ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

ÉPREUVE REGROUPÉE

Janvier 2020

BRANCHE : **ALGORITHMIQUE ET PROGRAMMATION OBJET (APO)**

CLASSE : **ESIG2 Classe A (Groupes 1, 2 et 3)**

DATE : 15 janvier 2020

NOM : .....

PRÉNOM : .....

PROFESSEUR : Eric Batard

N° du poste de travail : ESIG-PB.....

N° de clé USB : .....

Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
  Documentation électronique (disquettes, CD, clé USB, …) : Autorisée si elle a été recopiée dans un répertoire sur `C:\ESIGUsers` ***avant*** le début de l'épreuve.
- Tout partage de ressources de votre poste de travail avec le réseau ainsi que toute tentative de communication seront considérés comme fraude et sanctionnés comme tels par la note minimale.

Démarrage

- Connectez-vous au réseau sur le poste de travail qui vous a été attribué.
- Copiez dans `C:\ESIGUsers\APO-JAN20` le *contenu* du dossier réseau qui vous sera indiqué au début de l'épreuve, normalement
  `G:\ESIG Distribution\2019_2020\ESIG-2\APO\Eléments ER APO 15-01-20`
- Les éléments fournis comprennent cet énoncé et un répertoire `BinApo` qui est un ***répertoire de projet IntelliJ IDEA***. Vous rendrez le répertoire dans lequel vous avez travaillé.

Travail à faire

- Lisez tous les documents fournis.
- Complétez les procédures/fonctions/méthodes ou classes demandées de manière à ce qu'elles répondent aux spécifications de l'énoncé.
- Vous rendrez le répertoire du projet avec les fichiers modifiés.

C'est *à vous* de vérifier que le répertoire rendu contient bien
la dernière version de votre travail

Le nom du *répertoire de projet* IntelliJ IDEa doit être préfixé par vos initiales

---

A ajouter au plus tard juste avant la reddition

Quand les deux cas se présentent, le masculin est utilisé dans cet énoncé de façon générique.

Le binaire dans l'APO

Contexte

Nous allons jouer avec un jeu appelé, entre autres, Binero, Binairo ou Takazu (cf. https://fr.wikipedia.org/wiki/Takuzu).

Il n'est pas nécessaire de connaître, savoir ou aimer jouer à ce jeu pour réaliser l'épreuve. Ses règles sont extrêmement simples : il faut remplir une grille carrée avec des zéros et des uns (ou des ronds et des croix, ou une autre paire de symboles). La contrainte principale[1] est qu'il ne doit jamais y avoir 3 symboles consécutifs identiques, ce que ce soit en ligne ou en colonne (les diagonales ne comptent pas).

L'objectif de cette épreuve est de produire une interface permettant certains traitements sur une grille de Binairo de taille[2] 8x8.

