

Tema 1 - League of OOP

- **Data publicarii:** 17.10.2017 16:10
- **Data ultimei modificari:** 17.10.2017 16:10
- **Deadline soft:** 07.11.2017, 23:55
- **Deadline hard:** 09.11.2017, 23:55
- **Responsabili:** Ionuț Bîrsu, Laurențiu Stamate

Obiective

- familiarizarea cu Java și conceptele de bază ale POO
- fundamentarea practică a constructorilor și a agregării/moștenirii
- dezvoltarea unor abilități de bază de organizare și design orientat-obiect
- respectarea unui stil de codare și de comentare

Scenariu

Suntem într-un joc MMO-style. Eroii noștri își petrec viața într-un univers 2D, pe care îl explorează și în care se dezvoltă.

După cum este cunoscut din jocurile MMO și RPG, personajele noastre sunt diverse. Fiecare aparține unei anumite clase și are un anumit set de abilități la care se pricepe cel mai bine și pe care le poate dezvolta cel mai repede; le vom analiza în detaliu mai jos.

La începutul jocului, eroii noștri sunt plasați pe hartă în locuri bine definite. Apoi vor exista runde de durată unitate, în care toți eroii vor executa câte o anumită mișcare (bine definită) pe hartă.

Când doi eroi ajung în același loc, ei se vor lupta. În această rundă, fiecare erou combatant își va folosi toate abilitățile disponibile împotriva adversarului, o singură dată. După luptă, ei își vor vedea de drum începând de runda următoare.

La sfârșitul jocului (un număr stabilit de runde, cu locații inițiale și mișcări stabilite), programul nostru se va uita la eroii rămași în viață.

Harta

Harta pe care se desfășoară jocul este un dreptunghi 2D, compus din locații ("pătrățele") de dimensiune unitate. Fiecare locație are un anumit set de proprietăți; tipurile de locații sunt:

- Land
- Volcanic
- Desert
- Woods

Eroi

Tipurile de eroi disponibile în LOOP sunt:

- Knight
- Pyromancer
- Rogue
- Wizard

Toți dispun de un număr de hit points (HP, also known as “viață”) și de un număr de puncte de experiență (XP). Există, de asemenea, și un mecanism de level-up în funcție de experiența câștigată.

Experiență și level up

Toate personajele au **XP inițial = 0**, corespunzător nivelului 0. La nivelul 0 toate abilitățile au doar efectul de bază. În momentul în care un personaj câștigă o luptă (își omoară adversarul) XP-ul său va crește după următoarea formulă :

$$XP_Winner = Xp_Winner + \max(0, 200 - (Level_winner - Level_loser) * 40)$$

După cum se poate observa și din formulă, câștigarea unei lupte cu un adversar care este cu cel puțin 5 nivele mai mic nu va aduce nici un plus de XP, dar câștigul în fața unui oponent de nivel mai mare poate aduce chiar mai mult de 200XP.

Pragurile de XP necesare pentru a face level-up se vor calcula după formula :

$$XP_level_up = 250 + nivel_curent * 50$$

La level up un erou va reveni la 100% HP.

Abilități

Fiecare tip de erou are un anumit set de abilități, ai căror parametri sau ale căror efecte depind de **terenul pe care se desfășoară lupta (land modifiers) și de personajul asupra căruia acționează (race modifier)**. Detaliem în continuare.

Pyromancer

Abil în manevrarea focului.

HP: 500 initial, +50 per nivel.

Fireblast - damage mare în runda curentă.

- Damage: 350, +50/level

Victima	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Ignite - damage în runda curentă + damage mai mic în următoarele 2 runde.

1. base damage: 150, +20/level
2. 50 damage per runda, +30/level.

Modificatorii de mai jos se aplică atât pentru base damage, cât și pentru damage periodic:

Victima	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Pe terenul de tip Volcanic Pyromancer-ul se hrănește cu energia mediului înconjurător iar abilitățile sale dau cu 25% mai mult damage.

Knight

HP: 900 initial, +80/level.

Execute - damage în runda curentă; dacă adversarul are un număr de HP mai mic decât o anumită limită, va fi ucis instantaneu.

