Programare orientată pe obiecte

Tema - League of Warriors

Partea a II-a

Data postării: 11.12.2024

Deadline: 10.01.2025 ora 23:55

Ultima modificare: 11.12.2024 ora 17:30

Echipă temă: Carmen ODUBĂȘTEANU, Ștefănel TURCU, Alexandru TUDOR

Facultatea de Automatică şi Calculatoare

Universitatea Națională de Știință și Tehnologie Politehnica București

Anul universitar 2024 - 2025

Seria CC

1 Descriere

Tema constă în implementarea unui joc de aventură. 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.

După autentificare, jucătorul 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
   - Se va intra în interfața de login 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 alege următoarea mutare pe hartă, din interfața grafică.
   - 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 Șabloane de proiectare

4.1 Singleton Pattern

Singleton este un design pattern din categoria **Creational Patterns**, folosit pentru a asigura că o clasă are o singură instanță și pentru a oferi un punct de acces global la acea instanță. Acest pattern este utilizat în situații în care este necesar un obiect unic care să fie accesibil de oriunde în program, precum un logger global, o conexiune la baza de date sau un obiect de configurare.

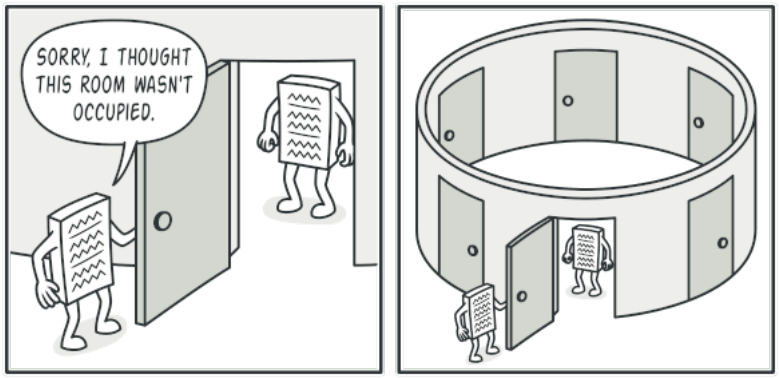

Figura 1: Acces la instanța centrală - Modelul Singleton

Toate implementările șablonului Singleton includ urmează aceeași structură:

- Clasa definește constructorul ca fiind **private**, astfel încât nu poate fi instanțiată din afara clasei.
- Clasa furnizează o metodă **statică** ce are ca scop returnarea instanței unice a clasei.
- Instanța unică a clasei este stocată într-o variabilă statică, ceea ce permite ca aceasta să fie accesibilă de oriunde.

Lazy initialization (sau inițializarea întârziată) este o tehnică de programare prin care crearea și inițializarea unui obiect sunt amânate până când este efectiv necesar în timpul execuției. Această abordare este folosită pentru a economisi resurse și pentru a reduce timpul de inițializare al aplicației, creând obiectele doar atunci când sunt folosite pentru prima dată.

Observații

Trebuie să utilizați acest șablon, împreună cu inițializarea întârziată, pentru a restricționa numărul de instanțieri ale clasei **Game**.

4.2 Builder Pattern

Builder este un design pattern din categoria **Creational Patterns**, utilizat pentru a separa construirea unui obiect complex de reprezentarea sa, permițând astfel crearea treptată și personalizată a obiectului final. Acest pattern este util în situații în care un obiect poate fi construit în mai multe etape sau prin combinarea diverselor componente, ca în cazul unui obiect cu numeroși parametri.

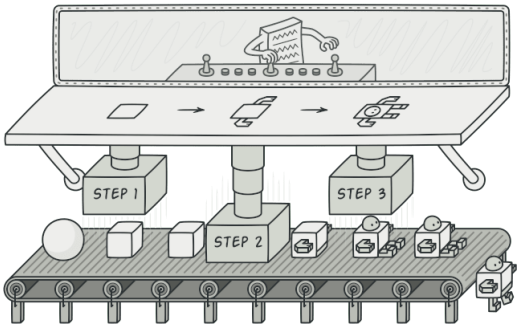

Figura 2: Construcția pas-cu-pas a unui obiect complex - Modelul Builder

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 prin apeluri succesive ale metodelor acestuia, fiecare metodă configurând o parte a obiectului. În acest mod, se obține un control mai mare asupra procesului de construcție de noi obiecte. Cu alte cuvinte, Builder construiește un obiect pas cu pas la comanda utilizatorului.

