Programare orientată pe obiecte

Tema - League of Warriors

Partea I

Data postării: 16.11.2024

Deadline: 13.12.2024 ora 23:55

Ultima modificare: 16.11.2024 ora 20:00

Echipă temă: Carmen ODUBĂȘTEANU, Ștefănel TURCU, Alexandru TUDOR

Colaboratori: Andrei VOICU

Facultatea de Automatică şi Calculatoare

Universitatea Națională de Știință și Tehnologie Politehnica București

Anul universitar 2024 - 2025

Seria CC

1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să aplice corect principiile programării orientate pe obiecte studiate în cadrul cursului;
- să construiască o ierarhie de clase, pornind de la un scenariu propus;
- să utilizeze un design orientat-obiect;
- să trateze excepțiile ce pot interveni în timpul rulării unei aplicații;
- să transpună o problemă din viața reală într-o aplicație.

Tema constă în implementarea unui joc de aventură bazat pe text (text adventure game). Jocul este reprezentat sub forma unei matrice de dimensiune n×m, unde fiecare celulă poate conține diverse elemente, precum inamici sau comori. Programul va oferi jucătorului opțiuni variate, în funcție de tipul de celulă în care se află, oferind permanent suport pentru toate activitățile.

Jucătorul are posibilitatea de a crea mai multe caractere. La începutul fiecărui nivel, va alege personajul cu care dorește să joace. Fiecare personaj posedă atribute și abilități unice, evoluând în timp pe măsură ce câștigă experiență după fiecare eveniment. Experiența va fi cuantificată printr-o valoare numerică, iar la atingerea unor praguri stabilite, personajul va avansa în nivel, îmbunătățindu-și atribute precum puterea, dexteritatea și carisma.

Proiectul propune crearea unui astfel de joc folosind principiile programării orientate pe obiect și funcționalitățile puse la dispoziție de limbajul de programare JAVA, aplicând noțiunile studiate în cadrul orelor de curs și laborator.

2 Reguli de joc

Jocul începe cu crearea unei table de joc, sub formă de matrice, de dimensiuni variabile.

Caracterul jucătorului va fi plasat într-o celulă aleasă aleatoriu. Conținutul celorlalte celule va fi vizibil jucătorului doar în momentul în care caracterul va ajunge pe una dintre celule.

Pentru deplasare, există patru posibilități:

- NORTH - caracterul se va deplasa în celula de deasupra;
- SOUTH - caracterul se va deplasa în celula de dedesubt;
- WEST - caracterul se va deplasa în celula din stânga;
- EAST - caracterul se va deplasa în celula din dreapta.

La aterizarea pe o celulă, jucătorul va putea întâlni unul dintre evenimentele:

- **Inamic.** Este momentul în care caracterul, controlat de jucător prin comenzi speciale, se va lupta cu un inamic. Acesta are două variante - fie folosește un atac obișnuit, fie folosește una dintre abilitățile speciale (foc, gheață, pământ). Desigur, fiecare avantaj costă, iar fiecare inamic ripostează! Dacă jucătorul reușește să învingă inamicul, va câștiga experiență și își va reîncărca o parte din viață și mană.
- **Sanctuar.** Locul în care caracterul se poate recupera, prin încărcarea manei și a vieții.
- **Portal.** Prin portal caracterul trece pentru a merge la următorul nivel. Jocul se va reseta, mana și viața caracterului vor primi valorile maxime, iar harta va primi noi dimensiuni și o nouă așezare a elementelor.
- **Pustiu.** Este o celulă de tranzit, fără un eveniment special.

Jocul se termină doar atunci când, în urma unei bătălii, viața caracterului va ajunge la 0. Se va declara "GAME OVER" iar jucătorul va fi adus în meniul principal unde poate începe un nou joc.

Jucătorul are la dispoziție mai multe personaje din care poate alege la începutul jocului. Fiecare personaj are atribute și abilități unice. Personajele vor evolua în timp, după fiecare eveniment câștigând experiență, cuantificată printr-un număr. La avansarea în nivel, atributele caracterului (puterea, dexteritatea, carisma) vor evolua.

