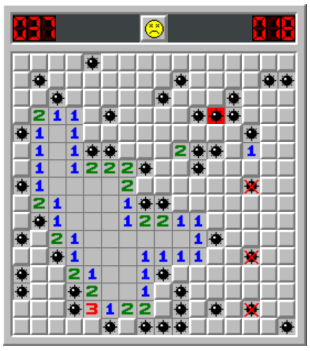
INF1 - Projet **Démineur** UNIVERSITÉ DE RENNES 1

Le but de ce projet est d'implémenter le jeu Démineur en Java. Si vous ne connaissez pas le jeu, vous pouvez facilement en trouver en ligne, par exemple http://demineur.org/.

Dans ce projet, **on ne fera pas d'interface graphique**, tout se fera directement par de l'affichage dans le terminal/Eclipse. Voici par exemple deux parties (toutes les deux perdues), jouées avec la fonction `main` remplie à la question 4.e] en fin de projet :

```
$ java minesweeper
Hauteur de la grille :
3
Largeur de la grille :
3
Nombre de mines :
1
  |A|B|C|
 0| | | |
 1| | | |
 2| | | |

Saisir la commande :
r0A
Perdu !!!
  |A|B|C|
 0|!| | |
 1| | | |
 2| | | |
```

```
$ java minesweeper
Hauteur de la grille :
4
Largeur de la grille :
4
Nombre de mines :
1
  |A|B|C|D|
 0| | | | |
 1| | | | |
 2| | | | |
 3| | | | |

Saisir la commande :
r0A
  |A|B|C|D|
 0|0|0|0|0|
 1|0|1|1|1|
 2|0|1| | |
 3|0|1| | |
Saisir la commande :
r3D
  |A|B|C|D|
 0|0|0|0|0|
 1|0|1|1|1|
 2|0|1| | |
 3|0|1| |1|
Saisir la commande :
r2C
Perdu !!!
  |A|B|C|D|
 0|0|0|0|0|
 1|0|1|1|1|
 2|0|1|!| |
 3|0|1| |1|
```

On utilisera deux tableaux 2D (`int[][]`) pour représenter la grille de jeu :

— Le premier tableau `T` est un tableau de tableaux d'entiers permettant de représenter l'état de chaque case. Une case peut être révélée (ou non), ou bien marquée d'un drapeau (afin de pouvoir marquer les cases que le joueur suppose contenir des mines). On choisit de représenter cela de la manière suivante :
  — `T[i][j] = 0` si la case n'est pas révélée.
  — `T[i][j] = 1` si la case est révélée.
  — `T[i][j] = 2` si la case a été marquée par un drapeau.
— Le deuxième tableau `Tadj` est un tableau de tableaux d'entiers, qui va permettre de savoir si la case (i, j) est une mine, ou bien le nombre de mines adjacentes. C'est à dire (voir question 1.d] pour un exemple) :
  — Si `Tadj[i][j]` est égal à `-1`, alors la case (i, j) contient une mine.
  — Sinon, `Tadj[i][j]` contient un chiffre entre `0` et `8` égal au nombre de mines dans les cases adjacentes.

Exercice 1 : Initialisation des tableaux

1.a] Prenez connaissance du fichier squelette `projet_demineur.java`, qui contient des commentaires et une organisation globale à respecter : une fonction à remplir, question par question. Les signatures des fonctions sont toutes de type retour `void` sans argument, et sont à modifier. Ajoutez vos deux Prénoms Noms et Groupe en haut du fichier.

Dans votre programme, les variables `T` et `Tadj` sont des variables *globales*, définies en dehors des fonctions et de la fonction `main`. Elles sont déclarées au début, et toutes les deux de types `int[][]` (et le mot clé `static`). Il faut les initialiser aux bonnes dimensions `hauteur`, `largeur` en 1.b].