Mise en place de l'interface de départ

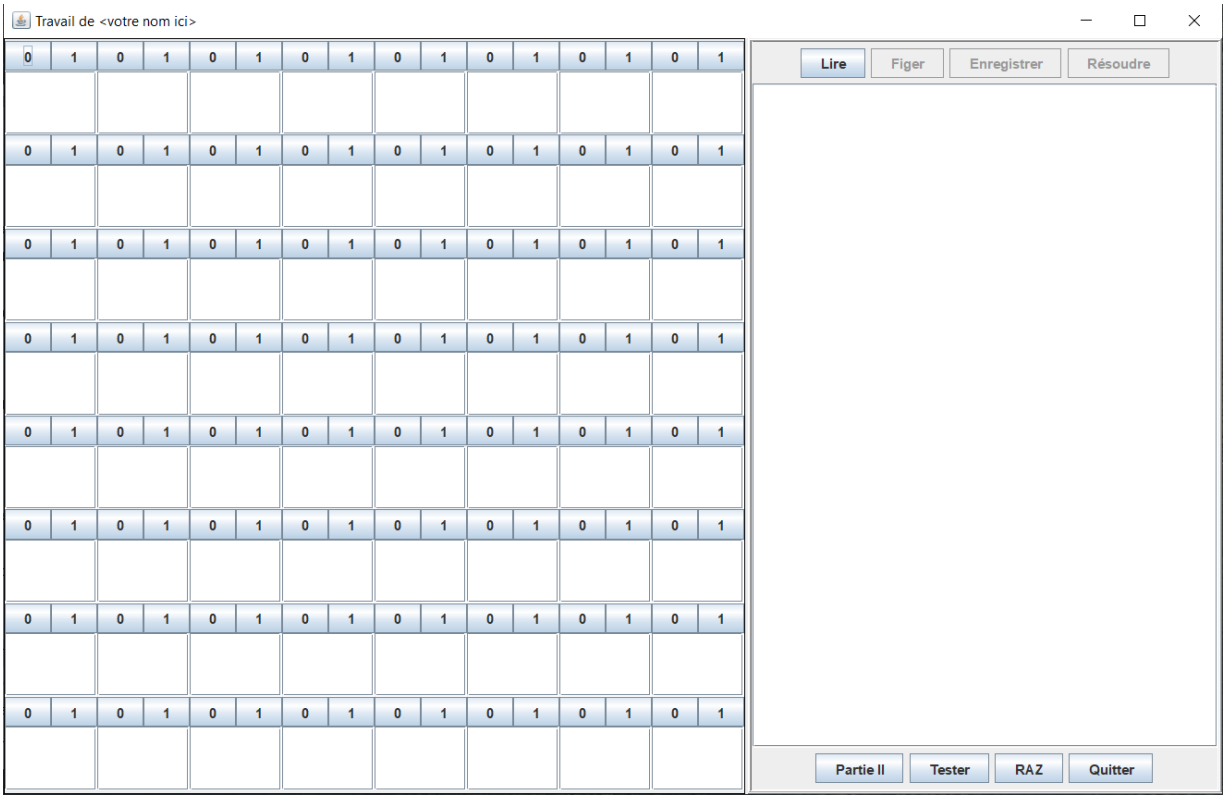

Figure 1

Cette interface peut impressionner mais elle met en œuvre des éléments non triviaux qui sont fournis, donc qui vont s'utiliser comme des composants déjà connus.

Le projet IntelliJ IDEA s'appelle `BinApo` et contient déjà deux classes à compléter `BinApoMain` (la classe exécutable) et `BinApoFen` (qui correspondra à la fenêtre ci-dessus). Vous utiliserez aussi, ***sans les modifier***, `BinApoPanel` ainsi que (indirectement) `BinApoTextField`.

Vous avez également à disposition dans `BinApoFen` deux méthodes : `afficher` pour afficher une chaîne (avec passage à la ligne) dans un `JTextArea` et `findExtension` pour récupérer l'extension d'un fichier (sans le point et en minuscule).

En premier lieu, vous indiquerez votre nom dans le titre à la place de `<votre nom ici>`.
Vous veillerez aussi que la case de fermeture ferme proprement votre programme.

[1] Il y a aussi la règle que deux lignes ou deux colonnes ne peuvent être identiques. Mais elle ne nous servira pas ici.
[2] La taille est fixe ici, pour simplifier. On trouve des tailles de grille allant 6x6 à 14x14.

Structure de la fenêtre

Dans cette fenêtre, il y a plusieurs zones à considérer. La gestion d'événements est détaillée dans la section suivante.

Clairement on a une partie gauche et une partie droite qui sont côte à côte. Commençons par la partie droite. Elle se compose de trois zones : deux zones de bouton et un zone centrale qui n'est autre qu'une zone de texte (`JTextArea`).

— *Zone de boutons du haut :*
  — Un bouton « Lire » toujours actif
  — 3 boutons « Figer », « Enregistrer », « Résoudre » désactivés au départ
— *Zone centrale appelée dans la suite le* **log** :
  — Un `JTextArea` (de 20 sur 40) non modifiable et disposant de barres de défilement verticale et horizontale (qui n'apparaissent que quand c'est nécessaire).
— *Zone de boutons du bas :*
  — 4 boutons, « Partie II », « Tester », « RAZ », « Quitter », toujours actifs.

