Programare orientată pe obiecte
Tema - World Of Marcel

Responsabili temă: Andrei Voicu, Geani Dumitrache, Doina Chiroiu

23 decembrie 2021

Deadline: 16 Ianuarie 2022 Ora 23:55

Ultima Modificare: 23 Decembrie 2021

Facultatea de Automatică și Calculatoare

Universitatea Politehnică din București

An universitar 2021-2022

Seria CC

1 Obiectivele temei

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 ce se poate realiza folosind noțiunile dobândite în cadrul cursului.

Tema constă în implementarea unui joc bazat pe text (text adventure game). Jocul este sub forma unei matrice de dimensiune n x m, fiecare celulă putând conține diferite elemente (inamici, magazine de aprovizionare cu poțiuni etc.). Astfel, programul implementat oferă jucătorului anumite alegeri în funcție de tipul celulei și afișează la terminal/GUI un mesaj corespunzător.

Jucătorul are la dispoziție mai multe personaje și poate crea câte dorește (când pornește jocul va selecta cu ce personaj dorește să înceapă un nivel nou).

Fiecare personaj are atribute și abilități unice. Personajele vor evolua în timp, după fiecare eveniment câștigând experiență, cuantificată printr-un număr. La atingerea unor cote stabilite de voi, jucătorul va avansa ca nivel, astfel crescandu-i atributele (puterea, dexteritatea, carisma).

Flow-ul jocului

1. Inițializări

a. Utilizatorul alege ce mod de joc va deschide (terminal sau fereastră grafică)

b. Se alege un cont

c. Se alege un personaj de pe contul respectiv

d. Se generează harta și se va poziționa jucătorul pe o celulă goală

2. Loop / Execuție continuă

a. Se afișează harta generată in fiecare rundă în care jucătorul se poate deplasa (nu se va afișa harta când se inspectează magazinul sau când jucătorul este în luptă)

b. Dacă jucatorul se află pe o celulă magazin se va afișa o listă de poțiuni pe care acesta le poate cumpăra, iar daca se află pe o celulă inamic se vor afișa opțiunile: atacă inamic, folosește abilitate, folosește poțiune (lupta va fi pe ture, alternativ, după fiecare alegere a personajului urmează o alegere a inamicului; inamicul va putea ataca sau folosi abilitate, șanse alese de voi; daca nu va avea destulă mană să folosească o abilitate atunci va ataca normal).

c. Se citește următoarea mutare de la terminal

d. Se face mutarea (dacă celula este nevizitată se va afișa o poveste nouă)

e. Repetă

3. Final

Jocul se termină în momentul în care jucătorul găsește celula de final sau cand viața jucătorului ajunge la 0. Inițial celulele vor fi afișate identic, urmând ca pe măsură ce personajul le-a atins acestea să fie descoperite (dacă personajul se întoarce într-o căsuță cu inamic vizitată deja, inamicul nu va reapărea).

2 Cerințe

2.1 Cerința 1. Arhitectura aplicației si Testare - (120 puncte)

Game

Game este clasa care va reprezenta jocul nostru.

Clasa va conține:

- lista de conturi in care s-au autentificat jucatorii / $eg. List < Account >$
- un dicționar având drept cheie tipul celulei și drept valoare o listă cu șiruri ce modelează poveștile / $eg. Map < CellEnum, List < String >>$
- metodă de **run** care pornește un joc nou
  Metoda încarcă datele parsate dintr-un JSON și va oferi utilizatorului posibilitatea să aleagă la intrare contul și personajul cu care va începe jocul. Dacă a fost selectat modul text atunci se va rula un set de instrucțiuni predefinite, altfel va deschide fereastra cu interfața grafică.
- metodă ce afișează lista de opțiuni disponibile în funcție de celula curentă și preia urmatoarea comandă
- metodă ce afișează o poveste pentru căsuța curentă
  Se va afișa o poveste, dacă celula nu a fost vizită înainte, în funcție de tipul acesteia.

  Vom utiliza șablonul Singleton cu **instanțierea întârziată** pentru a restricționa numărul de instanțieri ale clasei Game. Mai multe detalii în secțiunea "Șabloane de proiectare".

Account

Account este o clasă care va conține:

- informații despre jucator, obiect de tip Information
- listă cu toate personajele contului
- numărul de jocuri jucate de către utilizator

Information

Information este o **clasă 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)

Se va folosi șablonul Builder pentru a instanția un obiect de tip Information. Mai multe detalii în secțiunea "Șabloane de proiectare".