3 Flow-ul jocului

1. Inițializări
   - Jocul începe cu partea de autentificare care va permite introducerea credențialelor (email și parolă) unui cont. Dacă există un cont asociat credențialelor introduse, se va afișa o listă cu personajele contului.
   - Se alege un personaj existent pe contul respectiv.
   - Se generează harta și se poziționează caracterul pe o celulă goală.
2. Loop / Execuție continuă
   - Se afișează harta generată în fiecare rundă în care jucătorul se poate deplasa (harta nu se va afișa când jucătorul este în luptă).
   - Dacă jucătorul se află pe o celulă cu un inamic, se vor afișa două opțiuni - atacă inamic sau folosește abilitate. Lupta va fi pe ture, alternativ. După fiecare alegere de atac a jucătorului (abilitățile vor fi însoțite de costul de mană și dauna pe care o va provoca inamicului) urmează o alegere a inamicului. Această alegere este generată aleator - fie folosirea unei abilități (generată aleator, acțiune descrisă în secțiunile următoare), fie un atac normal (inamicul va ataca normal și atunci când nu are mană pentru a folosi o abilitate). După înfrângerea unui inamic, viața jucătorului se va dubla (fără a depăși pragul maxim) iar mana va fi reîncărcată complet. De asemenea, va primi un număr aleator de puncte de experiență.
   - Dacă jucătorul se află pe o celulă de tip Sanctuar, se vor adăuga valori alese aleator la nivelul de viață și mană.
   - Dacă jucătorul se află pe o celulă Portal, va primi un număr de puncte de experiență egal cu numărul nivelului pe care l-a terminat, înmulțit cu 5. Portalul va reseta jocul și va incrementa nivelul și numărul de jocuri jucate de utilizator.
   - Dacă jucătorul se află pe o celulă de tip Pustiu, jucătorul va trebui să aleagă o nouă direcție de mers.
   - Se citește următoarea mutare de la terminal.
   - Se face mutarea. Caracterul se poate muta și pe celule deja vizitate, toate aceste celule fiind acum de tip Pustiu.
   - Repetă.
3. Final
   - Jocul se termină în momentul în care jucătorul selectează părăsirea jocului sau când viața jucătorului ajunge la 0. În ambele cazuri, utilizatorul va fi dus în pagina de selectare a unui caracter.
   - Inițial, celulele vor fi afișate identic, urmând ca pe măsură ce personajul le-a vizitat, acestea să fie descoperite (de exemplu, dacă personajul se întoarce într-o căsuță în care a fost un inamic, acesta nu va reapărea pentru o nouă luptă).

4 Arhitectura aplicației

4.1 INTERFEȚE

4.1.1 Battle

Această interfață definește metode comune pentru toate entitățile de tip Character sau Enemy:

- ```
  public void receiveDamage(int);
  ```

  - Înregistrarea unei pierderi de viață.
  - În funcție de cele două atribute secundare ale jucătorului, va exista o șansă de 50% ca damage-ul primit să se înjumătățească.

- ```
  public int getDamage();
  ```

  - Calcularea valorii corespunzătoaredamage-ului pe care entitatea o aplică. Modul în care calculați damage-ul oferit rămâne la latitudinea voastră.
  - În funcție de atributul principal al jucătorului, va exista o șansă de 50% ca damage-ul primit să se dubleze.
  - Dacă se folosește o abilitate, la damage-ul inițial se va adăuga și damage-ul corespunzător abilității.

Observații

- Un atac obișnuit nu va costa mană.
- Sunteți liberi să alegeți formulele după care se calculează damage-ul, însă trebuie să aveți în vedere cerințele speciale de dublare/înjumătățire a valorilor menționate.

4.2 ENUMERĂRI

4.2.1 CellEntityType

Conține tipurile de celule: **PLAYER**, **VOID**, **ENEMY**, **SANCTUARY**, **PORTAL**.

4.3 CLASE

Atenție!

Conținutul claselor este minimal. Puteți adăuga orice alte atribute și metode pe care le considerați folositoare. De asemenea, puteți crea orice alte clase care să vină în ajutor, în raport cu principiile programării orientate pe obiecte.

Metodele menționate trebuie să conțină și să folosească parametrii specificați, însă puteți adăuga alte argumente dacă vă ajută în implementare.

4.3.1 Game

Clasa reprezintă jocul nostru. Aceasta are ca scop autentificarea utilizatorului, care va selecta un caracter pentru continuarea unui joc. De asemenea, clasa va implementa logica jocului și trebuie să conțină:

- Lista de conturi existente. **ArrayList<Account>**
- Harta jocului. **Grid**
- Metodă ce afișează lista de opțiuni disponibile în funcție de celula curentă și preia următoarea comandă.
- ```
  public void run();
  ```
  - încarcă datele parsate dintr-un JSON și oferă utilizatorului posibilitatea să aleagă la intrare contul și personajul cu care va începe jocul,

4.3.2 Account

Clasa conține:

- Informații despre jucător, obiect de tip **Information**.
- Listă cu toate personajele contului. **ArrayList<Character>**
- Numărul de jocuri jucate de către utilizator.