La partie de gauche n'est pas si complexe quand on considère la classe `BinApoPanel` qui correspond aux affichages suivants (donnés en exemple) :

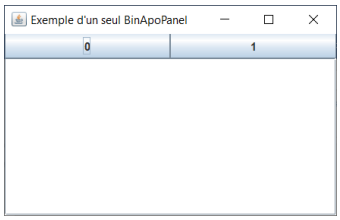

Figure 2a – un `BinApoPanel` *tout seul*

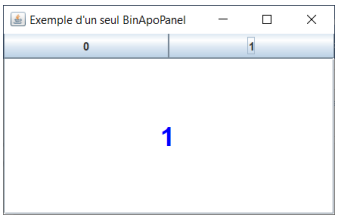

Figure 2b – le même après clic sur le bouton « 1 »

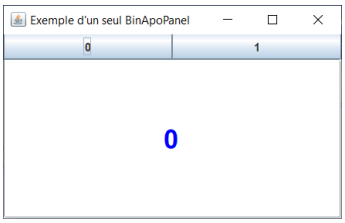

Figure 2c – le même après clic sur le bouton « 0 »

Cette classe rassemble dans un même `JPanel` deux boutons et un `BinApoTextField` qui n'est autre qu'un `JTextField` amélioré. Cela correspond à une case de la grille de Binairo. Le mode de fonctionnement est le suivant :

— normalement la case est créée vide (on verra comment plus loin).
— si on clique sur 0 ou 1, cette valeur apparaît.
— si la case contient déjà une valeur (0 ou 1) et qu'on clique sur l'autre (1 ou 0), c'est l'autre valeur qui apparaît.
— si la case contient déjà une valeur (0 ou 1) et qu'on clique sur la même valeur (0 ou 1), la case est vidée.

Notez bien que ce fonctionnement est ***déjà*** programmé : vous n'avez qu'à poser les `BinApoPanel` pour en bénéficier.

La partie de gauche n'est donc une seule grande zone avec 8 x 8 = 64 `BinApoPanel`.

— *Zone des cases d'une grille :*

Cette zone est manifestement organisée en une grille.

Pour la remplir, vous utiliserez le tableau appelé `grille`, déclaré et dimensionné dans la classe `BinApoFen`. Il s'agit d'un tableau regroupant les 64 `BinApoPanel` à déposer sur cette zone.

Pour instancier chaque `BinApoPanel` (ce qui est obligatoire pour les déposer), il faut fournir au constructeur 3 paramètres :

1) la valeur figurant dans la case. Comme les cases sont créées vides, il faut donner -1.
2) la ligne où se trouvera le `BinApoPanel`, c'est-à-dire la case qu'il représente.
3) la colonne où se trouvera le `BinApoPanel`, c'est-à-dire la case qu'il représente.

Le plus simple sera donc d'utiliser une boucle imbriquée classique (sur les lignes et les colonnes de chaque ligne). (Cette structure de boucle imbriquée servira à plusieurs reprises.)

Cinématique de l'interface

Clic sur « Lire »

Comme son nom l'indique, un clic sur « Lire » provoque l'ouverture d'une boîte de choix de fichier (`JFileChooser`).

Si l'utilisateur annule sans rien choisir, il ne se passe rien.

Dès que l'utilisateur a choisi un fichier,

— la grille est remise à zéro (cf. plus loin les explications du bouton « RAZ »).

— le nom du fichier choisi est affiché dans le log, comme ceci (police non significative) :

Fichier choisi :
I:\ESIG\APO\BinApo\GrilleRemplie.txt

— les boutons « Figer », « Enregistrer » et « Résoudre » deviennent actifs.

— le fichier texte est lu et est affiché dans la grille.

Pour ce dernier point, trois grilles sont fournies : `GrilleRemplie.txt`, `GrillePresqueRemplie.txt` et `GrilleARemplir.txt`. Ces fichiers contiennent exactement 64 nombres valant 0, 1 ou -1. Il suffit de donner cette valeur à chacun des éléments du tableau `grille` au moyen de la méthode (fournie bien sûr) `setValeur()`.

Ci-dessous le résultat de la lecture de la grille presque remplie.

