Programmation Java - Master TTT

13 janvier 2010 - 2 heures

Les documents hors livres sont autorisés.

On prendra soin de bien préciser les visibilité des membres des classes.

On étudie dans ce sujet un morceau de programme permettant de jouer à la bataille navale. Un plateau de bataille navale est une grille sur laquelle sont disposés des bateaux horizontalement et verticalement. Chaque case de la grille est représentée par une paire d'entiers.

Sur un plateau, deux joueurs s'affrontent, l'attaquant et le défenseur. A chaque tour, l'attaquant (qui ne voit pas les bateaux) lance une bombe sur une case et touche la portion du bateau qui occupe éventuellement cette case, puis le défenseur peut agir sur un de ses bateaux. Si toutes les portions d'un bateau sont touchées, il est coulé.

	0	1	2	3	4	5
0		■				
1						
2		■				
3		■				
4					■	
5		■				

Exercice n° 1

Écrire une interface `Bateau` qui déclare les méthodes suivantes :

- une méthode `coup` qui prend en argument deux entiers x et y et retourne vrai si le bateau est visible (voir exercice 2) et occupe la case (x, y) ; la portion de ce bateau sera alors touchée.
- une méthode `estPresent` qui prend en argument deux entiers x et y retourne vrai si le bateau occupe la case (x, y) ; contrairement à `coup`, cette méthode ne permet pas de toucher le bateau.
- une méthode `estCoule` sans argument qui retourne vrai si le bateau est coulé.
- des méthodes `getX`, `getY` qui permettent de connaître les coordonnées de la case de référence d'un bateau.

Exercice n° 2

Écrire complètement une classe `SousMarin` qui implémente `Bateau` de sorte que :

- un sous-marin occupe une seule case sur le plateau ;
- le constructeur de `SousMarin` prend deux arguments : x et y , où (x, y) est la case où est placé le sous-marin ;
- `SousMarin` contient une méthode `plonge` qui lui permet de plonger, une méthode `surface` qui lui permet de refaire surface et une méthode `estVisible` qui retourne vrai si et seulement si le sous-marin est en surface (ce qui est le cas à sa création) ;
- `SousMarin` contient une méthode `deplace` qui prend en argument deux entiers dx et dy compris entre -1 et 1 et qui permet de déplacer le sous-marin de la case (x, y) à la case $(x + dx, y + dy)$; un sous-marin en plongée ne peut pas se déplacer ; la méthode retourne vrai si le déplacement est possible, faux si le sous-marin est en plongée ou si les arguments ne sont pas corrects ;
- un sous-marin en plongée n'est pas visible et ne peut donc pas être touché.

Exercice n° 3

Écrire complètement une classe `Croiseur` qui implémente `Bateau` de sorte que :

- un croiseur occupe trois cases alignées sur le plateau ;
- le constructeur de `Croiseur` prend trois arguments : x , y et h , où (x, y) est la case de référence du croiseur, c'est-à-dire celle du milieu, et h est vrai si et seulement si le croiseur est horizontal (sinon, il est vertical) ;
- une méthode `estHorizontal` qui retourne vrai si le croiseur est horizontal et faux sinon.
- `Croiseur` contient une méthode `deplace` qui prend en argument un entier d égal à 1 ou -1 et déplace le croiseur de (x, y) vers $(x + d, y)$ s'il est horizontal, vers $(x, y + d)$ s'il est vertical ;
- un croiseur est toujours visible.

Exercice n° 4

Écrire une classe `Plateau` contenant les éléments suivant :

- deux entiers m et n qui stockent la taille du plateau (m colonnes numérotées de 0 à $m - 1$ et n lignes numérotées de 0 à $n - 1$);
- une `LinkedList` de `Bateau` présents sur le plateau.
- un constructeur avec deux arguments entiers m et n indiquant la taille du plateau;
- une méthode `quelBateau` avec deux arguments entiers x et y et qui retourne le `Bateau` qui occupe la case (x, y) ; si celle-ci n'est occupée par aucun bateau, la méthode retourne `null`;
- une ou des méthodes `ajouteBateau` qui prennent en argument un bateau, vérifient que les cases qu'il occuperait sont dans les limites du plateau et sont libres, et l'ajoute à la liste des bateaux présents. La méthode retourne vrai si l'ajout est réussi, faux sinon.
- une méthode `coup` qui prend deux arguments entiers x et y ; si la case (x, y) est vide ou occupée par un sous-marin en plongée, `coup` retourne 0; si elle est occupée par un bateau, celui-ci est touché et la méthode retourne 1 s'il n'est pas coulé, 2 si c'est un croiseur coulé, et 3 si c'est un sous-marin; tout bateau coulé est enlevé de la liste des bateaux présents. S'il ne reste alors qu'un bateau présent et que c'est un sous-marin, il fait surface.
- une méthode `deplace` qui prend en argument un `Croiseur` et un entier d et déplace ce croiseur s'il appartient à la liste des bateaux présents, et s'il n'est pas bloqué par un autre bateau ou le bord du plateau.
- une méthode `deplace` qui prend en argument un `SousMarin` et deux entiers dx et dy et déplace ce sous-marin s'il appartient à la liste des bateaux présents, s'il n'est pas le seul bateau présent et s'il n'est pas bloqué par un autre bateau ou le bord du plateau.
- des méthode `plonge` et `surface` qui prennent en argument un `SousMarin` et font plonger (ou apparaître) ce sous-marin s'il appartient à la liste des bateaux présents; s'il est le seul bateau présent, il ne peut pas plonger.
- une méthode `estVide` qui retourne vrai s'il n'y a pas de bateau sur le plateau.

Exercice n° 5

A l'aide des classes précédentes, écrire un programme qui permet de jouer à la bataille navale de la manière suivante. On joue sur un plateau 10×10 .

Le programme demande au joueur 1 (le défenseur) de placer trois croiseurs numérotés 1, 2 et 3; pour chaque croiseur, le joueur doit entrer au clavier deux entiers représentant la case de référence du bateau et une lettre, h ou v indiquant si le croiseur est horizontal ou vertical; ensuite, il doit placer un sous-marin en entrant deux entiers indiquant sa case. Pour chaque bateau, si le placement échoue, le programme redemande au joueur toutes les informations sur ce bateau.

Pour chaque tour de jeu, le joueur 2 entre deux entiers x et y indiquant où il attaque; le programme répond par "Coup dans l'eau", "Croiseur touché", "Croiseur coulé", "Sous-marin touché" selon le résultat du coup. Si le plateau est vide, le programme annonce la fin du jeu et le nombre de tours écoulés (le but de l'attaquant est de gagner au plus vite, celui du défenseur de résister le plus longtemps possible).

Le tour se termine ensuite par une action du joueur 1 parmi les cinq suivantes :

- il tape 0 pour indiquer qu'il ne fait rien;
- il tape un numéro i entre 1 et 3 suivi de 1 ou -1 pour déplacer le croiseur i ;
- il tape 4 suivi de p pour faire plonger son sous-marin;
- il tape 4 suivi de s pour que son sous-marin fasse surface;
- il tape 4 suivi de m et de deux entiers entre -1 et 1 pour déplacer son sous-marin.

On commence ensuite un autre tour de jeu.

Exercice n° 6

On désire ajouter deux actions possibles à la fin du tour :

- 5 suivi d'un nom n permet de sauvegarder la partie en cours dans un fichier de nom n ;
- 6 suivi d'un nom n permet de charger la partie stockée dans le fichier de nom n .

Décrire ce qu'il faut ajouter au programme et aux classes pour que ceci soit possible.