Credentials

Credentials este o clasă care conține:

- datele de autentificare ale utilizatorului (email și parolă)
  Această clasă va fi implementată respectând principiul încapsulării.

Grid

Grid este clasa care va modela tabla de joc, sub forma unei liste de liste. Clasa Grid moștenește **ArrayList**. Constructorul clasei Grid va fi **privat** (nu veți putea instanția direct un obiect de tip Grid).

Clasa va conține:

- lungimea și lățimea tablei de joc
- referință la personajul din jocul curent
- 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 magazine si 4 inamici.
- celula curentă a personajului
- 4 metode de traversare a căsuțelor / *eg.goNorth(), goSouth(), goWest(), goEast()*
  Dacă personajul nu se poate deplasa, atunci se va afișa un mesaj corespunzător.

Cell

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

Clasa va conține:

- coordonatele pe Ox și Oy în hartă
- enum ce definește tipul celulei
- un obiect de tip CellElement care conține inamicul/magazinul din căsuța respectivă (EMPTY, ENEMY, SHOP, FINISH)
- un indicator al stării căsuței (vizitată / nevizitată)

CellElement

CellElement este o interfață implementată de către Enemy și Shop. Aceasta reprezintă elementul aflat pe pătrățelul respectiv. Interfața va defini:

- o metodă ce întoarce caracterul de afișat în terminal sau GUI / *eg.toCharacter*()

Entity

Entity este o **clasa abstractă** care 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
- protecțiile la elementele din joc (fire, ice, earth), reprezentate prin 3 valori boolean
- urmatoarele metode concrete:
  - funcție pentru regenerarea vieții
    Regenerează viața cu valoarea dată ca parametru (nu se va depăsi maximul).
  - funcție pentru regenerarea manei
    Regenerează mana cu valoarea dată ca parametru (nu se va depăsi maximul).
  - funcție pentru folosirea unei abilități
    Funcția primește abilitatea aleasă și inamicul curent. Dacă există suficientă mană, va folosi abilitatea împotriva acestuia.

- următoarele metode abstracte (vor fi implementate ulterior de către clasele Enemy, Rogue, Warrior si Mage):
  - înregistrarea unei pierderi de viață / eg.receiveDamage(int)
  - calcularea valorii corespunzătoare damage-ului pe care entitatea o aplică / eg.getDamage()

    Formulele dupa care se vor calcula pierderile rămân la latitudinea voastră.

Character

Character este o **clasă abstractă care extinde clasa Entity**.

Clasa va conține:

- numele personajului
- coordonatele curente ale personajului
- un obiect de tip Inventory, care să gestioneze lista de poțiuni
- experiența curentă, reprezentată printr-un număr întreg
- nivelul curent al personajului, reprezentat printr-un număr întreg
- 3 câmpuri care definesc atributele: putere (strength), carismă (charisma) și dexteritate (dexterity), calculate în funcție de nivelul curent (formula rămâne la alegerea voastră).
  Acestea vor influența damage-ul primit și dat ale personajului.
- o metodă de cumpărare a unei poțiuni (va verifica dacă există bani suficienți și spațiu suficient în inventar pentru poțiunea primită ca parametru)

Warrior-Mage-Rogue

Clasele Warrior, Mage și Rogue vor extinde clasa Character, fiecare având proprietăți diferite.
Warrior: deține cea mai mare greutate maximă a inventarului, este imun la foc (fire), iar atributul lui principal este puterea (strength)
Rogue: deține o greutate medie a inventarului, este imun la pământ (earth), iar atributul lui principal este dexteritatea (dexterity)
Mage: deține cea mai mică greutate a inventarului, este imun la gheață (ice), iar atributul lui principal este carisma (charisma)

Clasele vor implementa:

- funcția ce înregistrează o pierdere de viață ținând cont de atributele secundare ale personajului (cele care nu sunt primare; va exista o șansă ca damage-ul primit sa fie înjumătățit în funcție de atributele secundare) (receiveDamage(int))
- funcția de calculare a valorii damage-ului ținând cont de atributul primar al personajului (va exista o șansă ca damage-ul dat sa fie dublu în funcție de atributul primar) (getDamage())

Veți alege voi o formulă care să țină cont de atributele primare, respectiv secundare și să se scaleze bine odată cu creșterea atributelor în urma creșterii nivelului personajului (veți alege și cu ce valori incrementați per nivel).

Vom utiliza șablonul Factory pentru a instanția personajele din lista contului. Mai multe detalii în secțiunea "Șabloane de proiectare".

Spell
Spell este o clasa abstractă care modelează abilitățile din joc si conține o valoare pentru damage și una pentru costul de mană necesar utilizării acesteia.