Remarque sur la fonction main : Dans ce projet, l'écriture de la fonction `main` permettant de jouer au jeu de démineur fait l'objet d'une question spécifique 4.e] en fin d'exercice 4. Par contre, à chaque question quand vous compléterez, nous vous recommandons (très fortement) d'utiliser la fonction `main` pour effectuer des tests et vous aider à vérifier le bon comportement de chaque question.

1.b] Écrire une fonction `init` qui prend en paramètres trois entiers `h`, `l` et `n` et ne renvoie rien. Cette fonction doit en premier initialiser les tableaux `T` et `Tadj` en des tableaux ayant `hauteur` lignes et `largeur` colonnes (`hauteur`, `largeur` ≥ 1). Ensuite, elle doit placer exactement `n` mines au hasard dans le tableau (`n` ≥ 1), en remplissant des cases de `Tadj` à `-1` si elles contiennent une mine, ou `0` sinon (la question 1.d] s'occupera de calculer le nombre de mines dans les cases adjacentes, pour les cases n'ayant pas de mines). Vous utiliserez la fonction `entierAleatoire(a,b)` (utilisée au TP2), qui renvoie un entier aléatoire uniforme entre `a` (inclus) et `b` (inclus). Pour le tableau `T`, bien évidemment, au départ, toutes les cases sont non révélées.

Par exemple, avec une grille de dimensions `hauteur = 11`, `largeur = 10` et `n = 12` mines aléatoirement réparties, le tableau `Tadj` sera rempli comme cela (voir question 1.d] pour le calcul entier de `Tadj`) :

```
 0   0   0   0   0   0  -1  -1   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0  -1   0   0   0   0   0
 0   0   0   0   0   0   0   0   0  -1
 0   0  -1   0   0   0   0   0   0   0
 0   0   0  -1   0   0   0  -1  -1   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0  -1   0   0
 0   0   0   0   0   0   0   0   0   0
-1   0   0  -1   0   0   0  -1   0   0
 0   0   0   0   0   0   0   0   0   0
```

1.c] Écrire une fonction `caseCorrecte` qui prend en paramètres deux entiers `i` et `j` et qui renvoie un booléen. Cette fonction doit renvoyer `true` si les indices `i` et `j` désignent des coordonnées valides pour les dimensions `hauteur * largeur` des deux tableaux `T` et `Tadj`, et `false` sinon. Elle sera très utile pour la plupart des autres fonctions nécessaires au projet.

1.d] Écrire une fonction `calculerAdjacent` qui ne prend aucun paramètre et ne renvoie rien. Cette fonction doit remplir le tableau `Tadj` avec le nombre de mines adjacentes. Ainsi, le tableau final sera par exemple :

```
 0   0   0   0   0   1  -1  -1   1   0
 0   0   0   1   1   2   2   2   1   0
 0   0   0   1  -1   1   0   0   1   1
 0   1   1   2   1   1   0   0   1  -1
 0   1  -1   2   1   0   1   2   3   2
 0   1   2  -1   1   0   1  -1  -1   1
 0   0   1   1   1   0   2   3   3   1
 0   0   0   0   0   0   1  -1   1   0
 1   1   1   1   1   0   2   2   2   0
-1   1   1  -1   1   0   1  -1   1   0
 1   1   1   1   1   0   1   1   1   0
```

Exercice 2 : Affichage de la grille

2.a] Écrire une fonction `afficherGrille` qui prend en paramètre un booléen `affMines` et ne renvoie rien. Cette fonction va devoir afficher l'état courant du jeu. Si une case n'est pas révélée, alors on affichera seulement un espace. Si une case est marquée par un drapeau, alors on affichera le caractère `X`. Sinon, on affiche le nombre de mines adjacentes. Si le booléen `affMines` vaut `true`, alors on affichera également la position de toutes les mines de la grille avec un caractère `!`. Ceci sera utilisé quand la joueuse tentera de révéler une case contenant une mine (et aura donc perdu).

