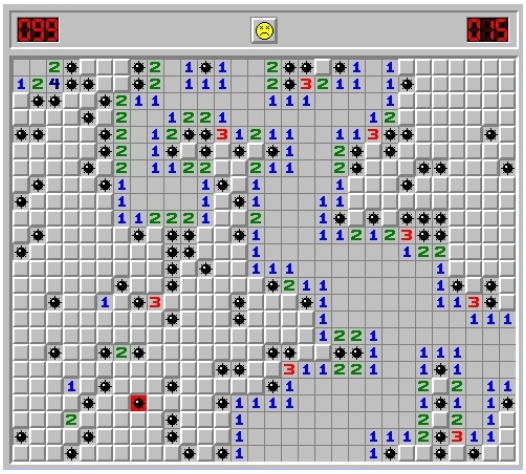
TP Bonus – Partie 1/2

Démineur en mode console

1 – Présentation du TP

Objet du TP

Le jeu de démineur a longtemps été un classique sous Windows. Le plateau de jeu est une grille dont le contenu des cases est initialement masqué. Chaque case contient ou non une mine.



Le but du jeu est de localiser toutes les mines, mais sans cliquer dessus. Pour aider le joueur, le jeu affiche le nombre de mines touchant les cases déjà dévoilées.

Ce TP consiste à implémenter une version en mode console de ce jeu : le plateau de jeu est affiché sous forme de texte et les commandes sont saisies au clavier.

Dans une deuxième partie, ce jeu sera repris en mode graphique : le jeu sera alors affiché dans une fenêtre et l'utilisateur pourra cliquer sur les cases du démineur.

2 – Démarrage du TP

Suivez la procédure de démarrage du TP 1, en remplaçant le nom du projet « tp1 » par « minesweeper ». Toutes les autres actions sont identiques.

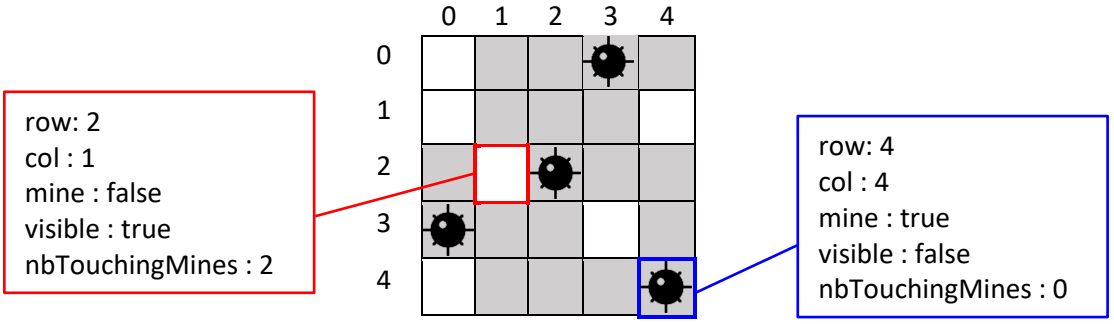
3 – Case du démineur

Modélisation

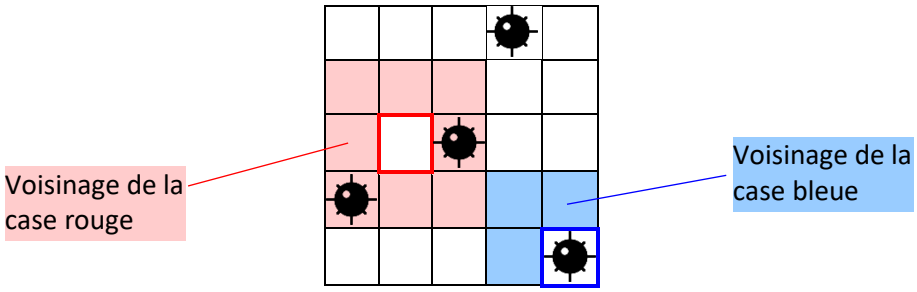
Une case du jeu de démineur est modélisée par une classe « Cell » ayant les attributs suivants :

Nom	Type	Description
row	Nombre entier	Ligne de la case dans la grille (commence à 0)
col	Nombre entier	Colonne de la case dans la grille (commence à 0)
mine	Booléen	Indique si la case contient une mine.
visible	Booléen	Indique si le contenu de la case est visible.
nbTouchingMines	Nombre entier	Nombre de mines dans le voisinage de la case

La figure ci-après illustre ces attributs sur une grille 5 x 5. Les cases masquées sont illustrées en gris.



On considère que le voisinage d'une case est constitué des 8 cases en contact avec cette case, comme illustré en rouge ci-dessus. Ce voisinage peut contenir moins de cases sur les bords, comme illustré en bleu.



