Université de Paris **Programmation Orientée Objet**
L2 **Année 2020-2021**

TP - Séance n°2

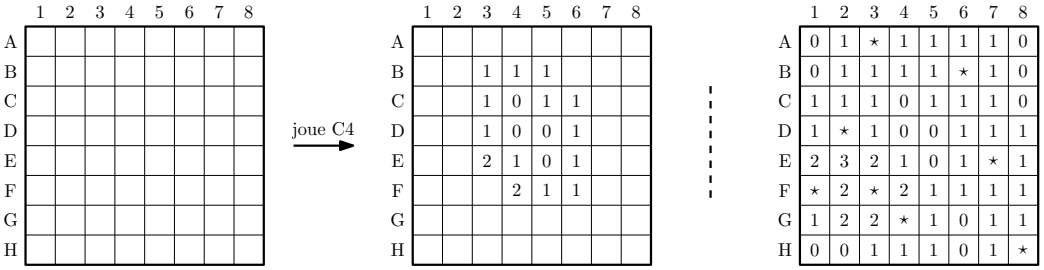
Révisions : jeu du démineur

Le but de ce TP est d'implémenter un jeu de **démineur**. La section 1 explique les règles du jeu et la section 2 donne les principaux ingrédients de l'implémentation. Nous vous donnons ensuite un déroulé pas à pas dans la section 3. Si vous vous sentez à l'aise, vous pouvez décider de ne pas suivre les instructions pas à pas, et de modéliser différemment votre jeu. Dans ce cas, prenez le temps de bien planifier vos classes avant de vous mettre à coder.

1 Jeu du démineur

Le démineur est un jeu où le joueur doit trouver les mines d'un terrain miné sans les déclencher. Dans cette section, nous rappellons les règles du jeu. Vous trouverez plusieurs implémentations en ligne, par exemple : `https://www.minesweeper-online.org/` (pour voir un plateau 8×8, cliquez sur Game → Beginner.)

Au départ, le terrain est complètement invisible. À chaque tour, le joueur peut décider de découvrir une case. Si cette case est dépourvue de mine la case est révélée, s'il y avait une mine le jeu s'arrête et on a perdu.



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

	1	2	3	4	5	6	7	8
A	0	1	⋆	1	1	1	1	0
B	0	1	1	1	1	⋆	1	0
C	1	1	1	0	1	1	1	0
D	1	⋆	1	0	0	1	1	1
E	2	3	2	1	0	1	⋆	1
F	⋆	2	⋆	2	1	1	1	1
G	1	2	2	⋆	1	0	1	1
H	0	0	1	1	1	0	1	⋆

FIGURE 1 – À gauche, le joueur joue son premier coup en C4 et découvre toute une zone. À droite, le terrain complètement découvert. Les mines sont indiquées par des étoiles.

Chaque case révélée indique le nombre de mines dans les 8 cases adjacentes. Si une case révélée contient un zéro [1], alors on révèle aussi les 8 cases autour de cette case, car on ne risque pas de réveler une mine. Si parmi ces 8 cases il y a encore

1. Dans les implémentations graphiques, au lieu d'afficher un chiffre '0', ces cellules sont simplement laissées vides ; dans notre implémentation, on affichera le '0' explicitement.

une case contenant un zéro, alors on révèle aussi ses voisins, et on continue ainsi autant que possible.

Le joueur a aussi la possibilité de poser un drapeau sur certaines cases pour se souvenir qu'il soupçonne qu'il y ait une mine sur cette case. Ce drapeau sert comme une sécurité : si le joueur demande à révéler une case avec un drapeau dessus, le jeu refuse de révéler la case. Le joueur doit d'abord enlever le drapeau.

Le jeu s'achève par une victoire lorsque toutes les cases dépourvues de mines sont révélées.

2 Les grandes lignes

On va implémenter quatre classes : `Lanceur`, `Jeu`, `Joueur` et `Plateau`.

`Lanceur` permettra de lancer le jeu. Sa fonction `main` demandera en boucle à l'utilisateur s'il veut jouer, si oui quels paramètres il veut (comme la taille du plateau, le nombre de mines), et lancera alors le jeu. Il reposera les mêmes questions quand le jeu s'arrête. Quand le joueur dira non, le programme s'arrêtera.

`Joueur` contiendra une fonction qui demandera en ligne de commande quelle action le joueur veut effectuer, ainsi que quelques éléments personnalisant le joueur (un nom, le nombre de parties gagnées et perdues ...).

`Plateau` contiendra des tableaux représentant l'état du jeu, ainsi que des fonctions pour agir sur le plateau, pour savoir si la condition de victoire est remplie, ou si on a perdu.

`Jeu` contiendra un `Joueur` et un `Plateau`, et fera interagir les deux entre eux.

