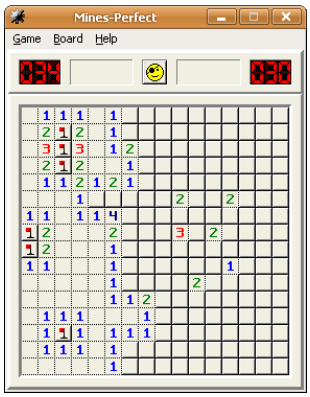
IS2A 3 - PPO - T.P. 8 : Démineur

> Si vous le souhaitez, vous pouvez rendre ce TP jusqu'au **lundi 30 mai 2022** par mail `bastien.cazaux@polytech-lille.fr`. Dans le rendu de TP, vous mettrez en plus de vos fichiers de codes (avec l'arborescence) le fichier `Grille_evaluation_TP8.ods` (que vous trouverez sur Moodle) que vous aurez au préalablement rempli (modifiez uniquement les cases vertes).

L'objectif de ce TP est de programmer un démineur. Ce TP se déroulera en **deux grandes parties** : une première partie sur l'implémentation d'un **noyau fonctionnel** correspondant au démineur et une seconde partie sur l'**interface graphique**. Une trame vous est proposée, mais d'autres méthodes seront peut-être nécessaires. Pour les grandes étapes du noyau fonctionnel, ajouter quelques tests pour permettre de vérifier votre code.

1 Principe du jeu

Pour commencer, si vous ne connaissez pas le jeu du démineur, voici les rêgles :



> *Le plateau est divisé en cases, avec des mines placées aléatoirement. Pour gagner, vous devez ouvrir toutes les cases qui ne contiennent pas de mines. Cliquer sur une case qui n'a pas de mine révélera un nombre. Ce nombre correspond au nombre de mines adjacentes à cette case. À l'aide de ces informations, vous pouvez déterminer les cases sûres et les cases contenant des mines.*

Vous pouvez tester une variante à l'adresse suivante : `https://www.google.com/search?q=jouer%20au%20d%C3%A9mineur`

2 Noyau fonctionnel

Pour commencer, implémentez une classe `MatriceBomb` qui prend une hauteur et une largeur et un nombre de bombe et qui crée aléatoirement (en utilisant `new Random().nextInt(int a)` de `java.util.Random`) en interne une matrice (tableau de tableau) de `boolean` correspondant aux positions des bombes (`true` veut dire que la bombe est présente à cet endroit). Ajouter une méthode `getVoisinage(int i, int j)` qui calcul pour une position (i, j), le nombre de bombe dans son voisinage (on compte les cases en diagonales aussi).

Ensuite, implémentez une deuxième classe `MatriceValeur` qui hérite de `MatriceBomb` et qui construit en plus en interne la matrice du nombre de bombes dans chaque voisinage (pour chaque position (i, i), on veut le nombre de bombe dans le voisinage de (i, j) ou -1 si c'est une bombe).

Enfin, construisez une troisième classe `MatriceSecret` qui hérite de `MatriceValeur` et qui construit en plus en interne la matrice des cases que l'on a déjà dévoilées (matrice de `boolean`). Ajouter une méthode `testCase(int i, int j)` qui correspond au choix du joueur de la case (i, j). Si la case est une bombe, la méthode renvoie l'exception `DefaiteException`. Si la case n'est pas une bombe et que c'était la dernière case qui n'était pas une bombe qui n'était pas encore découverte, la méthode renvoie l'exception `VictoireException`.

3 Interface graphique

Dans cette partie, vous allez avoir besoin de deux classes `Main.java` pour activer l'application et `TableSecretGui` qui héritera de `JFrame`.
Indices :

- Utilisez une matrice de `JButton` pour les cases que vous mettrez dans un `JPanel` en utilisant `GridLayout`
- Ajouter aussi un `JLabel` qui permettra d'afficher la victoire ou la défaite que vous mettrez dans un autre `JPanel`.
- Associer vos deux `JPanel` à votre classe (qui hérite de `JFrame`) en utilisant `BorderLayout` (par défaut).
- Vous pouvez utiliser `.setPreferredSize(new Dimension(longueur,largeur))` pour ajouter une préférence sur la longueur et la largeur d'un `JPanel`.

4 Un peu de récursivité

Pour aider un peu le joueur de votre démineur, on se rend compte que si on appuie sur une case voisine d'aucune bombe, on peut appuyer sans risque sur toutes les cases voisines. Améliorer votre méthode `testCase(int i, int j)` pour que si on appuie sur une case voisine de zéro bombe, on teste récursivement toutes les cases du voisinage.

5 Améliorer le design

D1 Ajouter une couleur différente pour chaque case avec un numéro différent. (Exemple : tous les 0 sont blanc, tous les 1 sont bleu, ...)

D2 Afficher le nombre de bombe

D3 Ajouter au lancement du jeu, une nouvelle fenêtre où on choisit la taille de la grille et le nombre de bombe et un bouton pour créer la grille.

6 *(Pour le fun)* Implémentation d'autres variantes

Il existe d'autres options que l'on peut implémenter pour créer des variantes. Implémentez, une, plusieurs ou toutes ces variantes :

V1 La première case ne peut pas être une bombe

V2 Utiliser un système de drapeau pour marquer avec le clic droit la pause d'un drapeau sur un endroit où vous pensez qu'il y a une mine. Vous ne pouvez plus cliquer sur la case tant qu'il y a le drapeau. Refaite un clic droit pour enlever le drapeau. Ajouter un affichage du nombre de drapeaux mis.

Vous pourrez utiliser le code suivant pour détecter les clics droits

```
.addMouseListener(new MouseAdapter(){
public void mouseClicked(MouseEvent e){
if (e.getButton() == MouseEvent.BUTTON3) { JButton o = (JButton)
e.getSource(); } }
```

V3 Pouvoir mettre plusieurs bombes par case : `https://fr.wikipedia.org/wiki/D%C3%A9mineur_(genre_de_jeu_vid%C3%A9o)#/media/Fichier:Firefox_Multiple_mines.png`

V4 Certaines cases sont au départ grisées (ne contiennent pas de bombe mais on ne peut pas avoir d'information sur le nombre de bombes dans le voisinage) `https://upload.wikimedia.org/wikipedia/commons/b/ba/Minesweeper_games2relaxnet.png`

V5 Calculer le *Bechtel's Board Benchmark Value* (3BV) d'une grille (`http://www.minesweeper.info/wiki/3BV`) et afficher le sur la grille