- base damage: 200, +30/level
- HP limit: $20\% \times \text{viața teoretic maximă a victimei la nivelul ei}$; +1% /level, până la un maxim de 40%

Damage-ul (nu și limita de viață) se modifică în funcție de victimă ca mai jos:

Victima	Modificator
Rogue	+15%
Knight	+0%
Pyromancer	+10%
Wizard	-20%

Slam - damage + incapacitarea adversarului (imposibilitate de mișcare) pentru următoarea rundă.

- base damage: 100 base damage. +40 /level

Victima	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%

Victima	Modificator
Wizard	+5%

Knight-ul este expert în lupta corp la corp. Pe terenul de tip land el este mai puternic decât celelalte clase fiind specializat pe lupta în câmp deschis și primește 15% bonus damage.

Wizard

HP: 400 initial, +30 per nivel.

Wizard-ul are o capacitate mentală superioară care îi permite să reziste în mediul de deșert prin meditație și îmbunătățirea abilităților. Pe terenul de tip **Desert** abilitățile sale ofensive sunt cu **10%** mai puternice.

Drain - scade din viața adversarului proporțional cu cât are deja.

- procent: 20%, +5% /level
- HP de bază: $\min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$
- $\Rightarrow \text{damage} = \text{procent} * \min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$

Modificatorii de mai jos se aplică asupra variabilei procent. Ei sunt multiplicativi, i.e. dacă de exemplu avem un Wizard level 4 ($\Rightarrow 40\%$) și el aplică Drain asupra unui Rogue, avem un procent total de $40\% * 0.8 = 32\%$, deci formula de calcul de damage devine $\text{damage} = 0.32 * \min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$.

Victima	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Deflect - dă damage egal cu un procent **din damage-ul total (fără race modifiers) pe care îl primește de la adversar**

- procent: 35%, +2% /level, până la un maxim de 70%
- *nu are niciun efect asupra unui Wizard* (doi eroi de tip Wizard nu își dau reciproc/recursiv damage)

Victima	Modificator
Rogue	+20%
Knight	+40%
Pyromancer	+30%
Wizard	N/A

Rogue

Expert în sneak-attacks.

HP : 600 initial, +40 /level

Backstab - damage în runda curentă, cu posibilitate de critical hit.

- base damage: 200, +20 dmg /level.
- O dată la 3 lovituri (lovitură = aplicat Backstab, pe orice teren) Rogue-ul poate da **1.5x** damage, **doar dacă în acel moment se afla pe terenul de tip Woods**, altfel se reia ciclul.

Victima	Modificator
Rogue	+20%
Knight	-10%
Pyromancer	+25%
Wizard	+25%

Paralysis - damage prelungit + incapacitarea adversarului pentru un număr de runde

- damage/rundă: 40, +10/level - se aplică în runda curentă (în care se desfășoară lupta) + rundele extra
- număr de runde overtime: 3 (6, dacă lupta se desfășoară pe teren Woods)

Victima	Modificator damage
Rogue	-10%
Knight	-20%
Pyromancer	+20%
Wizard	+25%

Datorită abilităților sale de camuflaj Rogue-ul este mai puternic pe terenul de tip pădure (**Woods**). Pe acest tip de teren Rogue-ul primește **15%** bonus damage.

Mecanism de joc

Jocul este bazat pe runde.

Într-o rundă toate personajele execută câte o mișcare (bine determinată), iar dacă două ajung în aceeași locație, se luptă. O luptă funcționează astfel: fiecare erou își va calcula parametrii propriilor abilități, în funcție de nivelul lor și de terenul pe care se află. Apoi fiecare abilitate va fi modificată în funcție de victimă. După aplicarea abilităților, eroii își vor calcula noile HP și XP după caz, după care runda se încheie.

Atentie, în cazul în care doi eroi se întâlnesc pe aceeași poziție, se calculează în primul rând damage-ul primit de cei doi de la abilitățile overtime. Dacă unul din ei moare din cauza unei abilități overtime, lupta nu va mai avea loc.

