

TD2 : conception de jeu de tests - Revision

Révision à compléter

De manière générale, les actions que **doit réaliser** une méthode sont de 3 types :

Les actions **réalisées par** une méthode dépendent de 3 choses :

Comment procéder ?

Illustration de la procédure :

```
public class IntIterator {
    private int max = -1;
    private int current = -1;
    public IntIterator(int nbValeurs) {
        if (nbValeurs > 0) max = nbValeurs-1; }
    public boolean hasNext() { return (current < max); }
    public int next() { if (current >= max) return max;
        current++; return current; }
    public int getCurrent() {return current; }
    public int getMax() {return max; }
}
```

Question

1. Listez les cas d'usage à tester pour la méthode `hasNext`, et pour chaque cas d'usage, listez les valeurs à tester.
2. Proposez des scénarii de tests permettant de tester tous les cas d'usage et réalisez la méthode `hasNextTest` qui permet de tester la méthode `next`.
3. Listez les cas d'usage à tester pour la méthode `next`, et pour chaque cas d'usage, listez les valeurs à tester.
4. Proposez des scénarii de tests permettant de tester tous les cas d'usage et réalisez la méthode `nextTest` qui permet de tester la méthode `next`.

Etude de cas « Bataille Navale »

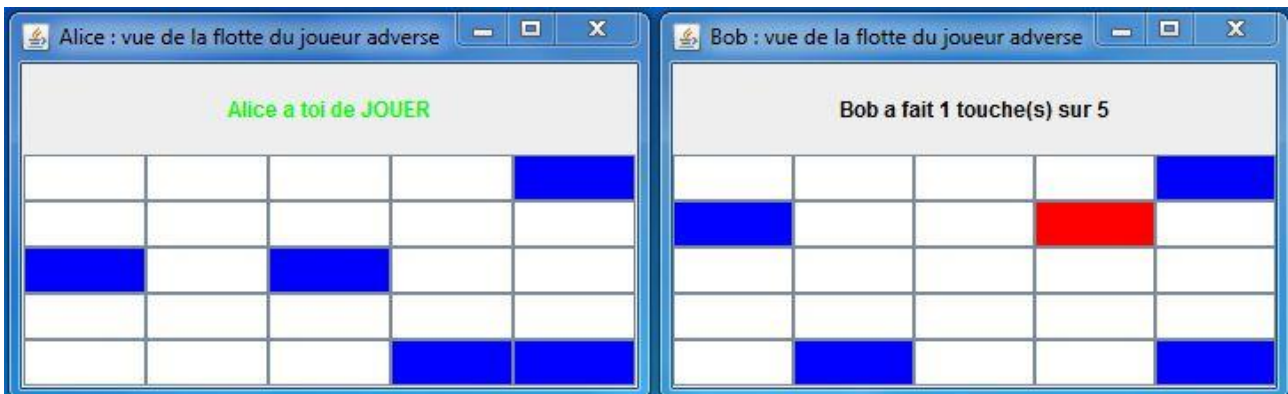
Comme son nom l'indique, la bataille navale est un jeu consistant à torpiller les navires de son adversaire. La flotte de chaque joueur est placée aléatoirement au début du jeu par l'application.

Ensuite, tour à tour, les joueurs lancent virtuellement des torpilles sur l'ennemi en indiquant les coordonnées d'un tir (par exemple B4).

L'application indique si l'un de ses bateaux a été atteint ou pas. Le gagnant est celui qui parvient à torpiller complètement les navires de l'adversaire avant que tous les siens ne le soient.

La taille des grilles de la bataille navale dépend du niveau (elle est plus importante dans les niveaux supérieurs).

Chaque joueur « voit » à une grille correspondant à la connaissance qu'il a acquise de la flotte de son adversaire. Par exemple, une case « vide » est une case non torpillée, une case bleue indique une torpille qui n'a pas touché un navire ; une case rouge indique une touche.