Par exemple, au tout début du jeu, la grille s'affichera ainsi (ce n'est pas la même grille qu'avant) :

```
  |A|B|C|D|E|F|G|H|I|J|
00| | | | | | | | | | |
01| | | | | | | | | | |
02| | | | | | | | | | |
03| | | | | | | | | | |
04| | | | | | | | | | |
05| | | | | | | | | | |
06| | | | | | | | | | |
07| | | | | | | | | | |
08| | | | | | | | | | |
09| | | | | | | | | | |
10| | | | | | | | | | |
```

Après avoir joué quelques coups, la grille ressemblera (par exemple) à la Figure 1 si `affMines = false`, et à la Figure 2 si `affMines = true`.

```
  |A|B|C|D|E|F|G|H|I|J|
00| | |1|0|0|0|0|2| | |
01| | |1|0|0|0|1|3| | |
02| | |1|0|0|0|1|X| | |
03| | |1|0|0|1|2| | | |
04| | |1|1|1|2|X| | | |
05| | | | | | |3| | |
06| | | | | | | | | |
07|1|1| | | | | |2| |
08|0|1| | | | | | | |
09|0|1| | | | | | | |
10|0|1| | | | | | | |
```

Figure 1

```
  |A|B|C|D|E|F|G|H|I|J|
00| | |1|0|0|0|0|2|!| |
01| |!|1|0|0|0|1|3|!| |
02| | |1|0|0|0|1|!| | |
03| | |1|0|0|1|2| | | |
04| |!|1|1|1|2|!| | | |
05| | | | |!| | |3|!| |
06|!| | | | | | |!| | |
07|1|1| | | | | |2| |
08|0|1| | | | | |!| | |
09|0|1|!|!| | | | | | |
10|0|1| | | | | | | | |
```

Figure 2

Note : on considérera que la grille contiendra au plus 52 colonnes (une pour chaque lettre de l'alphabet en majuscule et minuscule) et au plus 100 lignes (entiers de 0 à 99).

Exercice 3 : Révéler une case

3.a] Écrire une fonction `caseAdjacenteZero` qui prend en paramètres deux entiers `i` et `j` et qui renvoie un booléen. Cette fonction doit renvoyer `true` si au moins une case adjacente à la case de coordonnées (i, j) est révélée *et* n'a aucune mine adjacente (*i.e.* la valeur correspondante dans `Tadj` est zéro). Par exemple, en reprenant l'exemple donné précédemment, `caseAdjacenteZero(7, 1)` (qui correspond à la case `7B`) doit renvoyer `true`. On considérera que lors de l'appel de la fonction, les coordonnées (i, j) sont valides, par contre dans la suite, pensez bien à appeler la fonction `caseCorrecte(i, j)` avant d'appeler `caseAdjacenteZero(i, j)`.

3.b] Écrire une fonction `revelation` qui prend en paramètres deux entiers `i` et `j` et qui ne renvoie rien. Cette fonction doit d'abord révéler la case de coordonnées (i, j), en modifiant la case `T[i][j]` (on supposera encore que les indices sont valides lors de l'appel de la fonction). Ensuite, si la case de coordonnées (i, j) n'avait aucune mine adjacente (*i.e.* `Tadj[i][j] == 0`), alors il faut révéler les cases adjacentes qui ne sont pas des mines. Il faut ensuite recommencer si une des ces cases n'avait également aucune mine adjacente.

Pour ceci, l'idée est la suivante :

1. Commencer par révéler la case (i, j).
2. Si cette case n'a aucune mine adjacente alors :
   (a) Parcourir toute les cases de grille.
   (b) Si une case (non révélée) possède une case adjacente qui n'a aucune mine adjacente adjacente (*i.e.* la fonction `caseAdjacenteZero` renvoie `true`), alors on révèle également cette case.
   (c) Après avoir parcouru toute la grille, si on a révélé *au moins* une nouvelle case, alors on recommence à l'étape (a).