Observații

În cadrul acestei aplicații, pattern-ul va fi 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.

4.3 Factory Pattern

Factory este un design pattern din categoria **Creational Patterns**, folosit pentru a crea obiecte fără a specifica exact clasa de instanțiere. Este utilizat în situații în care crearea obiectelor poate fi delegată unei metode specializate, iar tipul exact al obiectului depinde de anumite condiții sau de configurațiile programului.

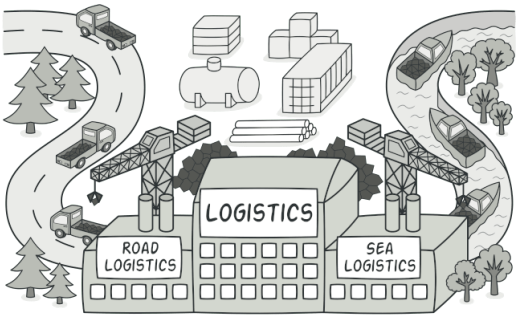

Figura 3: Crearea obiectelor fără a expune detalii - Modelul Factory

Observații

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

4.4 Visitor Pattern

Visitor este un design pattern din categoria **Behavioral Patterns**, care permite adăugarea de noi comportamente unui obiect fără a face modificări asupra clasei. Acest pattern este folosit pentru a separa logica comportamentală de structura obiectelor, oferind posibilitatea de a adăuga operațiuni pe obiecte fără a le modifica.

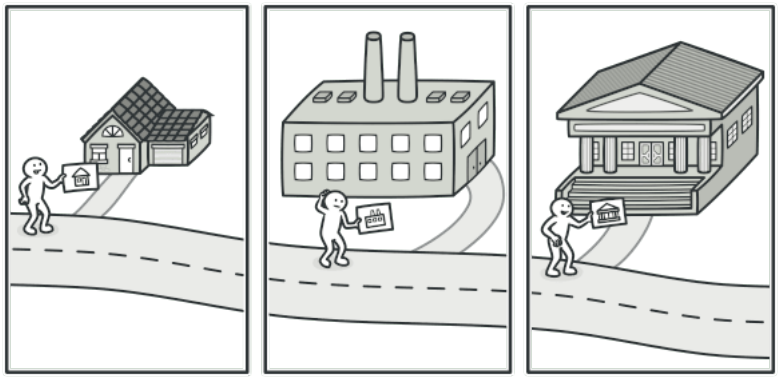

Figura 4: Comportament diferit în funcție de tipul obiectelor - Modelul Visitor

În cadrul jocului, fiecare entitate poate utiliza abilități ce au efecte specifice asupra altor entități. Fiecare abilitate poate avea un efect diferit în funcție de imunitatea entității țintă, precum și de tipul abilității. Șablonul Visitor va fi utilizat pentru a modela efectele abilităților asupra entităților.

Va trebui să creați următoarele interfețe:

- `public interface Element <T extends Entity>`

  - Conține metoda **void accept(Visitor<T> visitor);**

- `public interface Visitor <T extends Entity>`

  - Conține metoda **void visit(T entity);**

Observații

În jocul nostru, **Entity implementează Element**, iar **Spell implementează Visitor**.

5 Interfață grafică

Va trebui să realizați o interfață grafică a jocului, folosind pachetul `SWING`. Această interfață trebuie să cuprindă **obligatoriu** următoarele pagini:

- Pagină de autentificare.
  - În urma autentificării se poate selecta unul dintre personajele de pe contul respectiv și începerea unui joc.
- Paginile de joc.
  - Utilizatorului îi sunt prezentate alegerile disponibile pentru mutarea curentă (deplasare, atac inamic sau folosire abilitate dacă este cazul) 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 inamicul curent.
- Pagina finală.
  - Se va afișa progresul personajului de la finalul nivelului (experiența câștigată, nivelul la care a ajuns, numărul de inamici doborâți, etc..).

