Séance 9: Bataille navale

Université Paris-Diderot

Objectifs:

- Manipulation de tableaux de tableaux
- Faire un programme entier

Le but de ce TP est de programmer un jeu de bataille navale auquel vous pourrez jouer contre l'ordinateur. Ce jeu se joue à deux joueurs, chacun dispose d'une grille de 10 cases sur 10 cases, les colonnes de cette grille sont indiquées par une lettre de A à J et les lignes sont numérotées de 1 à 10 . Sur cette grille sont placés 5 bateaux en horizontal ou en vertical :

1. Un **porte-avions** (5 cases)
2. Un **croiseur** (4 cases)
3. Un **contre-torpilleurs** (3 cases)
4. Un **sous-marin** (3 cases)
5. Un **torpilleur** (2 cases)

Le but de chaque joueur est de couler tous les bateaux de l'autre joueur. Chaque joueur joue tour à tour en proposant une position où lancer une torpille pour toucher un bateau adverse en indiquant une position sur la grille (par exemple B3) et l'adversaire répond *Touché* si la torpille touche un bateau, *Coulé* si l'adversaire touche un bateau et le coule (c'est-à-dire qu'il a touché toutes les cases du bateau correspondant) ou *À l'eau* si la torpille n'a rien touché ou a touché un emplacement d'un bateau déjà coulé.

Pour encoder une grille de bataille navale, nous utiliserons un tableau contenant 10 tableaux de 10 entiers avec le codage suivant :

- 0 indique qu'il n'y a pas de bateau
- 1 indique la présence d'un porte-avions
- 2 indique la présence d'un croiseur
- 3 indique la présence d'un contre-torpilleurs
- 4 indique la présence d'un sous-marin
- 5 indique la présence d'un torpilleur
- 6 indique la présence d'un endroit de bateau touché

Par exemple la grille ci-contre sera encodée par le tableau de tableaux où le premièr tableau vaudra `{ 5, 6, 0, 4, 0, 0, 2, 2, 2, 2}`, le deuxième tableau vaudra `{0, 0, 0, 4, 0, 0, 0, 0, 0, 0}`, etc. Si `t` est la variable contenant ce tableau, alors `t[0][1]` vaut 6 car à la position B1 un torpilleur est touché.

	A	B	C	D	E	F	G	H	I	J
1	5	6	0	4	0	0	2	2	2	2
2	0	0	0	4	0	0	0	0	0	0
3	0	0	0	4	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	3	3	3	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	1	1	1	1

Pour ce TP, toutes vos fonctions et votre programme seront encodés dans un fichier `bataille.java`. Dans ce fichier commencez par créer deux variables (qui seront utilisées comme variables globales par vos fonctions) `gridComp` et `gridPlay`.

```java
import java.util.Random;
public class bataille {
  public static int [][]gridComp = new int [10][10];
  public static int [][]gridPlay = new int [10][10];
  ...
}
```

La première variable contiendra la grille de l'ordinateur et la deuxième la grille du joueur.

Exercice 1 (Remplissage de la grille de l'ordinateur, ⋆-⋆⋆)
La première étape consiste à remplir la grille de l'ordinateur. Pour ce faire, on a besoin de différentes fonctions.

```java
public static boolean posOk(int [][]grille, int l, int c, int d, int t)
   { ... }
public static void initGridComp() {...}
```

1. *La première fonction* `posOk` *prend en paramètres une grille, un numéro de ligne et un numéro de colonne (compris entre 0 et 9), un entier* `d` *codant une direction (1 pour horizontal et 2 pour vertical) et un entier* `t` *donnant le nombre de cases d'un bateau, et renvoie* `true` *si on peut mettre le bateau sur les cases correspondantes (sachant que la position donnée par* `l` *et* `c` *correspond à la case la plus à droite dans le cas horizontal et à la case en haut dans le cas vertical). Si on ne peut pas mettre le bateau, la fonction renvoie* `false`*. Il y a deux choses à vérifier : d'une part, les cases où l'on souhaite mettre le bateau doivent valoir* `0`*, d'autre part le bateau ne doit pas sortir de la grille.*
2. *Écrire une procédure* `initGridComp` *qui va mettre au hasard les 5 bateaux sur la grille* `gridComp`*. Pour faire cela, pour chacun des bateaux vous devez tirer au hasard trois nombres : un numéro de ligne entre 0 et 9, un numéro de colonne entre 0 et 9, et un numéro de direction entre 1 et 2 (1 pour horizontal et 2 pour vertical). Il vous faut répéter le tirage au hasard si le tirage ne donne pas une position de bateau correcte, éventuellement plusieurs fois de suite, jusqu'à ce que le tirage soit correct (on utilisera une boucle* `while` *et la fonction* `posOk`*). Dès qu'on a tiré un ensemble de valeurs correctes, on met le bateau sur la grille* `gridComp`*. Notez que la variable* `gridComp` *est une variable globale, on n'a pas besoin de la donner comme paramètre de la fonction, ni de la renvoyer.*
   Indication : *Pour tirer des entiers au hasard entre* `a` *inclus et* `b` *exclu, utilisez la fonction suivante.*

```java
public static Random rand = new Random();
public static int randRange(int a, int b){
  return rand.nextInt(b-a)+a;
}
```