Ice-Fire-Earth

Aceste clase vor moșteni clasa Spell și vor defini efectul pe care îl au acestea asupra entităților.

Enemy
Clasa care modelează tipul de inamici din joc. Aceasta va **extinde clasa abstractă Entity** și **implementează interfața CellElement**. La instanțierea unui inamic nou i se va seta viața și mana cu valori situate într-un interval ales de voi, iar cele 3 valori de protecție (pentru fiecare element) vor primi o valoare aleatoare. De asemenea, se vor instanția 2-4 abilități alese aleator dintre cele 3 tipuri.

Pe langă această clasă se vor implementa următoarele metode:

- funcția ce înregistrează o pierdere de viață (există o sansă de 50% să evite damage-ul) (receiveDamage(int))
- funcția de calculare a valorii damage-ului (există o sansă de 50% ca sa dea damage dublu; puteți alege o valoare fixă pentru damage-ul inițial sau situată intr-un interval) (getDamage())

Inventory
Clasa va conține:

- o listă de poțiuni
- greutatea maximă a inventarului
- numărul de monede (la fiecare casută liberă nevizitată există o sansă de 20% să se găsească monede; inamicii au o sansă de 80% să ofere monede in urma înfrangerii acestora)
- 3 metode de lucru pe lista de poțiuni:
  - pentru adăugarea unei poțiuni în listă
  - pentru eliminarea unei poțiuni din listă
  - pentru calcularea greutății rămase

Potion

Este o interfață care definește metodele:

- de utilizare a poțiunii (regenerează viața sau mana în funcție de tipul poțiunii)
- de preluare a prețului
- de preluare a valorii de regenerare
- de preluare a greutății poțiunii

HealthPotion si ManaPotion
Clasele HealthPotion și ManaPotion implementează interfața Potion și adaugă câmpurile corespunzătoare prețului, greutății și valorii de regenerare (valori pe care le veți alege voi la instanțiere).

Shop

Clasa Shop va implementa interfața CellElement și va conține o listă de poțiuni care este populată în momentul instanțierii unui obiect nou (se vor instanția între 2-4 poțiuni de tip aleator). Va oferi următoarea metodă:

- selectarea unei poțiuni (în funcție de un index) din lista și eliminarea ei (întoarce poțiunea selectată)

InvalidCommandException
Se va arunca o excepție de tipul InvalidCommandException în momentul citirii unei comenzi invalide de la terminal.

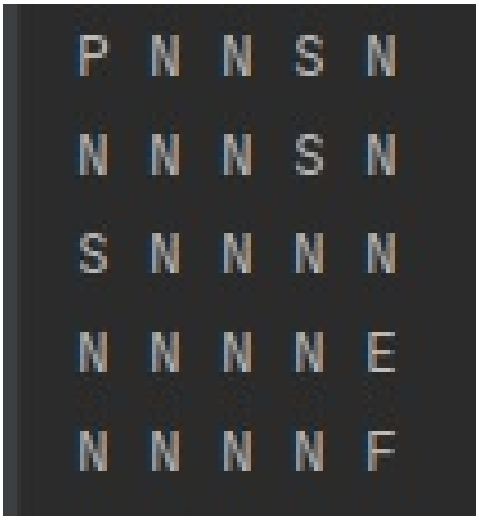
Testare

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

La rulare, se va putea selecta de la tastatură dacă aplicația va rula în terminal sau în interfață grafică. Se va apela funcția run din clasa Game, unde în funcție de modul de joc se va face **citirea și parsarea unor fișiere JSON** care conțin personaje și povești (puteți folosi orice parser de JSON și să vă creați propria clasa wrapper pentru a modulariza citirea din fișiere).

Veți găsi fișierele JSON care conțin conturile cu personaje și povești in arhiva temei. 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:



3. Se va deplasa jucătorul 3 căsuțe la dreapta.
4. Se va cumpăra o poțiune de mană și una de viață de la magazin.
5. Se va deplasa jucatorul o căsuță la dreapta și 3 căsuțe in jos.
6. In lupta cu inamicul se vor utiliza toate abilitațile disponibile, se vor folosi cele 2 poțiuni, iar restul atacurilor vor fi atacuri normale.
7. Se va deplasa jucătorul o căsuță in jos, ajungandu-se pe celula de sfarșit.

Trecerea la următoarea mutare se va face la apăsarea tastei P (scenariul cerut va fi "hardcodat" folosind metodele implementate anterior).