4.3.3 Information

Clasa internă clasei Account și reține următoarele informații:

- Credențialele jucătorului, obiect de tip **Credentials**.
- O colecție sortată alfabetic care reține jocurile preferate ale jucătorului.
- Informații personale despre jucător (nume, țară).

4.3.4 Credentials

Această clasă este implementată respectând **principiul încapsulării** și conține:

- Adresa de e-mail a contului.
- Parola asociată adresei de e-mail.

4.3.5 Grid

Grid este clasa care va modela tabla de joc, sub forma unei **liste de liste**. Clasa va extinde **ArrayList<ArrayList<Cell> >**. Constructorul va fi **private**, ceea ce nu va permia instanțierea directă a unui obiect de tip Grid. Această clasă va conține:

- Lungimea și lățimea tablei de joc.
- Referință la personajul din jocul curent. **Character**.
- Celula curentă a personajului. **Cell**.
- O metodă statică de generare a unei hărți, în funcție de lungimea și lățimea primite ca parametru. Metoda va întoarce o hartă ce conține cel puțin 2 sanctuare, cel puțin 4 inamici, 1 singur Portal și celula pe care se află jucătorul. Tabla va avea maxim dimensiunea 10x10.
- 4 metode de traversare a căsuțelor: **goNorth()**, **goSouth()**, **goWest()**, **goEast()**. Dacă personajul nu se poate deplasa, atunci **se va arunca o exceptie** cu un mesaj corespunzător, care va fi tratată în clasa Game.
  ```

4.3.6 Cell

Cell este clasa care va modela un pătrățel (o celulă) din tabla de joc. Aceasta conține:

- Coordonatele pe Ox și Oy în hartă.
- Enum ce definește tipul celulei. **CellEntityType**.
- Un indicator al stării căsuței (vizitată sau nevizitată).

4.3.7 Entity

Entity este o **clasă abstractă** care **implementează interfața Battle** care modelează toți războinicii jocului. Fiecare războinic (**mai puțin inamicii generați pe hartă**) au 2 atribute secundare și una principală (acestea vor fi descrise mai jos). Clasa conține:

- O listă de abilități.
- Un câmp pentru viața curentă și unul pentru cea maximă.
- Un câmp pentru mana curentă și unul pentru cea maximă.
- Imunitățile la atacurile inamicului (fire, ice, earth), reprezentate prin 3 valori **boolean**.
- Următoarele metode concrete:
  - Metodă pentru regenerarea vieții. Această metodă regenerează viața cu valoarea dată ca parametru (nu se va depăși maximul).
  - Metodă pentru regenerarea manei. Această metodă regenerează mana cu valoarea dată ca parametru (nu se va depăși maximul).
  - Metodă pentru folosirea unei abilități. După ce utilizatorul alege o abilitate din lista afișată, se apelează această metodă care primește abilitatea aleasă și inamicul curent. Dacă există suficientă mană, va folosi abilitatea împotriva acestuia (apelând mai departe metodele *receiveDamage*/*getDamage*). Inamicul primit ca parametru poate fi atât un inamic găsit într-o celulă de pe hartă (jucătorul atacă inamicul), cât și jucătorul (inamicul de pe hartă atacă jucătorul). **Această metodă nu aplică efectiv lovitura scăzând viața inamicului, ci va implementa logica pentru detectarea abilității folosite (verificare mană suficientă, verificare imunitate împotriva abilității, etc.) prin utilizarea metodelor din interfață.**

Observații

La folosirea unei abilități, vor fi afișate toate detaliile acesteia - tipul, damage-ul și costul de mană.

4.3.8 Character

Character este o clasă **abstractă care moștenește clasa Entity**. Aceasta va conține:

- Numele personajului.
- Experiența curentă, reprezentată printr-un număr întreg.
- Nivelul curent al personajului, reprezentat printr-un număr întreg.
- 3 câmpuri care definesc atributele ce vor influența damage-ul primit și oferit de personaj:
  - Strength (putere)
  - Charisma (carismă)
  - Dexterity (dexteritate)

Sunteți liberi să creați formulele după care evoluează aceste atribute, precum și damage-ul oferit unui inamic, așa cum doriți (astfel încât jocul să evolueze într-o manieră echilibrată și captivantă.

4.3.9 Warrior-Mage-Rogue

Clasele **Warrior, Mage și Rogue** vor **extinde** clasa Character, fiecare având proprietăți diferite, menționate în tabelul de mai jos.

Caracter	Imunitate	Atribut principal
Warrior	Fire	Strength
Rogue	Earth	Dexterity
Mage	Ice	Charisma

Clasele vor implementa metodele definite în superclasă:

- Metoda ce înregistrează o pierdere de viață ținând cont de atributele secundare ale personajului.
- Metoda de calculare a valorii damage-ului ținând cont de atributul primar al personajului.

Observații

Veți alege voi formulele care să țină cont de atributele primare, respectiv secundare și să se scaleze bine odată cu creșterea atributelor în urma creșterii nivelului personajului.

4.3.10 Enemy

Clasa care modelează tipul de inamici din joc. Aceasta va **extinde clasa abstractă Entity**. La instanțierea unui inamic nou, acestuia i se vor seta viața, mana și damage-ul oferit cu un atac normal cu valori random situate într-un interval ales de voi, iar cele 3 valori de imunitate (pentru fiecare element - foc, gheață, pământ) vor primi o valoarea aleatoare. De asemenea, se vor instanția 3-6 abilități alese aleator dintre cele 3 tipuri (descrise în cele ce urmează). Clasa va implementa metodele din superclasă:

- Metoda ce înregistrează o pierdere de viață. Există o șansă de 50% să evite damage-ul.
- Metoda de calculare a valorii damage-ului. Există o șansă de 50% ca să dea damage dublu.

Observații

Inamicii nu au atribute speciale.

4.3.11 Spell

Spell este o **clasă abstractă** care modelează abilitățile din joc și conține o valoare pentru damage-ul pe care îl oferă abilitatea inamicului și una pentru costul de mană necesar utilizării acesteia. **Dacă jucătorul/inamicul nu are suficientă mană pentru a folosi o abilitate, se va folosi automat un atac default**.

Clasa conține o metodă pentru a ajuta la afișarea detaliilor abilității.