Implémentation

Ajoutez une classe « Cell » à votre projet.

Déclarez les attributs décrits ci-dessous, avec les types appropriés et une visibilité privée.

Implémentez le constructeur suivant :

```
public Cell(int row, int col)
```

qui initialise les attributs `row` et `col` à partir des paramètres, et les autres attributs avec les valeurs par défaut suivantes :

- Pas de mine dans la case ou dans le voisinage de la case
- Contenu de la case non visible

Implémentez ensuite les accesseurs suivants :

Attribut	Accesseur en lecture	Accesseur en écriture
row	getRow	
col	getCol	
mine	isMine	setMine
visible	isVisible	setVisible
nbTouchingMines	getNbTouchingMines	setNbTouchingMines

Deux choses sont notables ici :

- Dans le cas particulier des booléens, on utilise le nom « is + nom de l'attribut » pour l'accesseur en lecture plutôt que « get + nom de l'attribut ».
- Il n'y a pas d'accesseur en écriture pour les coordonnées de la case, qui sont fixées à la construction. L'encapsulation permet de rendre certains attributs en lecture seule.

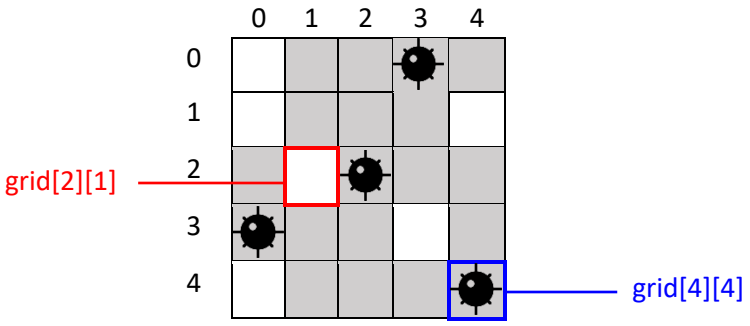
4 – Grille du démineur

Modélisation

La grille du démineur est modélisée par une classe « MineSweeper » ayant les attributs suivants :

Nom	Type	Description
nbRows	int	Nombre de lignes
nbCols	int	Nombre de colonnes
grid	Cell[][]	Cases de la grille

Les cases de la grille sont stockées dans un tableau à 2 dimensions. Chaque case de ce tableau est une instance de la classe Cell. Le premier niveau de crochets correspond aux lignes, et le deuxième aux colonnes. Ainsi, `grid[i][j]` contient la case à la ligne `i` et à la colonne `j` de la grille, comme illustré ci-dessous :



Récupération de la classe MineSweeper

Pour vous faire gagner du temps, on vous fournit une classe MineSweeper partiellement implémentée. Cette classe contient les attributs décrits ci-avant, ainsi que quelques attributs et méthodes qui vous serviront ultérieurement.

Récupérez le fichier MineSweeper.java sur la page Moodle du cours et ajoutez-le à votre projet par l'un des moyens suivants (choisissez celui que vous préférez) :

1. Glisser-déposer le fichier depuis l'explorateur de fichiers vers le répertoire « src » de votre projet dans IntelliJ
2. Sélectionnez le fichier dans l'explorateur, appuyez sur CTRL + C, sélectionnez ensuite le répertoire « src » dans de votre projet dans IntellJ et appuyez sur CTRL + V
3. Ouvrez le répertoire de votre projet dans l'explorateur de fichiers, et copiez le fichier dans le répertoire « src » ; le projet se mettra à jour automatiquement dans IntelliJ.

Implémentation du constructeur

Implémentez le constructeur suivant :

```
public MineSweeper(int nbRows, int nbCols)
```

qui :

- Initialise les attributs `nbRows` et `nbCols`
- Initialise le tableau `grid`
- Initialise chaque case du tableau `grid` (chaque case du tableau doit contenir une instance de la classe Cell, avec les bonnes coordonnées)

Test

Pour tester le bon fonctionnement de vos classes, instanciez la classe MineSweeper dans le main(), et affichez le jeu en utilisant la méthode print() de cette classe, déjà implémentée pour vous.

Si tout fonctionne correctement, chaque case du jeu sera affichée avec le caractère `#`, symbolisant que son contenu est masqué.

Par exemple, pour un jeu de dimensions 4 x 6, le programme devrait produire l'affichage :

```
  0 1 2 3 4 5
0 # # # # # #
1 # # # # # #
2 # # # # # #
3 # # # # # #
```

5 – Placement des mines

Méthode de placement des mines

Dans la classe MineSweeper, écrire une méthode :

```
public void putMines(int nbMines)
```

qui place aléatoirement `nbMines` mines dans la grille.