Observații

- Pentru afișarea hărții puteți folosi GridLayout și butoane/label-uri pentru reprezentarea hărții sau o combinație de JTable. **Se acceptă și alte implementări.**
- Pentru grafica butoanelor sau pentru alte imagini integrate în joc, vă recomandăm următorul site: https://iconscout.com/unicons/free-line-icons.
- Se vor puncta creativitatea și design-ul, precum și folosirea de imagini sugestive (de exemplu, pentru inamici).
- Sunteți liberi să adăugați orice funcționalități suplimentare pe care le considerați utile din punct de vedere al interfeței grafice.
- Puteți utiliza biblioteci suplimentare pentru a crea un aspect mai estetic, de exemplu pentru manipularea imaginilor. Orice lucru nou pe care îl utilizați trebuie menționat în README (ce ați folosit și în ce mod).

Atenție!

Folosirea jocului în terminal este permisă **strict** pentru selectarea tipului de joc în interfața grafică și pentru afișarea mesajelor de informare. Orice altă acțiune efectuată în terminal nu va fi punctată în interfața grafică.

6 Exemplu de interfață grafică

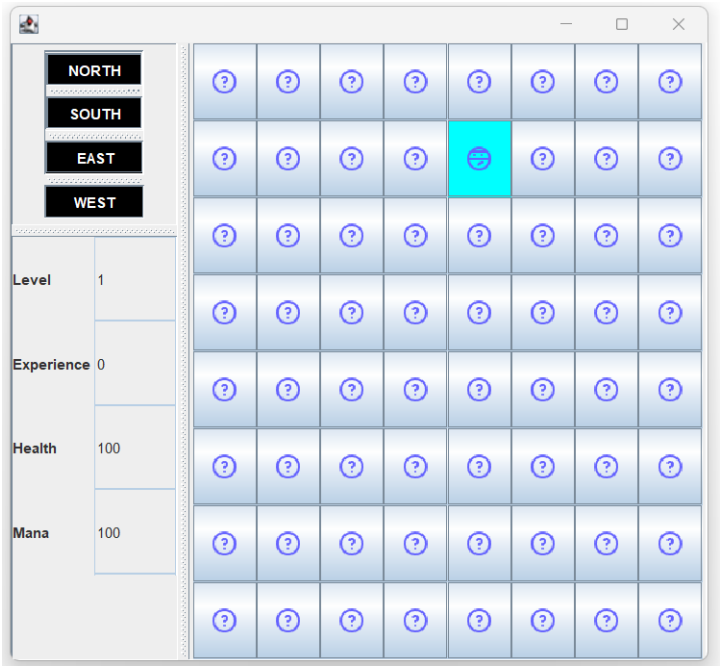

Figura 5: Pagina principală a jocului

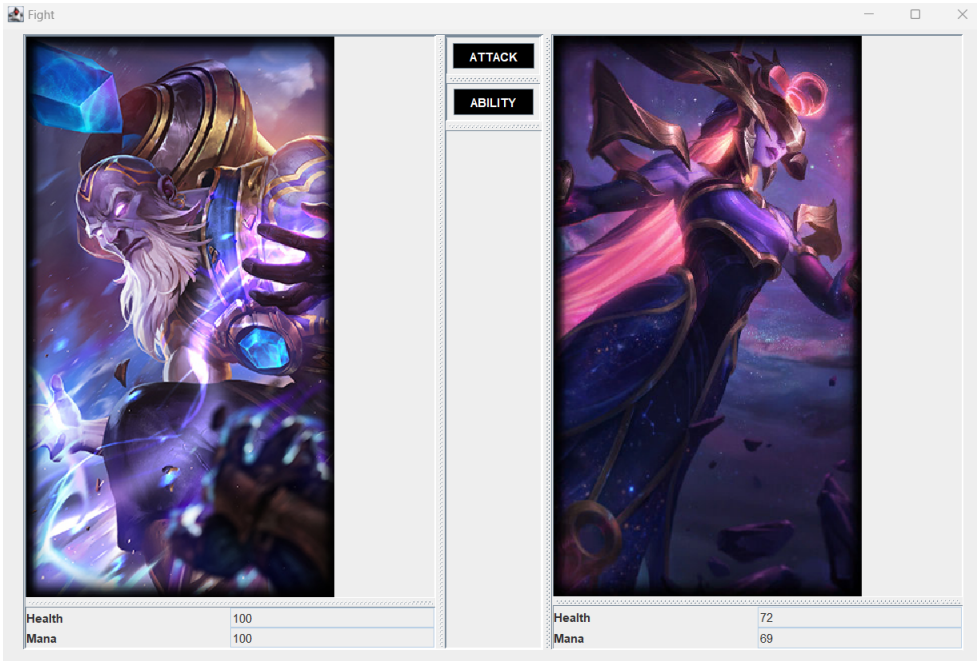

Figura 6: Pagină pentru lupta cu inamicul

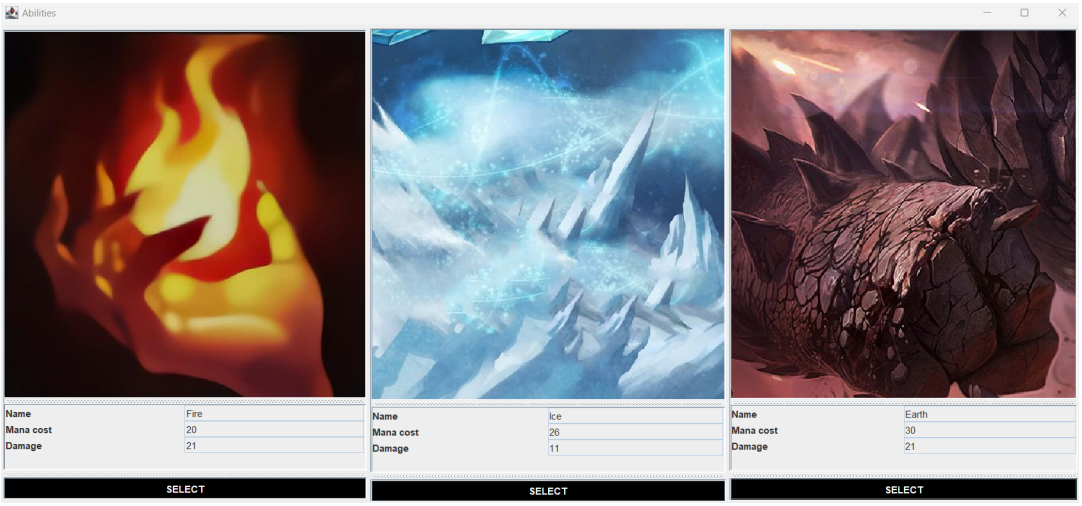

Figura 7: Pagină pentru folosirea unei abilități

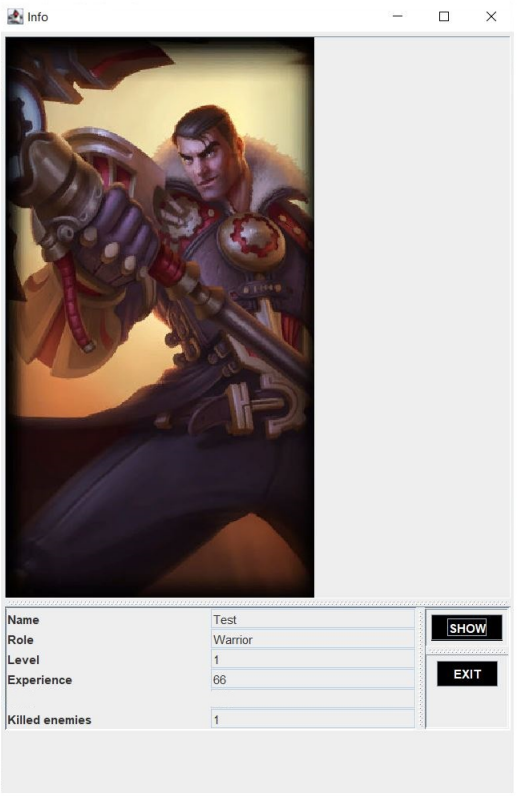

Figura 8: Pagină de final

7 Punctaj

Șablon de proiectare	Puncte
Singleton	0.05
Factory	0.05
Builder	0.1
Visitor	0.1
Total	**0.3**

Interfață grafică	Puncte
Pagina de autentificare	0.05
Paginile de joc	0.3
Pagina de sfârșit	0.05
Total	**0.4**

Atenție!

- Tema (a II-a 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_tema2.zip** (ex. 326CC_Popescu_Andreea_tema2.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.