```
public String toString();
```

Observații

- De fiecare dată când începe o luptă, listele de abilități (ale jucătorului și inamicului) vor fi populate cu un număr generat aleator de abilități, cuprins între 3 și 6.
- Tipul abilităților va fi generat aleator (minim o abilitate de fiecare tip), la fel și costul manei sau damage-ul provocat.
- Inamicul va ataca aleator cu o abilitate (dacă are suficientă mană) sau cu un atac normal.
- După folosirea unei abilități, aceasta se va șterge din listă.

Atenție!

Se va folosi clasa **Random** pentru generarea numerelor.

4.3.12 Ice-Fire-Earth

Aceste clase vor **moșteni** clasa Spell și vor apela constructorul superclasei.

5 Excepții

Pentru ca jocul să funcționeze corect în toate cazurile, va trebui să tratați anumite excepții care pot apărea în timpul jocului. Astfel, se vor arunca următoarele excepții, cu mesaje sugestive:

- `InvalidCommandException` - dacă utilizatorul introduce o comandă invalidă;
- `ImpossibleMove` - dacă se încearcă mutarea jucătorului în afara tablei de joc (depășirea marginilor hărții).

Atenție!

Veți pierde puncte dacă în timpul utilizării jocului apar excepții pe care nu le tratați.

Sunteți liberi să adăugați orice alte tipuri de excepții.

În practică, un utilizator poate folosi greșit jocul (având în vedere că se bazează pe un input al utilizatorului) și se poate ajunge la un comportament imprevizibil sau erori ale jocului. De exemplu, utilizatorul poate introduce un șir de caractere acolo unde se așteaptă un număr întreg. Mai departe, în aplicație, acel șir de caractere este folosit într-o ecuație matematică, ceea ce ar produce o eroare. Trebuie să vă asigurați că nu apar astfel de cazuri în timpul jocului **(programare defensivă)**.

6 Fișiere de intrare

Pentru implementarea temei, se va folosi fișierul JSON pus la dispoziție.

- **accounts.json** - toate detaliile necesare despre toți utilizatorii creați.

În arhiva temei se află o clasă care are implementată deja logica de parsare a fișierului. Puteți apela metodele din această clasă, sau vă puteți crea propriul mod de citire din fișier.

Atenție!

Dacă folosiți fișierul de parsare din arhivă, aveți grijă să schimbați calea către fișierul JSON.

Metoda de parsare returnează un **ArrayList** cu obiecte de tip **Account** și va funcționa doar dacă sunt implementate clasele din cerință.

Pentru manipularea datelor în format JSON, recomandăm folosirea bibliotecii json-simple, care poate fi descărcată accesând acest **link**.

7 Observații

Atenție!

- **Tipizați** orice colecție folosită în cadrul implementării.
- Respectați specificațiile detaliate în enunțul temei și folosiți indicațiile menționate.
- Puteți adăuga orice alte detalii și funcționalități pentru aplicație.
- Puteți aduce modificări asupra fișierelor de input (de exemplu, pentru adăugarea unor flaguri care v-ar fi utile), dar fără a modifica detaliile deja existente.
- În programarea orientată pe obiecte, **încapsularea** este un principiu fundamental, folosit pentru ascunderea detaliilor și protecția datelor. Deși este considerată o practică profesionistă în dezvoltarea software-ului, nu impunem ca toate clasele arhitecturii să fie implementate folosind acest principiu.