3 Instructions

Construire le plateau

1. Créez une classe `Lanceur` qui contient la méthode `main`, qui affichera seulement le nom du jeu pour le moment. Nous utiliserons la classe `Lanceur` pour tester nos autres classes au fur et à mesure que nous les ajouterons.
2. Créez une classe `Plateau` qui contient les attributs suivants :
   - Quatre entiers `hauteur`, `largeur`, `nbMines` et `nbDrapeaux`.
   - Un tableau `private boolean[][] mines` qui indique où sont placées les mines.
   - Un tableau `private int[][] etats` qui enregistre les états courants de chaque case : la valeur `0` signfie une case cachée sans drapeau, `1` signifie cachée avec drapeau, `2` signifie révélée.
   - Un tableau `private int[][] adja` qui indique le nombre de mines adjacentes à chaque case.

   Créez un constructeur de `Plateau` qui prend en entrée une hauteur `ha`, une largeur `la` et le nombre de mines souhaités `mi`, et qui initialise les quatres variables entiers

et les trois tableaux de dimensions `ha` + 2 fois `la` + 2 (vous laisserez vides ces tableaux pour le moment, nous les remplisserons après).

Dans notre modélisation, le plateau du jeu sera stocké aux indices (1,1) à (ha, la). Les lignes supplémentaires seront utilisées pour faciliter les calculs pour les cases au bord, voir la q. 4 ci-dessous.

3. Créez une méthode `private void ajouteMinesAlea` qui ajoute `nbMines` dans le tableau de manière aléatoire. Faites attention :
   - ne posez pas plusieurs fois une mine à la même case ;
   - ne posez que des mines qui sont sur des cases sur le plateau, et pas sur l'une des lignes supplémentaires.

   Vous pouvez utiliser la méthode `nextInt` de la classe `Random`. Voir TP1 pour un exemple detaillé.
4. Créez une méthode `private void calculeAdjacence` qui remplit le tableau d'adjacence `adja`. Notez que vous n'avez pas besoin de distinguer les cases au bord, grâce aux lignes supplémentaires dans le tableau `mines`. Dans les cases aux lignes supplémentaires du tableau `adja`, mettez la valeur -1.
5. Ajoutez deux lignes au constructeur `Plateau` pour qu'il remplisse `mines` et `adja`.
6. Écrivez une méthode `public void afficheTout` qui affiche le plateau entier : une étoile pour une case contenant une mine, et pour les autres cases le nombre de cases voisines contenant une mine. Vous pouvez faire un affichage textuel avec la fonction *print*. Pour rappel, le caractère `\n` fait un retour à la ligne ("newline"). Par exemple :

```
   ********************
   * Mines / Drapeaux *
   *    10  /  0      *
   ********************
    1  2  3  4  5  6  7  8
--------------------------
A | *  3  *  1  0  0  0  0
B | *  3  1  1  0  0  1  1
C | 2  2  0  0  0  0  1  *
D | *  2  0  1  1  1  1  1
E | *  3  2  2  *  1  0  0
F | 2  *  2  *  2  1  0  0
G | 1  1  3  2  2  0  0  0
H | 0  0  1  *  1  0  0  0
```

7. Pour tester votre code jusqu'à ici, mettez les lignes suivantes dans la méthode `main` de la classe `Lanceur`, et vérifiez que tout fonctionne comme il faut.

```
Plateau p = new Plateau(8, 8, 10);
p.afficheTout();
```

Modifier les cases du plateau

8. Toujours dans la classe `Plateau`, créez une méthode `public void revelerCase` qui révèle la case dont on donne les coordonnées en entrée, en modifiant le tableau `etats`. Si la case contient un drapeau ou était déjà révelée, afficher un message.
9. (* un peu plus difficile) Modifiez la méthode `revelerCase` pour que, dans le cas où la case révélée n'a aucune mine adjacente, les 8 cases adjacentes soient révélées, et ainsi de suite.

   Indication. Utilisez la récursion. Pensez à vérifier que vous ne révelez que des cases qui sont sur le plateau. En cas de problèmes, vous pouvez toujours revenir à la version de la question précedente, et vous pouvez sauter cette question.
10. Créez une méthode `public void drapeauCase` qui pose/enlève (selon que la case a déjà un drapeau ou pas) un drapeau sur la case dont on a donné les coordonnées. La méthode doit aussi mettre à jour la variable `nbDrapeaux`. Si on essaie de mettre un drapeau sur une case déjà révelée, la méthode montre un message d'erreur, et ne modifie pas le plateau.
11. Créez une méthode `public void afficheCourant` qui affiche le tableau dans l'état courant, le nombre de mines (cachées) et le nombre de drapeaux placées. Le caractère `?` indique un drapeau et `.` une case cachée ; par exemple :

```
********************
* Mines / Drapeaux *
*   10  /  1       *
********************
  1 2 3 4 5 6 7 8
A . . . . . . . .
B . . 1 1 1 ? . .
C . . 1 0 1 1 . .
D . . 1 0 0 1 . .
E . . 2 1 0 1 . .
F . . . 2 1 1 . .
G . . . . . . . .
H . . . . . . . .
```

12. Créez des méthodes `public boolean jeuPerdu` et `public boolean jeuGagne` qui vérifient respectivement si le jeu est perdu (une mine a été révélée) et si le jeu est gagné (toutes les cases sans mines ont été révélées).

