eroul o va alege pe cea mai puțin vizitată, iar dacă sunt mai multe celule vizitate de același număr (minim) de ori, acesta va respecta ordinea de mai sus. Desigur, el poate alege numai dintre celulele care sunt libere (i.e. pe care nu exista un zid construit). Pentru implementare, veți folosi un tip de Colecție [1], din Java, pentru care veți scrie un Comparator [2] peste obiectele clasei care definește celula labirintului.

Dacă celula cu portalul de ieșire se află în vecinătatea eroului, o va alege pe aceea, indiferent de prioritatea anterioară.

2. Cel de-al doilea tip de antrenament (Task2)

Pentru acest tip de antrenament, eroul, pe langă înălțimea și lățimea labirintului, are la dispoziție chiar o hartă pe care este marcat întreg labirintul, cu **portalul de intrare ('I')**, **portalul de ieșire ('O')**, **ziduri ('#')** și **celule libere ('.')**. Fiind bun la logică matematică și algoritmică, eroul va calcula în avans cel mai scurt drum din poziția în care se află (celula portalului de intrare), către portalul de ieșire, pentru ca, mai apoi, să îl urmeze direct și neîntrerupt.

3. Date de intrare, ieșire

Tema va trebui să poată fi rulată astfel:

`./tema3 <nr_task> <fisier_input> <fisier_output>`, unde:

<nr_task> ϵ {1, 2}

<fisier_input>	<fisier_input>
<height> <width> $L_{0,0}$ $L_{0,1}$ ... $L_{0,w-1}$ $L_{1,0}$ $L_{1,1}$ ... $L_{1,w-1}$ ... $L_{h-1,0}$ $L_{h-1,1}$ ... $L_{h-1,w-1}$	<moves> start.i start.j m1.i m1.j ... exit.i exit.j
De pe prima linie se citesc înălțimea (h) și lățimea (w) labirintului, iar de pe următoarele h linii, matricea care îl reprezintă.	Pe prima linie, se scriu numarul m de mutări efectuate pana la ieșire, plus 1 (practic, numarul de celule prin care trece), iar pe urmatoarele m linii, perechea (i, j) de coordonate ale fiecarei celule.

Obs. Prima pereche din fișierul de output este chiar locația portalului de intrare, iar ultima, locația portalului de ieșire.

4. Exemple

- **Task1**

Test.in
```
5 5
..#I.
#.##.
.....
O#.#.
.....
```

```
..#^.
#.##.
.....
O#.#.
.....
```

```
..#I>
#.##.
.....
O#.#.
.....
```

```
..#I.
#.##v
.....
O#.#.
.....
```

```
..#I.
#.##.
....v
O#.#.
.....
```

```
..#I.
#.##.
...<.
O#.#.
.....
```

...

```
..#I.
#.##.
.....
v#.#.
.....
```

Vor rezulta 15 poziții:
`0 3, 0 4, 1 4, 2 4, 2 3, 2 2, 2 1, 1 1, 0 1, 0 0, 0 1, 1 1, 2 1, 2 0, 3 0`

- **Task2**

Drumul cel mai scurt, marcat pe harta de mai sus este:

```
..#@@
#.##@
@@@@@
@#.#.
.....
```

Vor rezulta 9 poziții:
`0 3, 0 4, 1 4, 2 4, 2 3, 2 2, 2 1, 2 0, 3 0`

3. Testare

Tema va fi testată pe **VMChecker** (va apărea în curând), astfel încât va trebui să aveți și un **makefile** cu regulile **build**, **clean** și **run**.

Teste vor fi disponibile spre sfârșitul săptămânii următoare.

4. Constrângeri

Constrângeri legate de platformă

- Pentru realizarea temei trebuie folosită o versiune de Java nu mai recentă de Java 7, aceasta fiind versiunea care rulează pe VMChecker.
- Limita de timp pentru fiecare test în parte va fi publicată mai târziu, odată cu postarea temei pe VMChecker.

Constrângeri/ghid de implementare

- Labirintul va fi implementat în clasa **Maze**, reprezentând o matrice de obiecte de tip **Cell** (clasă abstractă). Fiecare tip de celulă a labirintului va extinde clasa abstractă Cell.
- Dacă eroul încearcă să păsească în afara labirintului, vom întoarce o excepție de tipul **HeroOutOfGroundException**, iar dacă se încearcă deplasarea înspre unul dintre zidurile labirintului, vom arunca o excepție de tipul **CannotMoveIntoWallsException**.
- Pentru ambele task-uri veți folosi tipuri corespunzătoare de Colecții [1].
- Pentru task-ul 1 veți implementa un Comparator [2], necesar la stabilirea priorității de vizitare a celulelor.

5. Punctaj

- 60% Testare automată
- 20% Respectarea constrângerilor de implementare
- 20% Coding Style + Readme + JavaDoc + Makefile

Notă: Temele care nu trec niciun test nu vor primi punctajul aferent *coding style și explicații.*

6. Resurse

[1] Colecţii

[2] Comparator

7. Updates

v1 – Actualizare parametri rulare pentru a putea testa independent cele două task-uri.