Ahmed JALAL
Université de Bordeaux
02/12/2025

LAST BATTLE – CALCULATOR V1

Rapport Design Patterns

1. Singleton : ResourceManager :

Problématique :

Dans un jeu, il est essentiel de gérer les ressources (images, mouvement, positions ...) de façon centralisée pour éviter les doublons et faciliter la maintenance.

Solution :

J'ai créé la class < **ResourceManager** > en singleton, ce qui garantit qu'une seule instance existe et est accessible partout dans le projet. Cela simplifie le chargement et la gestion des ressources.

Exemple d'utilisation :

```
ResourceManager.GetInstance().loadResource(''sound'') ;
```

2. Strategy : AttackStrategy

Problématique :

Les personnages du jeu doivent pouvoir changer dynamiquement leur comportement d'attaque (corps à corps, distance , etc....)

Solution :

J'ai défini l'interface <**AttackStrategy**> et ses implémentation (MeleeAttack, RangeAttack).

Chaque personnage *exemple : spider ,* possède un champ <AttackStrategy> et une méthode Attack() qui délègue à la stratégie choisie.

Exemple d'utilisation :

```
Spider.setAttackStrategy(new MeleeAttack()) ;

Spider.attack() ;
```

3. Factory : EnemyFactory

Problématique :

La création des ennemis doit être centralisée et flexible pour faciliter l'ajout de nouveaux types.

Solution :

La class <**EnemyFactory**> propose une méthode statistique qui instancie les différents types d'ennemis selon le besoin.

Exemple d'utilisation :

Java

```
ICharacter enemy = EnemyFactory.createEnemy(''Spider'') ;
```

Ces patterns ont été choisis pour leur pertinence dans un jeu orienté objet :

Singleton : Gestion unique et centralisée des ressources

Strategy : Flexibilité et évolutivité des comportements.

Factory : Simplicité et extensibilité de la création d'objets[i]

Ils facilitent l'évolution du projet, l'ajout de nouveaux personnages, comportements ou ressources, tout en respectant les consignes du sujet