3.c] (**Challenge optionnel**) On peut remarquer que la méthode proposée pour la question **3.b** n'est pas très efficace. En effet, on est obligé de parcourir toute la grille, alors qu'on pourrait seulement regarder les cases adjacentes aux cases révélées. Proposer donc une nouvelle version de cette fonction qui soit plus efficace (sans nécessairement être optimale).

3.d] Écrire une fonction `actionDrapeau` qui prend en paramètres deux entiers `i` et `j`, et qui ne renvoie rien. Si la case de coordonnées (i, j) est déjà révélée (`T[i][j] == 1`), il est inutile d'ajouter ou enlever un drapeau. Sinon, on ajoute un drapeau si elle n'est pas marquée (et elle le devient), ou on enlève le drapeau si la case était déjà marquée. Rappel : `T[i][j] == 2` si la case est marquée d'un drapeau.

3.e] Écrire une fonction `revelerCase` qui prend en paramètres deux entiers `i` et `j`, et qui renvoie un booléen. La fonction doit renvoyer `false` si la case de coordonnées (i, j) contient une mine. Sinon, elle doit appeler la fonction `revelation`, puis renvoyer `true`.

Exercice 4 : Boucle de jeu

4.a] Écrire une fonction `aGagne` qui ne prend aucun paramètre et qui renvoie un booléen. Cette fonction doit renvoyer `true` si la joueuse a gagné. On considère qu'un joueur a gagné s'il a révélé toutes les cases de la grille sauf les mines.

4.b] Écrire une fonction `verifierFormat` qui prend en paramètre une chaîne de caractères et qui renvoie un booléen. Cette fonction doit vérifier que la joueuse a saisi une entrée au bon format, qui est le suivant :

— Si on veut marquer la case d'un drapeau, alors le format doit être `d + numéro de ligne + lettre de la colonne`. Par exemple, saisir `d02A` indiquera qu'on veut marquer (ou enlever le drapeau) la case `2A`. Le numéro de ligne sera *toujours* écrit sur deux chiffres (01, 02, ..., 09, 10, etc).
— Si on veut révéler une case, alors le format doit être `r + numéro de ligne + lettre de la colonne`. Par exemple, saisir `r10F` indiquera qu'on veut révéler la case `10F`.

La fonction doit renvoyer `true` si le format est valide, et `false` sinon. Pour vous aider à vérifier le format, noter que si le format est valide :

— Le premier caractère sera toujours soit `d` soit `r`.
— Le dernier caractère sera toujours une lettre, correspondant à la colonne.
— Les autres caractères doivent être des chiffres.

4.c] Écrire une fonction `conversionCoordonnees` qui prend en paramètre une chaîne de caractères `input` et qui renvoie un tableau d'entiers. On suppose que lors de l'appel à la fonction, la chaîne de caractères est au bon format (pensez donc à utiliser la fonction `verifierFormat` avant d'appeler cette fonction). La fonction doit renvoyer un tableau de trois entiers :

— Le premier élément correspond à l'indice de la ligne.
— Le deuxième élément correspond à l'indice de la colonne.
— le troisième élément vaut 0 si la joueuse a demandé à marquer la case, et 1 si il a demandé à révéler la case.

Par exemple, si la joueuse a saisi `d2A`, alors la fonction doit renvoyer le tableau {`2,0,0`}. Si le joueur a saisi `r10F`, alors la fonction doit renvoyer le tableau {`10,5,1`}. Note : pour passer d'une chaîne de caractère au nombre correspondant, vous pouvez utiliser la fonction `Integer.parseInt()`. Par exemple, `Integer.parseInt("10")` renverra la valeur `10` (en tant qu'entier).