- Pentru orice fel de întrebări, puteți folosi **forumul.**
- **Toate elementele care nu sunt specificate în cerință rămân la alegerea voastră.**

8 Testare

Atenție!

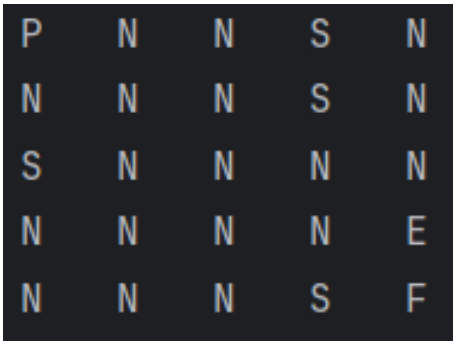
- Fiecare componentă creată trebuie să fie testată.
- **Nu se va puncta ierarhia de clase, fără ca acestea să fie testate**.
- **Va trebui să vă creați una sau mai multe metode main care testează toate funcționalitățile claselor implementate, altfel nu veți primi punctajul!**

Pentru a putea realiza o testare a aplicației, trebuie **să implementați o clasă Test** care va reprezenta **entry-point-ul aplicației** (funcția main).

La rulare, se va apela metoda run din clasa Game, unde se va face citirea și parsarea fișierului JSON care conține personajele.

Pentru testare aveți de implementat următorul scenariu:

1. Se va alege un cont și un personaj din fișierul de intrare parsat.
2. Se va genera harta cu următoarea configurație:



   Semnificația simbolurilor:
   - **P** - player-ul;
   - **N** - nevizitată (**inițial, toate celulele, în afară de celula jucătorului, vor fi N**);
   - **V** - celulă vizitată (pustiu, void)
   - **S** - sanctuary;
   - **E** - enemy;
   - **F** - celula de final (portal).
3. Se va deplasa jucătorul 3 celule la dreapta.
4. Se vor reîncărca viața și mana jucătorului.
5. Se va deplasa jucătorul o căsuță la dreapta și 3 căsuțe în jos.
6. În lupta cu inamicul, se vor utiliza toate abilitățile disponibile, iar restul atacurilor vor fi atacuri normale.
7. Se va deplasa jucătorul o căsuță în jos, ajungându-se pe celula de sfârșit.

Atenție!

- Scenariul descris anterior va fi "hardcodat" folosind metodele implementate anterior.
- Implementarea voastră trebuie să permită crearea unei hărți de dimensiuni generate aleator, cu plasarea random a entităților pe hartă.

9 Punctaj

Cerința	Puncte
Implementarea integrală a arhitecturii și funcționalităților propuse și testarea acestora	0.7

Atenție!

- Tema (prima parte) valorează **0.7 puncte** din nota finală a disciplinei POO!
- Tema este **individuală**! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.
- **Se vor puncta DOAR funcționalitățile care pot fi testate.** Acest lucru înseamnă că va trebui să aveți o clasă de test prin care **să oferiți** posibilitatea **testării** tuturor funcționalităților pe care le-ați implementat, **altfel nu veți primi punctaj**.
- Tema se va încărca pe site-ul de cursuri până la termenul specificat în pagina de titlu. Se va trimite o arhivă **.zip** ce va avea un nume de forma **grupa_Nume_Prenume_tema1.zip** (ex. 326CC_Popescu_Andreea_tema1.zip) și care va conține următoarele:
  - un folder **SURSE** ce conține doar sursele Java;
  - un folder **PROIECT** ce conține proiectul în mediul ales de voi (de exemplu NetBeans, Eclipse, Intellij IDEA);
  - un fișier **README.pdf** în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți specifica, pe scurt, în ce a constat dificultatea implementării.
- Lipsa fișierului README sau nerespectarea formatului impus pentru arhivă duce la o **depunctare** de **0.1** (fiecare) din punctajul temei.