Mențiuni

- Harta nu este circulară și nu veți întâmpina cazuri în care eroul să fie nevoit să iasă din hartă.
- Nu vor exista cazuri în care să se lupte mai mult de două personaje în același loc.
- La abilitățile cu efect prelungit (pe mai multe runde), dacă un personaj deja se află sub efectul unei abilități prelungite și suferă de la o nouă abilitate cu efect prelungit, noul efect îl va înlocui pe cel vechi; dacă de exemplu un erou mai are de suferit 5 runde de pe urma unui Paralysis și un Warrior îi aplică un Slam, atunci imobilizarea lui va dura doar o rundă și nu va mai primi

damage pe runde următoare. De asemenea, damage-ul unei abilitati overtime este calculat in functie de level-ul si terenul unde a avut loc batalia si nu se modifica daca cel afectat se muta pe un alt tip de land sau adversarul face level up.

- Damage-ul se calculează cu rotunjire la înt; folosiți `Math.round`. **Atentie, pentru a nu avea probleme din cauza rotunjirii, va sfatuim sa folositi `Math.round` imediat ce ati aplicat amplificarile datorate terenului si race-ului adversarului.**

Cerințe

Ne dorim să modelăm acest joc în stilul orientat-obiect. Vom citi configurația și desfășurarea unui joc dintr-un fișier de intrare, vom rula jocul și vom scrie într-un fișier de ieșire stările eroilor noștri.

În soluțiile voastre, entry-point-ul (metoda public static void main(String[] args)) va fi într-o clasă numită Main dintr-un pachet numit main. Primul argument este numele fișierului de intrare, al doilea este numele fișierului de ieșire. Nu schimbați numele clasei Main, numele pachetului main sau ordinea argumentelor.

Input

Pe prima linie a fișierului se află 2 numere N si M reprezentând dimensiunile terenului. Pe următoarele N linii se vor găsi N string-uri de lungime M fiecare, cu litere corespunzătoare numelor terenurilor (W,L,V,D).

Pe următoarea linie se va găsi un număr P reprezentând numărul de personaje. Pe următoarele P linii se vor găsi câte un string reprezentând rasa fiecărui personaj și poziția lui inițială (e.g. "W 0 1" înseamnă un Wizard poziționat pe rândul 0 coloana 1).

Pe următoarea linie se găsește R, numărul de runde. Fiecare rundă e descrisă de P litere din mulțimea 'U'(up ⇔ rând-), 'D'(down ⇔ rând++), 'L'(left ⇔ col-), 'R'(right ⇔ col++), sau '_' (fără mișcare) indicând direcția în care se mișcă fiecare personaj. Direcțiile sunt atribuite eroilor în ordinea în care au fost descrise personajele in fișierul de intrare.

Output

În fișierul de ieșire veți adăuga P linii în formatul `rasa_pers level_pers xp_pers hp_pers row col`, câte una pentru fiecare erou, în ordinea în care au fost "declarați" inițial în fișierul de intrare. Dacă personajul este mort atunci în loc de statistici, pe linia corespunzătoare lui puneți "dead" (e.g. K dead înseamnă un Knight mort).

Exemplu input/output

Exemplu Input:

```
1 1
L
2
W 0 0
R 0 0
2
—
—
```

Output asteptat:

```
W dead
R 0 200 340 0 0
```

API

Vă punem la dispoziție un [API](#) care vă permite lucrul cu fișiere. `FileIO` este clasa pe care o veți instanția pentru fiecare interacțiune cu fișierele. Ea constă în:

- constructorul `FileIO(String filename, boolean read)`; `filename` este numele fișierului cu care interacționați, `read` este setat `true` dacă interacțiunea este de citire, `false` pentru scriere
- metoda `String readLine()` - întoarce următoarea linie preluată din fișier
- metoda `void writeLine(String line)` - adaugă o linie nouă în fișier
- metoda `close()` - închide fișierul

Pentru includerea acestei clase în proiect consultați secțiunea [Utile](#), aflată mai jos.

De asemenea, metodele din clasa `String` (`split`, `charAt`, `length`, `toCharArray` etc) vă pot fi de mare ajutor. Consultați documentația clasei `String` (link în secțiunea [Utile](#)).