La classe Joueur

13. La classe `Joueur` contient un attribut `nom` de type `String` et un attribut `scanReponse` de type `Scanner` qui permet de lire ce que le joueur rentre au clavier. Écrivez un constructeur sans arguments qui met par défaut le nom du joueur à `"Anonyme"` et

qui attribue à `scanReponse` un nouveau `Scanner` qui opère sur l'entrée standard (`System.in`).

Indication. N'oubliez pas ajouter `import java.util.Scanner` en première ligne de la classe. Pour un exemple de l'utilisation de `Scanner`, cherchez-le dans la documentation de Java.

14. Écrivez deux méthodes auxiliaires, d'une ligne chacune : `public void setNom(String nom)` pour changer le nom du joueur, et `public void finish()` qui ferme `scanReponse`. Utilisez la méthode `close()` de la classe `Scanner`.
15. Écrivez des méthodes qui demandent des informations au joueur :
    - `public boolean veutJouer()` :
      (“`Voulez-vous jouer (oui/non) ?`”),
    - `public String demanderNom()`,
    - `public int[] demanderDimensions()` et `public int demanderNbMines()`,
    - `public char demanderAction()` :
      (“`Voulez-vous reveler une case (r) ou placer un drapeau (d) ?`”).
    - `public int[] demanderCoordonnes()`, qui demande les coordonnées sous la forme ‘B6’ et les renvoie sous la forme d'un tableau de longueur deux ; par exemple, si le joueur entre ‘B6’, la méthode renvoie le tableau d'entiers [2, 6].

    Attention : les méthodes que vous devez écrire ici ne modifient pas les attributs du joueur (comme son nom), ni d'un plateau ! Elles ne demandent que une information, vérifient le format, et renvoient le résultat avec un `return`. Vous appelerez ces méthodes dans les classes `Jeu` et `Lanceur`, voir ci-dessous.

    Indication. Pour rendre votre code plus clair, écrivez des méthodes auxiliaires, comme par exemple `private String demanderStr(String q)` qui pose une question au joueur et retourne la `String` donnée en entrée, et une méthode similaire, `private int demanderInt(String q)`, qui demande un entier au joueur.

Les classes Jeu et Lanceur

16. Créez la classe `Jeu`, qui a comme attributs un `Joueur` et un `Plateau`, et ajoutez un constructeur qui prend un `Joueur` et un `Plateau` en entrée.
17. Ajoutez à la classe `Jeu` une méthode `jouer` qui gère les différentes étapes de la partie. Une fois que la partie est finie, demander au joueur s'il veut rejouer. Réfléchissez bien à comment construire la méthode `jouer` en fonction du déroulement d'une partie. En particulier, aidez-vous des méthodes que vous avez créées dans la question 15.
18. Enfin, dans la classe `Lanceur`, dans le `main`, écrivez les instructions suivantes :
    - Créer un `Joueur`, lui demander son nom, s'il veut jouer, et, si oui, les paramètres du jeu.

— Lancer une partie (en appelant la méthode `jouer()`) et montrer le résultat quand la partie est terminée.

— Quand le joueur ne veut plus jouer, utiliser la méthode `finish()` de la classe joueur et quitter le programme.

4 Pour aller plus loin ...

4.1 Coopération

Si vous vous êtes contenté d'implémenter les méthodes publiques que nous vous avons conseillées, vous devriez pouvoir mélanger votre code avec celui de vos camarades. À la fin du TP, lorsque votre implémentation et celle d'un de vos camarades fonctionnent, essayez de prendre vos classes `Lanceur`, `Joueur`, `Jeu`, mais d'utiliser la classe `Plateau` de votre camarade. Ou encore mieux : prenez chaque classe chez quelqu'un de différent !

L'un des buts de la programmation orientée objet est de permettre ce genre d'échange, car on peut ainsi se répartir le travail sur un gros projet. Si votre code et celui d'un de vos camarades ne sont pas compatibles, essayez de voir ce qui bloque et de corriger le problème.

4.2 Optimisation de la fin de partie

Avec les méthodes `jeuPerdu` et `jeuGagne`, il faut reparcourir le plateau à chaque tour pour vérifier si la partie est terminée. Modifiez votre code pour savoir quand le jeu est perdu ou gagné sans avoir à reparcourir tout le plateau à chaque fois.

Indications. Pour optimiser `jeuGagne`, vous pouvez ajouter un attiribut au plateau pour compter le nombre de cases révelées. Pour optimiser `jeuPerdu`, vous pouvez vérifier si le joueur révèle une mine au moment où la case est révelée.

4.3 Utiliser une vraie interface graphique

Quand vous en saurez assez sur les interfaces graphiques, essayez de modifier le code du démineur pour qu'il s'affiche avec une interface graphique plutôt qu'un simple affiche textuel !