Université Paris Diderot **Programmation Orientée Objet**
L2 **Année 2016-2017**

TP - Séance n°3

Démineur

Dans ce TP, nous vous donnons un déroulé pas à pas (dans la section Instructions), ainsi que des fichier Java desquels vous pouvez partir. Ces derniers sont téléchargeables sur Moodle.

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.

Au départ, le terrain est complètement invisible. A 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.

Une case révélée indique le nombre de mines dans les 8 cases adjacentes à la case révélée. S'il n'y a aucune mine dans les 8 cases adjacentes, alors on révèle aussi les 8 cases autour de la première case révélée, car il n'y a aucun risque à le faire. Si dans ces 8 cases il y a encore une case sans mines autour d'elle, alors on révèle aussi ses voisins etc etc ...

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

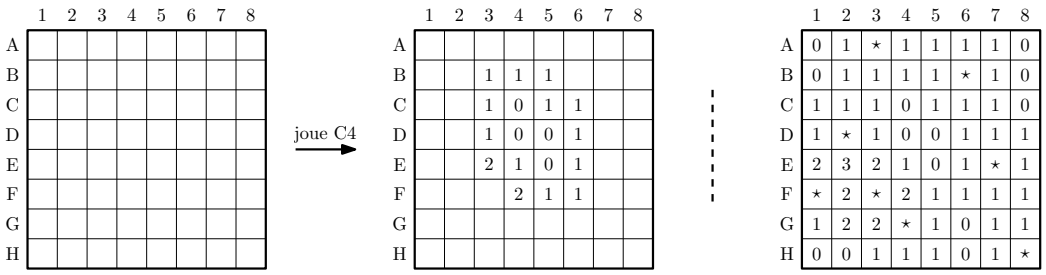

FIGURE 1 – À gauche, le joueur joue son premier coup en C4 et découvre toute une zone. À droite, le terrain complètement découvert

`Joueur` contiendra une fonction qui demandera en ligne de commande quelle action le joueur veut effectuer, ainsi qu'éventuellement 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

La classe `Lanceur` contient le main qui vous permettra de tester votre code petit à petit.

1. La classe `Plateau` contient les attributs suivants :
   — Trois entiers `hauteur`, `largeur` et `nbMines`
   — Un tableau `private final boolean[][] mines` qui indique où sont placées les mines.
   — Un tableau `private final int[][] etats` qui enregistre les états courants de chaque case (cachée avec/sans drapeau ou révélée)
   — Un tableau `private final int[][] adja` qui indique le nombre de mines adjacentes à chaque case

   Créez un constructeur qui prend en entrée les hauteur, largeur et nombre de mines souhaités.
2. Créez une méthode `private void ajouteMinesAlea` qui ajoute `nbMines` dans le tableau. On vous suggère d'utiliser la méthode `Random.nextInt` pour ce faire. `Random.nextInt(n)` renvoie un entier aléatoire entre 0 et $n-1$. Ensuite créez une méthode `private void calculeAdjacence` qui remplit le tableau d'adjacence.
3. Créez une méthode `private void revelerCase` qui révèle la case dont on a donné les coordonnées. *Bonus :* 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 : le plus facile est de le faire récursivement.

4. Créez une méthode `public String affichage` qui affiche le tableau dans l'état courant. Pour l'affichage, nous vous proposons de faire un affichage textuel (c'est-à-dire en utilisant la fonction *print*), où `'?'` indique un drapeau et `'.'` une case cachée (cf exemple).

```
****************
*Mines/Drapeaux*
*  8  /  1     *
****************
  12345678
A ........
B ..111?..
C ..1011..
D ..1001..
E ..2101..
F ...211..
G ........
H ........
```

   Utile : pour sauter des lignes dans une chaîne de caractère, on utilise le caractère spécial `'\n'`. (Peut être différent sous Windows, Mac, ou dans un éditeur particulier)

5. Créez une méthode `public boolean agir`, qui prend des coordonnées et une action en entrée et qui réalise l'action choisie sur la case choisie. Dans cet énoncé, nous vous proposons de représenter une action du joueur sur le plateau par un triplet d'entiers `{x,y,action}`. Il est également possible de définir une classe `Action` qui contiendrait trois champs (les 2 coordonnées, et l'action effectuée). On renvoie `true` si l'action était valide et `false` sinon. Si vous choisissez de représenter des actions ou les états des cases par des entiers, il peut être utile de définir des constantes entières de la façon suivante : `final int POSER_DRAPEAU = 2`. On pourra ainsi habilement utiliser un `switch case` comme ceci :

```
switch(typeAction) {
    case POSERDRAPEAU :
        //du code
    break;
    case REVELER :
        //du code
    break;
```

```
        //encore d'autres cas
        default :
        break;
    }
```

Note : On pourra alternativement utiliser des `enum`, si cela vous est familier.

6. Créez des méthodes `public boolean jeuFini` et `public boolean jeuGagne` qui vérifient respectivement si le jeu est fini et si le jeu est gagné.
7. La classe `Joueur` contient un attribut nom. Le constructeur initialise un scan qui pourra vous être utile par la suite, et met par défaut le nom du joueur à `"Anonyme"`. Ecrivez des méthodes `setNom` et `getNom` pour changer et renvoyer le nom du joueur.
8. Ecrivez les méthodes suivantes :
   — Une méthode `nombreChoisi` qui demande un nombre au joueur et le retourne. Si le joueur ne répond pas par un nombre, reposer la question.
   — Une méthode `ouiNon` qui demande au joueur de répondre par "oui" ou par "non". Si le joueur dit oui, retourner true, s'il dit non retourner false. Sinon reposer la question.
   — Une méthode `actionChoisie` qui demande au joueur une action choisie, sous la forme d'un triplet d'entiers (x,z,action). Si la réponse est invalide, reposer la question
9. La classe `Jeu` contient en attributs un joueur et un plateau. Ecrivez le constructeur correspondant ainsi qu'une méthode jouer qui lance une partie, et qui gère les différentes étapes de la partie. Un 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 les questions précédentes.
10. 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
    — Quand le joueur ne veut plus jouer, 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. Á 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 révélation des cases

Pour révéler les cases (quand on révèle des zones entières d'un coup), on pourra procéder différemment qu'utiliser une fonction récursive. Une manière est par exemple d'utiliser une pile (en utilisant la classe Java `Stack` par exemple) ou une file qui contient les cases sur lesquelles on doit passer, puis de la remplir et de la vider petit à petit. Pour de grands plateaux, cela peut beaucoup améliorer la vitesse et l'utilisation de mémoire du programme.

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 !