Thomas Barseghian
Hugo Leger
TP5

SAE31

Bataille navale : nos choix de conception

Introduction

Pour notre implémentation de la bataille navale en Java, nous avons cherché à structurer le code de façon modulaire et maintenable, en privilégiant les hiérarchies et interfaces pour permettre l'ajout de nouvelles fonctionnalités sans modifier l'existant. L'architecture adoptée suit le modèle MVC (Model, View, Controller) : chaque package correspond à une composante, facilitant la séparation des responsabilités.

Model

Le modèle contient la logique métier du jeu, avec trois parties principales : la grille, les objets plaçables (bateaux, armes, pièges) et les joueurs.

Grid

La classe Grid est composée de multiples Tile, chacune pouvant contenir un objet placable et possédant un type (WATER ou LAND). Cette organisation en petites unités (les `Tile`) permet de gérer de manière précise ce qui se passe sur chaque case, par exemple les tirs ou les effets des pièges. La grille stocke également des compteurs de statistiques (tirs réussis, ratés, recherches sur les îles).

PlacableObject

Les objets plaçables implémentent l'interface PlacableObject. Les bateaux sont représentés par une hiérarchie abstraite (Ship) avec des classes concrètes (AircraftCarrier, Cruiser, etc.), chaque bateau connaissant sa taille, ses points de vie et sa direction. Les armes (Weapon) et pièges (Trap) ont également des classes abstraites et concrètes, et utilisent des factories (ShipFactory, WeaponFactory, TrapFactory) pour faciliter leur création et l'ajout de nouveaux types concrets.

Les pièges comme le Tornado ont un effet dans le temps sur le joueur adverse. Pour gérer cela proprement, nous avons défini une interface Effects et une classe TornadoEffect qui stocke l'état du piège sans surcharger le joueur avec des attributs spécifiques. Cette approche permet d'ajouter facilement d'autres pièges avec des effets temporels.

Player

La classe Player (abstraite) stocke ses bateaux, armes et pièges, ainsi que sa grille. Les classes Human et AI héritent de Player et ajoutent des méthodes pour placer les objets et attaquer. Les actions des joueurs sont encapsulées pour respecter le principe d'encapsulation.

AI

La classe AI contient la liste m_tilesHit, qui enregistre les coups déjà joués par l'IA. Cette liste est utilisée par la méthode AttackWithTought pour choisir des coups de manière stratégique et réfléchie.

Game

La classe Game agit comme une façade, centralisant le contrôle du jeu : état courant, tour en cours, vérification de victoire et progression des tours. Cela permet à la vue et au contrôleur d'interagir avec une seule classe, simplifiant la communication et limitant les dépendances directes.

View

Les classes du package view représentent les différentes interfaces utilisateur (terminal, configuration, placement, écran principal et fin de partie). Elles affichent l'état du jeu et se mettent à jour en observant les classes utiles comme Grid, Player et Game selon le pattern Observer.

Controller

Les classes du package controller centralisent la logique applicative. Chaque contrôleur (jeu, placement, configuration, écran) manipule le modèle en réponse aux actions de l'utilisateur et coordonne les vues correspondantes, suivant le pattern MVC.

Patterns de conception

- Factory : pour créer les bateaux, armes et pièges de manière centralisée, facilitant l'ajout de nouveaux types.

- Observer : utilisé pour notifier les vues lorsqu'il y a des changements, comme le tour en cours, l'état du jeu. Les classes observés sont Game, Player et Grid et servent à changer l'affichage en fonction des changements des attributs.

- Strategy : utilisée pour les armes et les méthodes d'attaque. Elle permet à la classe Player de changer dynamiquement son comportement d'attaque en assignant une stratégie différente, sans modifier le code de la classe.

- Façade : la classe Game centralise les interactions, simplifiant la communication entre vue et modèle.

Choix supplémentaires

- Stockage des objets : les objets sont à la fois stockés dans les Tile et dans des listes au niveau des joueurs, permettant de gérer facilement les effets sur les objets (attaques, pièges) tout en ayant une vision globale pour le joueur.

- Effets temporels : encapsulés dans des classes dérivées d'Effects, pour ne pas alourdir Player avec des attributs spécifiques à chaque type de piège.

Conclusion

Notre conception vise à séparer clairement les responsabilités, à faciliter l'évolution du code et à rendre le jeu extensible et maintenable. Grâce aux patterns et aux interfaces, il est possible d'ajouter de nouveaux types de bateaux, armes ou pièges, ou de modifier les stratégies de tir, sans avoir à toucher au cœur de la logique du jeu.