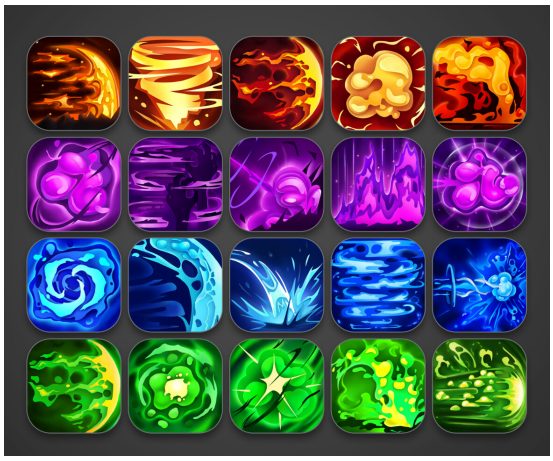
Laborator 10 – Vectori de obiecte neomogene

Tema: RPG Skills 🪄

Autor: ing. Eduard Donea



Poveste

În universul unui joc RPG, un erou trebuie să se bazeze pe abilități magice diverse pentru a supraviețui: vrăji ofensive care provoacă daune, efecte defensive care oferă protecție sau vindecări care restaurează viața.

În cadrul laboratorului veți construi un sistem simplu de **abilități polimorfe**, unde toate skill-urile sunt tratate uniform printr-o clasă abstractă, iar fiecare derivă comportamente diferite în metoda virtuală `activate()`.

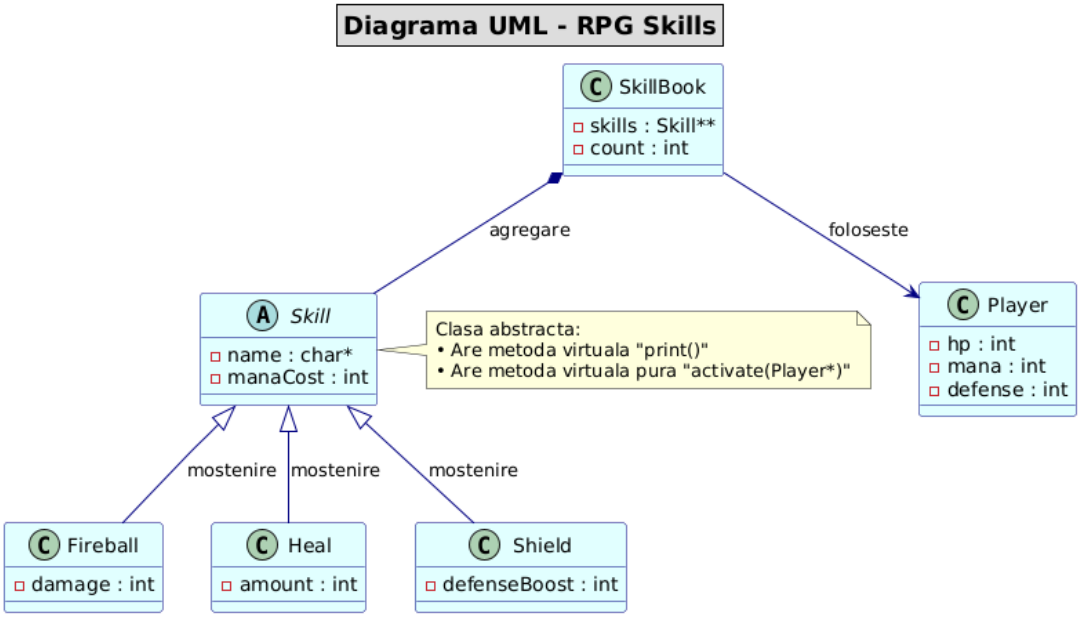
Scopul laboratorului

- înțelegerea folosirii clasei abstracte cu metode virtuale pure;
- aplicarea polimorfismului printr-un vector de pointeri la `Skill`;
- simularea unui sistem RPG folosind relații `is-a` și `has-a`;
- integrarea metodei virtuale `print()` cu `operator<<`;
- consolidarea cunoștințelor despre organizarea în memorie a unui vector de obiecte neomogene.

Structura proiectului

- **Skill** – clasa abstractă (nume, cost mana, metodă virtuală pură `activate()`);
- **Fireball**, **Heal**, **Shield** – abilități concrete cu efecte diferite;
- **Player** – clasa eroului (HP, mana, metode pentru modificarea statusului);
- **SkillBook** – clasa care gestionează un vector de pointeri la skill-uri;
- **main.cpp** – programul care creează skill-uri, le afișează și le activează.