Indicații

- separați conceptele de sine stătătoare în clase separate, nu le îmbinați
- adaptați agregarea și moștenirea la situație (abilitățile, eroii etc), grupați pe cât posibil informația și acțiunile comune în clase generale
- deduceți semnăturile metodelor așa încât să se poată aplica pe orice tip de erou, respectiv tip de locație, abilități etc
- încercați abordări din unghiuri diferite; de exemplu, un erou poate aplica o abilitate asupra altuia, dar ne putem gândi și că o abilitate se poate aplica asupra unui erou (eliminând astfel agentul), sau că un erou poate suferi de la o abilitate etc
- *folosiți-vă de abilitatea runtime-ului Java de a executa metodele din clasele derivate*
- **Nu vă apucați să scrieți direct**; alocați timp modelării și abstractizării, pentru că altfel vă puteți trezi cu o temă muncitorească, cu mult cod din care să nu înțelegeți prea multe și pe care să îl extindeți greu

Bonus

- Challenge-ul este gasiti acea/acele parte/parti ale cerintei unde se poate aplica corect **double dispatch** si sa il aplicati. Hint:[double dispatch](#)
- Va rugam sa mentionati in **README** daca ati implementat bonusul!
- **Atentie! Doar temele cu punctaj maxim pe teste sunt eligibile pentru bonus!**

Punctaj

- 60p trecerea testelor
- 20p **coding style** (vezi [checkstyle](#))
- 15p design și organizare
- 5p README clar, concis, cu "miez"
- 20p folosirea Double dispatch (bonus)

Checkstyle

Mai jos aveți câteva exemple de concepte de avut în vedere pentru a trece testul de coding-style/checkstyle

- numele fișierelor ([ref](#))
- organizarea fișierelor ([ref](#))
- indentarea pe verticală și orizontală ([ref](#))
- declarațiile și inițializările ([ref](#))
- numirea claselor, variabilelor, metodelor, etc. ([ref](#))
- tratarea cazurilor speciale ([ref](#))
- respectarea unui [coding style](#) (nu neapărat acesta, important este să fiți **consistenți** și **consecvenți**)

Nu este necesară exportarea de documente [Javadoc](#) deoarece toate interfețele din schelet sunt documentate.

Pentru a fi eligibil de bonus, tema trebuie să treacă testul de coding-style executat de [Checkstyle \[4\]](#) însă dacă este picat și numărul de erori depistate depășește 30 (o treime din punctajul maxim, fără bonus), atunci punctele pentru coding-style nu vor fi acordate iar dacă punctajul este negativ, *acesta se trunchiază la 0*.

Exemple:

- `punctaj_total = 100 și nr_erori = 200 ⇒ nota_finala = 80`
- `punctaj_total = 100 și nr_erori = 29 ⇒ nota_finala = 100`
- `punctaj_total = 80 și nr_erori = 30 ⇒ nota_finala = 80`
- `punctaj_total = 80 și nr_erori = 31 ⇒ nota_finala = 60`

Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va duce la **anularea punctajului** de pe parcursul semestrului și **repetarea materiei** atât pentru sursă(e) cât și pentru destinație(ii), fără excepție.

You shall indeed not pass !!

Structura arhivei

Arhiva pe care o veți urca pe **VMChecker** va trebui să conțină în directorul rădăcină:

- fișierul README
- fișierul main/Main.java (entry-point-ul aplicației voastre care se află în pachetul main)
- alte fișiere **organizate** cu implementarea voastră

Nu încărcați fișierele de test, checker-ul sau documente generate cu JavaDoc.

Link-uri utile

- [Parametri în linia de comandă în eclipse](#)
- [Documentație clasa String](#)
- [Validarea automată a codului în Java](#)
- [Clean code](#)
- [Tutorial checkstyle](#)
- [Double Dispatch - scurt tutorial](#)
- [Indicații pentru teme](#)
- [Checker](#)

From:

<http://elf.cs.pub.ro/poo/> - Programare Orientată pe Obiecte

Permanent link:

<http://elf.cs.pub.ro/poo/teme/tema1>

Last update: **2017/10/18 23:57**