Aide : Supposons que `nbRows` et `nbCols` représentent respectivement le nombre de lignes et de colonnes de la grille. Le code suivant permet de générer des coordonnées aléatoires dans la grille :

```
import java.util.Random;            // à placer en début de fichier

...

Random random = new Random();
int row = random.nextInt(nbRows);   // ligne aléatoire entre 0 et nbRows
int col = random.nextInt(nbCols);   // colonne aléatoire entre 0 et nbCols
```

La classe MineSweeper possède des drapeaux de debug, qui permettent d'afficher des informations normalement cachées.

Pour tester le bon fonctionnement de votre fonction :

- Fixez la valeur de l'attribut `showMines` à `true`.
- Ajoutez un appel à putMines() dans le main() avant l'appel à print()

Les mines devraient être affichées avec le symbole `*`.

Par exemple, si on place 5 mines dans une grille 4 x 6, on obtient un affichage de la forme :

```
  0 1 2 3 4 5
0 # # * # # #
1 # # # * # *
2 # # # * * #
3 # # # # # #
```

Note : Si vous observez moins de mines que prévu, c'est que le hasard a fait que plusieurs mines ont été placées au même endroit. Comment pouvez-vous corriger ceci ?

Placement des mines dans le constructeur

Ajoutez un paramètre `nbMines` dans le constructeur de la classe MineSweeper.

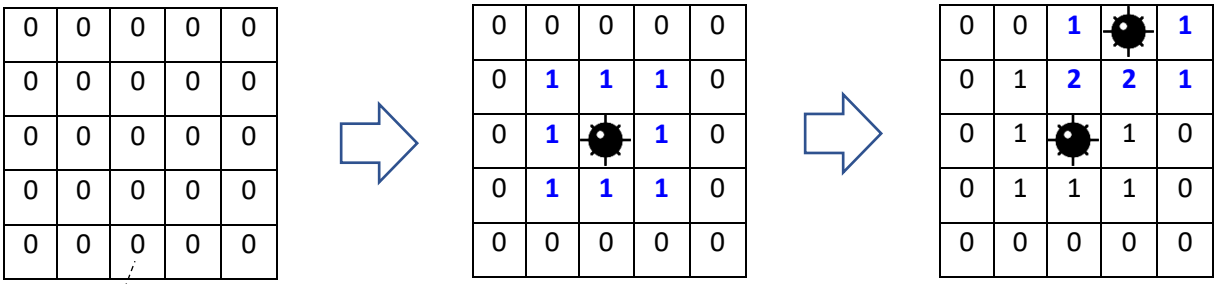
Appelez la méthode putMines() dans le constructeur.

Dans le main(), remplacez l'appel à putMines() avec un appel au constructeur de MineSweeper avec un paramètre `nbMines`.

Relancez votre programme pour vérifier qu'il a un fonctionnement équivalent à précédemment.

6 – Calcul des mines en contact

Modifiez le code de la méthode putMines() pour qu'elle incrémente l'attribut `nbTouchingMines` des cases voisines à chaque fois qu'elle dépose une mine dans une case, comme illustré ci-dessous (les mises à jour des attributs `nbTouchingMines` sont mises en évidence en bleu) :



Valeur de l'attribut `nbTouchingMines` de la case

Pour ce faire, vous pouvez vous appuyer sur la méthode getNeighbors() fournie, qui renvoie les voisins d'une case sous la forme d'une liste de type LinkedList. Pour parcourir les éléments d'une liste, vous pouvez utiliser une boucle de type « for each ».

Pour tester le bon fonctionnement de votre code, vous pouvez fixer la valeur du drapeau `showNbTouchingMines` de la classe MineSweeper à `true`. Le nombre de mines en contact sera alors affiché.

Sur l'exemple précédent, on obtiendrait un affichage de la forme :

```
   0 1 2 3 4 5
0  0 1 * 2 2 1
1  0 1 3 * 4 *
2  0 0 2 * * 2
3  0 0 1 2 2 1
```

Si tout fonctionne, remettez les drapeaux `showMines` et `showNbTouchingMines` à `false`.

7 – Dévoilement d'une case

Ajoutez une méthode à la classe MineSweeper :

```
public void unveil(int row, int col)
```

qui dévoile le contenu de la case située aux coordonnées (row, col).

Concrètement, ceci revient à fixer la valeur de l'attribut visible de la case correspondante à `true`.

Testez votre fonction en faisant des appels à unveil() dans le main().

8 – Commandes de l'utilisateur

Ajoutez une méthode à la classe MineSweeper :

```
public void play()
```

qui commence une partie de démineur.