Figura 1 – Diagrama UML a sistemului RPG Skills



Clasa abstractă: Skill

Clasa `Skill` reprezintă superclasa tuturor abilităților din joc. Ea definește elementele comune (nume, cost de mana) și două funcții virtuale:

- `print()` – folosită pentru operatorul `<<`;
- `activate(Player*)` – metodă virtuală pură, care descrie efectul skill-ului.

Această structură vă obligă să implementați comportamente diferite în clasele derivate, demonstând conceptul de polimorfism la runtime.

Clase derivate: Fireball, Heal, Shield

Fireball este o abilitate ofensivă care reduce HP-ul jucătorului vizat (sau poate fi folosită pentru auto-daune în cadrul laboratorului).

Heal reprezintă o vrajă de vindecare care crește HP-ul jucătorului.

Shield reprezintă o protecție temporară care crește caracteristica de apărare sau reduce daunele primite în timpul laboratorului.

Clasa `Player`

Jucătorul este entitatea asupra căreia se aplică skill-urile. Are atribute precum:

- **HP**;
- **mana**;
- eventual **defense** sau **shield active**.

Clasa trebuie să expună metode pentru:

- modificarea HP-ului;
- modificarea manei;
- afișarea statusului prin operatorul `<<`.

Clasa `SkillBook`

`SkillBook` conține un vector de pointeri la `Skill`. Nu știe ce tip concret de skill găzduiește, ceea ce face demonstrația polimorfismului evidentă.
Metoda `activateAll(Player*)` parcurge vectorul și aplică fiecare skill asupra jucătorului.

Fișierul `main.cpp`

```cpp
int main() {
    // TODO: creati un Player cu HP si mana
    // TODO: creati cateva skill-uri: Fireball, Heal, Shield
    // TODO: creati un SkillBook si adaugati skill-urile
    // TODO: afisati skill-urile cu operator<<
    // TODO: apelati activateAll(player)
    // TODO: eliberati memoria
    return 0; }
```

Cerințe (10p)

1. (3p) Implementați clasa abstractă `Skill` (constructori, destructor, metode virtuale).
2. (3p) Implementați clasele derivate `Fireball`, `Heal`, `Shield`.
3. (2p) Implementați clasa `Player` și statusul afișabil.
4. (1p) Implementați clasa `SkillBook` și gestionați corect vectorul de pointeri.
5. (1p) Realizați o sesiune completă de activare în `main.cpp`.

Cerință Bonus (1p) – Sistem de cooldown și prioritizare a skill-urilor

Adăugați în clasa `Skill` următoarele:

```
int cooldown; // numar de ture necesare pana cand skill-ul poate fi refolosit
int currentWait; // cate ture mai sunt pana devine disponibil

virtual int getPriority() const = 0;
```

În clasa `SkillBook`, implementați:

```
void activateTurn(Player* p);
// aplica DOAR skill-urile care au cooldown-ul disponibil
// activeaza mai intai skill-urile cu prioritate mare
```

Reguli bonus

- fiecare skill are un cooldown diferit;
- după activare, cooldown-ul se resetează;
- dacă un skill nu poate fi activat, `currentWait--`;
- ordinea de activare se stabilește prin `getPriority()`:
  - Fireball: prioritate mare (2)
  - Heal: prioritate medie (1)
  - Shield: prioritate mică (0)
- implementarea cere algoritmi simpli de sortare și decizie.

Compilare și rulare

```
g++ -g *.cpp -o lab10
valgrind ./lab10
```

Bonus GitHub Classroom

Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului, respectiv **1p bonus** dacă încarcă mai mult de jumătate dintre ele.

Pașii pentru încărcarea laboratorului:

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte, secțiunea **Resurse** → **Bonus GitHub laborator**.
2. Acolo veți găsi linkul **GitHub Classroom** pentru Laboratorul 10:

   `https://classroom.github.com/a/oCC3Uas3`

3. După acceptare, veți primi propriul repository, ex. `laborator-10-NumePrenume`.
4. Clonați repository-ul local:

```
git clone
↪  git@github.com:Laborator-POO-2025-2026/laborator-10-NumePrenume.git
cd laborator-10-NumePrenume
```

5. După ce ați adăugat fișierele, urmează comenzile:

```
git add .
git commit -m "Mesaj de commit"
git push origin main
```

Atenție: Laboratoarele copiate între studenți sunt considerate nevalide, iar niciunul dintre cei doi participanți nu va primi bonusul. Repository-urile sunt monitorizate automat de sistemul GitHub Classroom.