Quelques informations sur l'implémentation du jeu

```
public enum EtatCaseBN { BATEAU_TOUCHE, A_L_EAU, NON_DECOUVERTE ; }
```

L'attribut `torpillée` de `CaseBN` est à `mis à vrai`, une fois que la case est torpillée. A la construction de l'objet, l'attribut `aUnBateau` est mis à `vrai` si la case contient un bateau.

CaseBN
-x : int - y : int - aUnBateau : boolean - torpillée : boolean = false
+ getEtat() : EtatCaseBN + torpiller() : EtatCaseBN + CaseBN (x : int, y : int, aUnBateau : boolean) : void + getX() : int + getY() : int

La méthode `getEtat` retourne `NON_DECOUVERTE` tant que la case n'a pas été torpillée. Après que la case ait été torpillée, `getEtat` retourne `BATEAU_TOUCHE` si l'attribut `aUnBateau` est à `vrai` ; sinon `getEtat` retourne `A_L_EAU`.

La méthode `torpiller` de `CaseBN` met l'attribut `torpillé` à `vrai`. Puis, elle retourne `BATEAU_TOUCHE` si l'attribut `aUnBateau` est à `vrai` ; sinon elle retourne `A_L_EAU`.

Questions

5. Listez les cas d'usage de la méthode `getEtat` de la classe `CaseBN` et pour chaque cas d'usage, listez les valeurs à tester.
6. Listez les cas d'usage de la méthode `torpiller` de la classe `CaseBN` et pour chaque cas d'usage, listez les valeurs à tester.
7. Proposez les scénarios de tests permettant de tester tous les cas d'usage et réalisez la méthode `torpillerCaseTest` qui permet de tester la méthode `torpiller` de la classe `CaseBN`.
8. Proposez les scénarios de tester permettant de tester tous les cas d'usage et réalisez la méthode `getEtatTest` qui permet de tester la méthode `getEtat`.

Vous pouvez répondre aux deux questions précédentes simultanément.

Si le joueur a déjà perdu (à savoir que `nombreTouchesRecues` n'est pas inférieure au nombre de touches à faire) la méthode `torpiller()` de la classe `JoueurBN` doit retourner immédiatement l'état `TERMINEE` sans exécuter aucune action.

Sinon, la méthode `torpiller()` doit incrémenter la variable `nombreTirsRecus`. Si la case n'a pas été déjà été torpillée alors elle doit être torpillée et la variable `nombreTouchesRecues` doit être incrémentée dans le cas où la case torpillée contient une partie d'un bateau.

La méthode `torpiller()` doit retourner l'état `EN_COURS` si le joueur torpillé n'a pas encore perdu sinon elle doit retourner l'état `TERMINEE`.

Questions

9. Compléter avec des associations le diagramme de classes ci-dessus, si nécessaire.
10. Listez les cas d'usage de la méthode `torpiller` de la classe `JoueurBN` et pour chaque cas d'usage, listez les valeurs à tester.
11. Proposez un jeu de tests complet pour l'implémentation de la méthode `torpiller` de la classe `JoueurBN`.

La méthode `tirParJoueur(x, y, jx)` de la classe `ControleurBN` retourne `false` (elle n'appelle aucune méthode) si la partie est terminée, si `jx` n'est pas le joueur courant ou si `(x, y)` ne correspond pas au coordonnées d'une case.

Dans les autres cas, la méthode `torpiller(x, y)` du joueur non-courant (`jy`) est appelée. Si cette torpille lancée par `jx` résultant en la perte de la partie par `jy` alors `jx` est déclaré « gagnant » ; sinon le joueur courant change (`jy` devient le joueur courant).

Questions

12. Compléter le diagramme de classes ci-dessus avec des associations, si nécessaire.
13. Proposez un jeu de tests complet pour l'implémentation de la méthode `tirParJoueur` de la classe `ControleurBN`.

Diagramme de classes à compléter

```
public enum EtatJeu { EN_COURS, TERMINE; }
```

CaseBN

