Chapitre 8

Exemple final

Ce exemple correspond à la modélisation d'une bataille navale. Les principes sont les suivants :

- le jeu est composé de 3 équipes et d'un plateau de jeu (qui comprendra en particulier un "dessin" pour visualiser l'état du jeu et le déplacement des navires) ;
- chaque équipe est composée d'une liste de navires et peut être soit un bataillon (statut militaire), soit des pêcheurs (statut neutre) ; on considérera qu'il y a deux équipes militaires et une neutre ;
- les navires sont donc soit des navires de guerre, soit des bateaux de pêche ; il y a 2 types de navire de guerre (les destroyers et les sous-marins) et 1 seul type de bateau de pêche (les chalutiers) ; les destroyers et les chalutiers sont des navires de surface alors que les sous-marins sont des navires de profondeur ;
- une des trois équipes sera jouée par l'ordinateur, alors que les deux autres seront jouées par des humains ; l'attribution des rôles sera faite par tirage au sort lors du lancement de la partie ;
- en cours de partie, chaque équipe devra choisir à tour de rôle un navire dans sa liste et l'action à réaliser (déplacement, tir, pêche) ; les choix des équipes humaines seront faits par interaction avec les joueurs (entrée au clavier) alors que ceux de l'équipe artificielle seront faits par tirage aléatoire ;
- chaque navire est susceptible d'être coulé par un tir provenant d'un navire de guerre ; et les sous-marins sont susceptibles d'être endommagés par les chalutiers (hélices prises dans le chalut) ; dans ce cas, ils ne sont pas coulés mais ils ne peuvent plus se déplacer ;
- les capacités de déplacement et de tir dépendent de chaque navire ; dans chaque case du dessin, il ne peut y avoir au plus qu'un navire de surface et un navire de profondeur ; par souci de simplification, on considérera qu'un navire n'occupe qu'une seule case et donc qu'un seul tir au but suffit à le couler ; de même, pour qu'un sous-marin soit endommagé, il suffit qu'il se trouve sous un chalutier lorsque celui-ci pêche ;
- la fin de la partie se produit dès que l'une des 3 équipes n'a plus de navire ; le gagnant est l'équipe à qui il reste le plus de navires valides (donc non coulés et non endommagés) ; cela peut être l'équipe des pêcheurs.

Du point de vue de l'architecture Java, on trouve les éléments suivants :

- `Jeu` : gestion du Plateau et des 3 Equipes ; répartition initiale du type des joueurs ; enchaînement des coups, récupération des choix de chaque équipe et répercution sur le Plateau pour la visualisation et sur les équipes pour la suite du jeu ; détection et arrêt de la partie ;
- `Plateau` : gestion d'une matrice de CasePlateaux et visualisation de l'impact de chaque coup ;
- `CasePlateau` : détection et gestion de l'occupation (vide ou pas, occupée par quoi) ;
- `Equipe` : gestion d'un ensemble de navires, définition d'une commande suivant le type de l'équipe (si humain alors interface de saisie des commandes, si IA alors tirage au sort des commandes) ;
- `EqBataillon` : Equipe avec capacité de tir (impact sur le type de commande disponible) ;
- `EqPecheur` : Equipe avec capacité de pêche (impact sur le type de commande disponible) ;
- `Commande` : l'ensemble des informations nécessaires à une étape de jeu : l'équipe qui agit, le navire concerné, l'action à faire (déplacement, Tir, Pêche), la direction dans laquelle faire cette action (en cas de déplacement ou de tir) ;
- `Navire` : capacité de déplacement et accès à l'état courant ;
- `NavireSurface` : Navire avec niveau d'occupation en surface ;
- `NavireProfondeur` : Navire avec niveau d'occupation en profondeur ;

— `Destroyer` : NavireSurface avec affinement des capacités de déplacement et de tir (portée), état courant étant soit valide, soit coulé ;
— `SousMarin` : NavireProfondeur avec affinement des capacités de déplacement et de tir (portée), état courant étant soit valide, soit coulé, soit endommagé ;
— `Chalutier` : NavireSurface avec affinement de la capacité de déplacement, état courant étant soit valide, soit coulé.

La répartition entre interfaces, classes et enum pourrait être la suivante :

interfaces : `Joueur`, `JHumain` (hérite de `Joueur`), `JIA` (hérite de `Joueur`) ;

classes : `Jeu`, `Plateau`, `CasePlateau`, `Equipe` (implémente `JHumain` et `JIA`), `EqBataillon` (hérite de `Equipe`), `EqPecheur` (hérite de `Equipe`), `Commande`, `Navire`, `NavireSurface` (hérite de `Navire`), `NavireProfondeur` (hérite de `Navire`), `Destroyer` (hérite de `NavireSurface`), `SousMarin` (hérite de `NavireProfondeur`), `Chalutier` (hérite de `NavireSurface`) ;

enum : `Statut` (militaire ou neutre), `Nature` (humain ou IA), `TypeNav` (chalutier, destroyer ou sous-marin), `Action` (déplacement ou tir ou pêche), `Direction` (les 4 points cardinaux)

A noter aussi au moins deux types d'exception :
— `OccupException` : pour traiter le cas où un déplacement ne serait pas possible car il y a déjà trop de navires dans la case ciblée,
— `LimiteException` : pour traiter le cas où un déplacement ou un tir ne seraient pas possibles car la position visée serait hors du plateau.

Exercice 1 : Proposer un diagramme de classe UML correspondant à ce jeu.

Exercice 2 : Ecrire en Java
— une des exceptions : `OccupException`,
— une des énumérations : `Direction`,
— deux des interfaces : `Joueur` et `JHumain`,
— deux classes : `Equipe` et `EqBataillon`.