Dans une boucle infinie, la méthode play() va réaliser les actions suivantes :

- Afficher le jeu
- Attendre que le joueur saisisse les coordonnées de la case à dévoiler
- Dévoiler la case indiquée

Pour faire simple, la commande saisie par l'utilisateur pourra être constituée du numéro de ligne puis du numéro de colonne séparés par une espace.

Par exemple, si l'utilisateur saisit la commande :

```
1 3↵
```

Alors le jeu dévoilera la case aux coordonnées (1, 3).

Pour lire les coordonnées saisies par le joueur, on pourra utiliser la classe Scanner, et faire deux appels à la méthode nextInt() : un pour le numéro de ligne, et un pour le numéro de colonne.

Gestion des erreurs

Placez la gestion des commandes du joueur dans un `try/catch` pour gérer les saisies invalides (par exemple, une commande qui ne serait pas un nombre). Afficher un message d'erreur dans le `catch` pour indiquer au joueur de recommencer sa saisie.

9 – Fin de partie

Faites en sorte que le jeu se termine si une des conditions suivantes est vérifiée :

1. L'utilisateur dévoile une case contenant une mine : dans ce cas, on indique au joueur qu'il a perdu, et on affiche l'emplacement des mines en changeant la valeur du drapeau `showMines`.
2. Le joueur a dévoilé toutes les cases ne contenant pas de mines (les seules cases restantes sont celles contenant des mines) : dans ce cas, on indique à l'utilisateur qu'il a gagné.

Dans les deux cas, on interrompt la boucle de traitement des saisies du joueur (préférez l'utilisation d'un booléen).

La détection du cas n°2 nécessite d'ajouter un attribut dans la classe MineSweeper, qui sera mis à jour au cours de la partie.

10 – [Facultatif] Améliorations

Dévoilement des cases en cascade

Dévoiler les cases une à une peut rapidement devenir fastidieux, alors que dans certains cas, on peut le faire sans réfléchir. En effet, on sait que si une case ne touche aucune mine, on peut dévoiler sans crainte toutes les cases voisines : dans le pire des cas, on tombera sur une case *touchant* une mine, mais jamais une case *contenant* une mine.

Cette amélioration a pour but de dévoiler automatiquement les cases dans ces cas triviaux.

L'algorithme permettant de faire ceci s'appuie sur une file :

```
Créer une file vide
Ajouter la case de départ (supposée ne pas toucher de mines) dans la file
Tant que la file n'est pas vide, faire :
   -  Récupérer la case en tête de file
   -  Si la case est masquée, alors :
      -   Dévoiler la case
      -   Si la case ne touche aucune mine, alors ajouter les voisins de
          la case dans la file
```

Le résultat de cet algorithme est illustré ci-dessous : à partir de la case de départ, les cases sont dévoilées de proche en proche jusqu'à constituer un front de cases en contact avec les mines (constitué de chiffres). Ce front sépare la partie dévoilée de la partie non dévoilée.

Situation de départ (drapeau `showMines` activé)	Situation après dévoilement de la case (0,0)

`  0 1 2 3 4 5`
`0 # # * # # #`
`1 # # # # # *`
`2 # # # # * #`
`3 # # # # # #`
	`  0 1 2 3 4 5`
	`0   1 * # # #`
	`1   1 1 2 # *`
	`2       1 * #`
	`3       1 # #`

Modifier le code de la méthode unveil() pour implémenter l'algorithme décrit ci-dessus.

Aide :

- Réutiliser la méthode getNeighbors().
- On peut utiliser une LinkedList comme une file avec les méthodes suivantes :
  - Ajout en queue de liste : add()
  - Récupération de la tête de la liste : poll() – renvoie `null` si la liste est vide

Premier coup assuré

Faites en sorte que les mines ne soient placées dans la grille que lorsqu'une première case est dévoilée. Ceci permet d'assurer que le premier coup est toujours sûr.

Marquage des cases

Faites en sorte que le joueur puisse marquer les cases de la manière suivante :

- ! : case contenant une mine
- ? : case indécise

Pour cela :

- Modifiez la méthode play() de MineSweeper pour que les commandes du jeu deviennent :

Commande	Signification
`d i j`	Dévoiler la case (i, j)
`m i j !`	Marquer la case (i, j) comme contenant une mine
`m i j ?`	Marquer la case (i, j) comme étant indécise
`m i j #`	Supprimer tout marquage sur la case (i, j)
`q`	Quitter le jeu

- Modifiez la classe Cell pour ajouter la notion de marquage
- Modifiez la méthode getCellSymbol() de MineSweeper pour tenir compte du marquage de la case

Aide : Pour découper les différents parties de la commande, vous pouvez utiliser la méthode split() de la classe String.