Va recomandăm următorul pachet JSON:
`https://github.com/stleary/JSON-java`

Cateva exemple de reprezentare a hărții:

Figura 1: Harta la începutul jocului

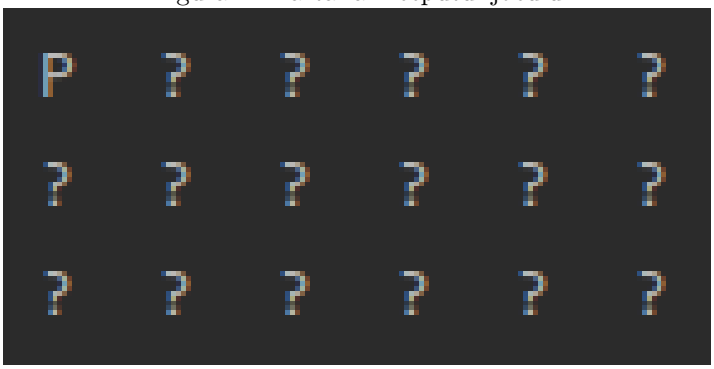
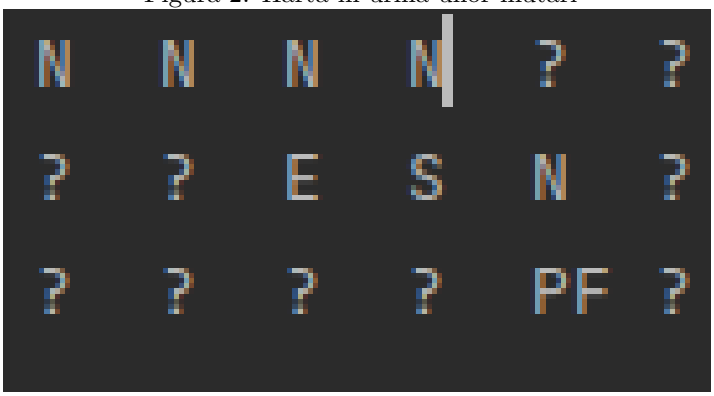


Figura 2: Harta în urma unor mutări



Atenție!

Punctajul pentru Cerința 1 se va acorda numai daca partea de testare este implementată.
În cazul în care nu s-a implementat partea de testare, se poate acorda punctaj parțial doar dacă există clase de test pentru fiecare funcționalitate a aplicației.

2.2 Cerința 2. Șabloane de proiectare - (50 puncte)

Singleton Pattern

Vom utiliza șablonul Singleton pentru a restricționa numărul de instanțieri ale clasei Game. Având o singură instanță a clasei, va fi facilitată accesarea informațiilor comune.

Se va utiliza **instanțierea întârziată**.

Builder Pattern

Acest șablon dorește separarea construcției de obiecte complexe de reprezentarea lor astfel încât același proces să poată crea diferite reprezentări. Builder-ul creează părți ale obiectului complex de fiecare dată când este apelat și reține toate stările intermediare. Când obiectul are toate câmpurile completate, utilizatorul primește rezultatul de la Builder. În acest mod, se obține un control mai mare asupra procesului de construcție de noi obiecte. Astfel, în comparație cu alte pattern-uri din categoria creational, Builder construiește un obiect pas cu pas la comanda utilizatorului.

În cadrul acestei aplicații, pattern-ul este folosit pentru a instanția un obiect de tip Information. Trebuie să vă asigurați că veți putea instanția, folosind mecanismul implementat pe baza pattern-ului Builder, orice tip de Information. Dacă se încearcă realizarea unui obiect Information fără credențiale sau fără nume se va arunca o excepție de tipul **InformationIncompleteException**.

Factory Pattern

Șablonul Factory este unul dintre cele mai utilizate tipuri de șabloane de proiectare din Java. Acesta face parte din categoria celor creational, oferind una dintre cele mai bune modalități de instanțiere pentru obiecte ce au tipuri care provin din aceeași ierarhie de clase.

La începutul unui joc nou se va folosi Factory Pattern pentru a instanția personajele din lista contului.

Visitor Pattern

Șablonul Visitor este folosit pentru a modela efectul pe care îl au abilitățile asupra entităților. Astfel, în funcție de elementul abilității, entitatea se poate proteja de damage-ul abilității.

public interface Element $<TextendsEntity>$

— void accept(Visitor< T >$visitor$);

public interface Visitor$<TextendsEntity>$

— void visit(T entity);

În jocul nostru Entity implementează Element, iar Spell Visitor.

2.3 Cerința 3. Interfață grafică - (80 puncte)

Va trebui să realizați o interfață grafică folosind componenta Swing.