3. *Pour tester votre fonction, écrivez une procédure* `printGrid` *qui prend en arguments une grille et l'affiche de façon lisible. Par exemple, pour la grille donnée en exemple précédemment, votre procédure devra afficher :*

```
    A  B  C  D  E  F  G  H  I  J
1   5  6  0  4  0  0  2  2  2  2
2   0  0  0  4  0  0  0  0  0  0
3   0  0  0  4  0  0  0  0  0  0
4   0  0  0  0  0  0  0  0  0  0
5   0  0  0  0  0  0  0  0  0  0
6   0  0  0  0  0  0  0  0  0  0
7   0  3  3  3  0  0  0  0  0  0
8   0  0  0  0  0  0  0  0  0  0
9   0  0  0  0  0  0  0  0  0  0
10  0  0  0  0  0  1  1  1  1  1
```

Utilisez les espaces pour avoir un bel alignement à l'affichage.

□

Exercice 2 (Remplissage de la grille du joueur, ⋆⋆)
Il faut ensuite demander au joueur de remplir sa grille. Pour cela, écrivez une procédure `initGridPlay` *qui demande à l'utilisateur de saisir une à une les positions de ses bateaux. Pour chaque bateau, on demande à l'utilisateur de saisir d'abord la lettre, puis le nombre, puis la direction (1 pour horizontal, 2 pour vertical), en répétant la question si l'utilisateur saisit une réponse incompréhensible (par exemple une lettre qui n'est pas comprise entre A et J, ou un mauvais nombre pour la ligne ou la direction). Ensuite, grâce à la fonction* `posOk` *de l'exercice précédent, on va tester si le bateau peut être placé dans* `gridPlay`*, et si ce n'est pas possible, on va de nouveau poser les trois questions jusqu'à obtenir un placement correct pour le bateau.* **Attention :** *en appelant* `posOk` *il faut convertir le nombre et la lettre en chiffres entre 0 et 9.*

Une exécution de cette fonction aura la forme suivante :

```
Donnez la lettre pour le porte avions :
B
Donnez le nombre pour le porte avions :
8
Voulez-vous qu'il soit horizontal (1) ou vertical (2) ?
2
Erreur : Le porte-avions ne rentre pas dans la grille.
Donnez la lettre pour le porte avions :
....
```

Indication : *Pour récupérer les données tapées par l'utilisateur au clavier, utilisez les fonctions ci-dessous.*

```java
public static String readString(){
  return System.console().readLine();
}
public static boolean isInt(String s){
  return s.matches("\\d+");
}
public static int readInt(){
  while(true) {
    String s = readString();
    if (isInt(s)) return Integer.parseInt(s);
  }
}
```

□

Exercice 3 (Premiers tirs de torpilles, ⋆⋆)

1. *Écrivez une fonction* `hasDrowned` *qui prend en paramètre une grille* `grid` *et le numéro d'un bateau compris entre 1 et 5, et qui renvoie* `true` *si le bateau correspondant a été coulé (c'est-à-dire s'il n'est plus présent dans la grille), et* `false` *sinon.*
2. *Nous allons maintenant écrire une procédure* `oneMove` *qui prend en paramètres une grille* `grid`*, deux entiers compris entre 0 et 9 (le premier pour un numéro de ligne et le deuxième pour un numéro de colonne) et qui affiche soit "Touché", soit "Coulé" (en indiquant de quel bateau il s'agit), soit "À l'eau". Cette procédure mettra à jour la grille le cas échéant en inscrivant un 6 lorsqu'on touche un bateau. Pour savoir si on a seulement touché un bateau ou si on l'a aussi coulé, on récupère le numéro du bateau, on met la grille à jour et ensuite on utilise la fonction* `hasDrowned`*.*
3. *Écrivez une fonction* `playComp` *qui renvoie un tableau contenant deux entiers tirés au hasard entre 0 et 9. Cette fonction sera utilisée pour faire jouer l'ordinateur.*
4. *Donnez le code d'une fonction* `isOver` *qui prend en paramètre une grille et renvoie* `true` *si tous les bateaux de cette grille ont été coulés.*

□

Exercice 4 (Bataille !, ⋆⋆⋆)

Nous disposons maintenant de tous les ingrédients pour programmer la bataille navale. Nous allons programmer le jeu dans une procédure `play` *qui ne prend aucun paramètre et qui fonctionnera de la façon suivante :*

- *tout d'abord la procédure remplira les grilles* `gridComp` *et* `gridPlay` *;*
- *ensuite tant que les tableaux des deux grilles ont encore des bateaux non coulés, elle fera jouer à tour de rôle l'ordinateur puis le joueur. Pour ce dernier, elle lui demandera de rentrer une lettre puis un chiffre correspondant au coup qu'il souhaite jouer. Si à un moment, l'ordinateur gagne alors la procédure ne devra pas faire jouer le joueur mais sortir en précisant qui a gagné. Nous vous rappelons qu'un coup de l'ordinateur modifie la grille du joueur* `gridPlay` *et qu'un coup du joueur modifie la grille de l'ordinateur* `gridComp`*. Votre procédure utilisera les fonctions précédentes et dans un premier temps, vous devrez faire le plus d'affichage possible (évolution des grilles, coups joués, etc) afin de vérifier son bon comportement.*

□