4.d] Écrire une fonction `jeu` qui ne prend aucun paramètre et ne renvoie rien. Cette fonction doit permettre à la joueuse de jouer au jeu tant que la partie n'est pas finie (soit le joueur gagne, soit il révèle une mine et perd). Cette fonction doit donc :

— Afficher l'état courant de la grille.
— Demander à l'utilisateur de saisir les coordonnées de la case qu'il veut révéler au format donné ci-dessus.
— Vérifier que ces coordonnées sont correctes, et si ce n'est pas le cas redemander.
— Réalise l'action saisie par l'utilisatrice (à l'aide des deux fonctions `revelerCase` et `actionDrapeau`). Si elle demande à révéler une case contenant une mine, alors on doit afficher `Perdu...`, afficher la grille avec les mines, et la partie s'arrête.
— On continue tant que le joueur n'a pas gagné ni perdu. Si la joueuse gagne, alors à la fin on affiche `Gagné !` et le programme termine.

4.e] Écrire la fonction `main` afin de pouvoir jouer au jeu. Commencer par demander à l'utilisateur les dimensions de la grille ainsi que le nombre de mines à placer. Penser

à vérifier que les dimensions et le nombre de mines sont valides (avec un `Scanner`). Ensuite, on doit initialiser la grille (fonction `init`), remplir le tableau `Tadj` (fonction `calculerAdjacent`) puis lancer le fonction `jeu`.

Exercice 5 : Pour aller plus loin (**challenge**)

Remarque : si vous travaillez sur cette partie, vous pourrez obtenir des points bonus !

5.a] Créer une fonction permettant d'aider le joueur. Cette fonction devra être appelée si la joueuse saisi la chaîne `aide`, au lieu d'une saisie dans le format classique décrit à la question 4.c. Le choix est libre sur la manière d'aider le joueur, mais on pourra par exemple lui proposer une liste des cases qui contiennent forcément une mine d'après les informations obtenues par les cases révélées, ou bien lui donner une liste de case qui ne posent aucun risque (en s'aidant des drapeaux).

Par exemple, dans la situation suivante :

```
  |A|B|C|D|E|
00| | | | | |
01| | |2| | |
02| |1|3|1| |
03| |2| |1| |
```

On sait que la case `02C` a 3 mines adjacentes, mais il n'y a également que 3 cases non révélées. On est donc sûr que ces cases sont des mines, et vous pouvez indiquer cela à la joueuse.

Un autre exemple :

```
  |A|B|C|D|E|
00| | | | | |
01| | |2| | |
02|1|X|1|1| |
03|1|1| |1| |
```

Le joueur a marqué la case `02B` comme étant une mine. Ainsi, on peut lui indiquer que d'après sa décision la case `03C` ne devrait pas contenir de mine. La raison pour cela est que la case `03B` ne possède qu'une seule mine adjacente, et la case `02B` est marquée. Donc la case `03C` ne devrait pas contenir de mine. C'est également le cas pour les cases `01A` et `01B`. Vous pouvez donc également donner ce type d'information à la joueuse.

5.b] Créer un programme qui va jouer au jeu tout seul (comme une intelligence artificielle). Le programme doit tout faire pour ne pas perdre (mais il n'a pas le droit d'accéder au tableau `Tadj` pour vérifier si une case est une mine avant de jouer). Par exemple, dans la situation suivante

```
  |A|B|C|D|E|
00|1|1|1| | |
01|1| |1| | |
02|1|1|1|1| |
03|0|0|0|1| |
```

on sait que la case `01B` contient une mine. Le programme ne doit donc jamais chercher à révéler cette case (la question challenge précédente peut être utile). Il va donc chercher à révéler les cases qu'il sait ne pas contenir de mines. Si jamais il n'y a pas de case respectant ce critère, alors il pourra par exemple jouer un coup au hasard.

5.c] Générer 1000 grilles différentes, aléatoirement, selon les paramètres suivants et compter le nombre de fois où le programme arrive à résoudre la grille en entier.

— Une grille 8×8 avec $n = 10$ mines.
— Une grille 16×16 avec $n = 40$ mines.
— Une grille 30×16 avec $n = 99$ mines.