Această interfață va trebui să cuprindă următoarele pagini:

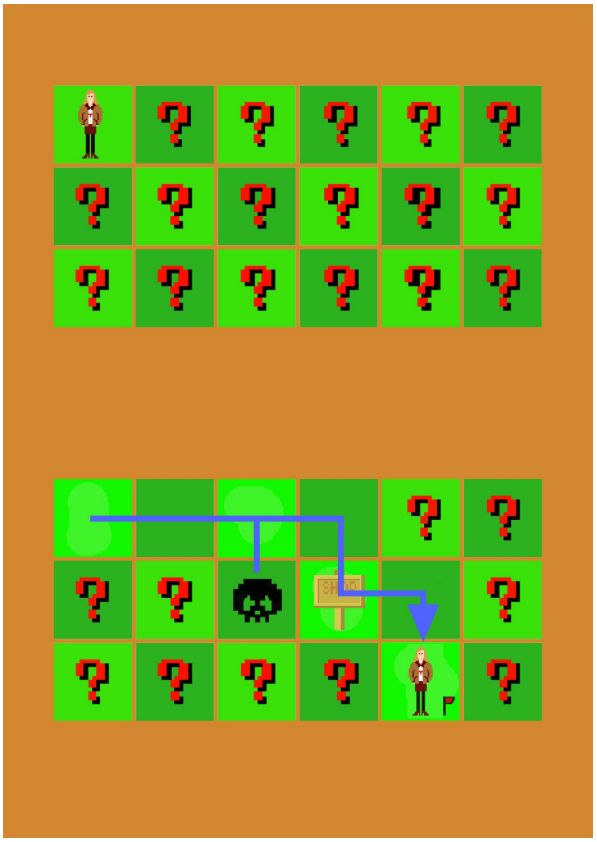
Pagină de autentificare.
: În urma autentificării se poate selecta personajul de pe contul respectiv și începerea unui joc.

Pagină principală a jocului.
: Utilizatorului îi sunt prezentate alegerile disponibile pentru mutarea curentă și poate să se deplaseze, având o reprezentare grafică a hărții sub forma unui grid (se vor afișa în permanență viața, mana, experiența, nivelul și alte informații utile legate de jucător și de inamicul curent).

  Idei de implementare: puteți folosi GridLayout și butoane/label-uri pentru reprezentarea hărții sau o combinație de JTable cu un renderer custom; se acceptă și alte implementări.

Pagină finală.
: Se va afișa progresul personajului de la finalul nivelului (experiența câștigată, nivelul la care a ajuns, numărul de inamici doborâți, numărul de monede dobândite .).

Pentru grafica butoanelor sau pentru alte imagini integrate în joc vă recomandăm următorul site: `https://iconscout.com/unicons/explore/line`



2.4 BONUS - (50 puncte)

Propuneri:

- salvarea progresului in JSON
- crearea de conturi/personaje noi prin intermediul interfeței grafice

Orice funcționalitate suplimentară utilă din punct de vedere al interfeței grafice va fi punctată ca bonus, în funcție de complexitatea acesteia.

3 Notare

Cerința	Punctaj
Cerința1 (Implementarea integrală a claselor compuse + Testare)	120 puncte
Cerința2 (Integrarea design pattern-urilor propuse)	50 puncte
Cerința3 (Realizarea paginilor propuse pentru interfața grafică)	80 puncte
BONUS	50 puncte

Atenție!

Tema va valora **2.5 puncte** din nota finală a disciplinei POO!
Soluțiile de genul folosirii de variabile interne pentru a stoca informații ce țin de tipul obiectului vor fi depunctate.
Tipizați orice colecție folosită în cadrul implementării. Respectați specificațiile detaliate în enunțul temei și folosiți indicațiile menționate. Pe lângă clasele, atributele si metodele specificate în enunț, puteți adăuga altele dacă acest lucru îl considerati util și potrivit, în raport cu principiile programării orientate pe obiecte.

Atenție!

Tema este individuală!

Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.
Tema se va prezenta în ultima săptămână din semestru. 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.zip
(eg.326CC_Popescu_Andreea.zip) și care va conține următoarele:

1. un folder cu SURSE, ce conține doar sursele Java și fișierele de test
2. un folder PROIECT ce conține proiectul în mediul ales de voi (eg.NetBeans, Eclipse, Intellij IDEA)
3. un fișier README.pdf, în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți detalia modul de implementare cu observații unde este cazul.

Lipsa acestui fișier sau nerespectarea formatului impus pentru arhivă duc la o depunctare de 10 puncte.