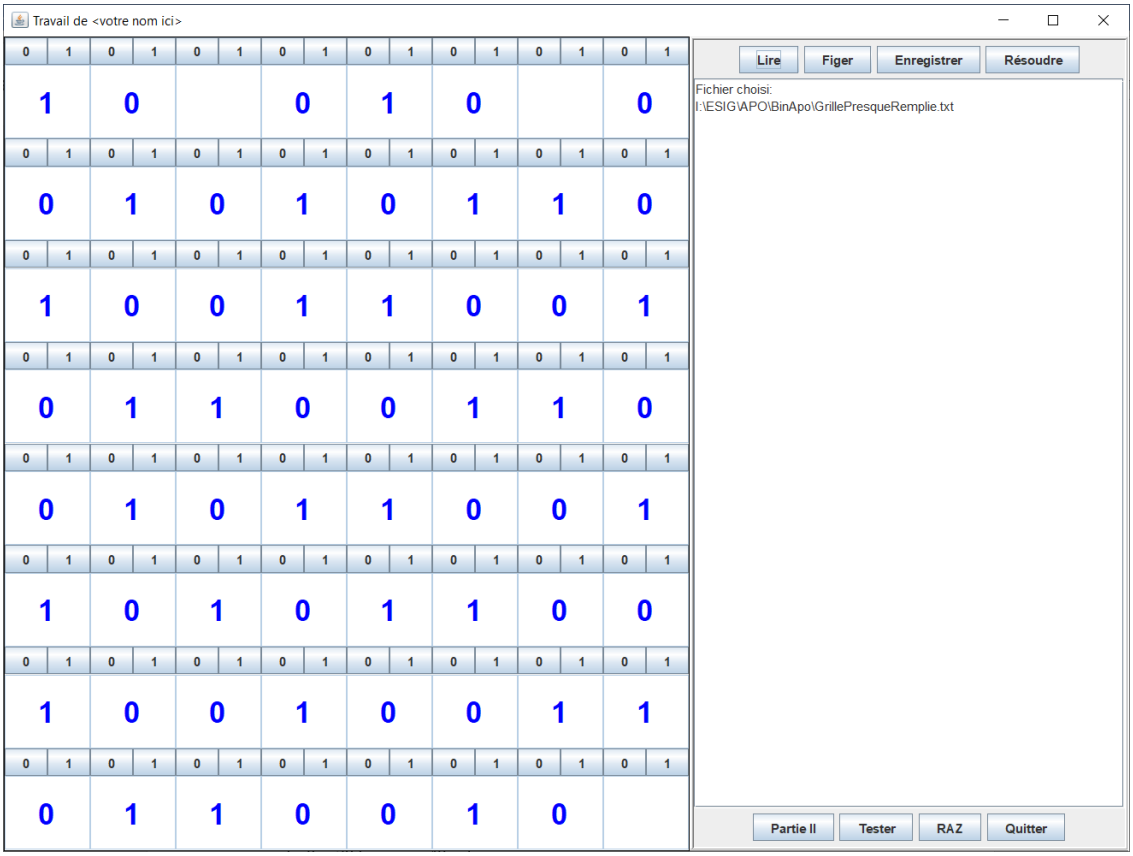

Figure 3

Clic sur « Enregistrer »

Comme nous n'avons pas vu l'enregistrement de fichier texte, c'est simplement :

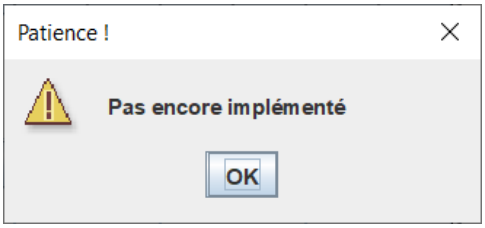

Figure 4

Clic sur « Résoudre »

La résolution complète d'une grille partiellement remplie est un peu technique et donc hors champ de cette épreuve. Mais il est déjà possible de regarder si la grille est complètement remplie ou pas. Autrement dit, chercher dans le tableau `grille` s'il y a encore une case dont la valeur est -1. Cela grâce à la méthode `getValeur()` (fournie) de la classe `BinApoPanel`.

Vous afficherez dans le log l'un ou l'autre de ces messages (police non significative) :

Ce que je peux dire pour l'instant, c'est que : Il reste de cases vides. La grille n'est pas remplie !	Ce que je peux dire pour l'instant, c'est que : La grille est remplie ! Le jeu est terminé.

Clic sur « Figer »

L'idée est, une fois une grille partiellement remplie, d'empêcher l'utilisateur de modifier les cases déjà remplies.

La réalisation est simple : il suffit de désactiver toutes les cases déjà remplies, c'est-à-dire dont la valeur est 0 ou 1, ou autrement celles dont la valeur n'est pas -1. Pour cela utilisez les méthodes `getValeur()` et `setEnabled()` (fournies) de la classe `BinApoPanel`.

Il faudra aussi indiquer que « La grille est figée » dans le log, comme illustré ci-dessous.

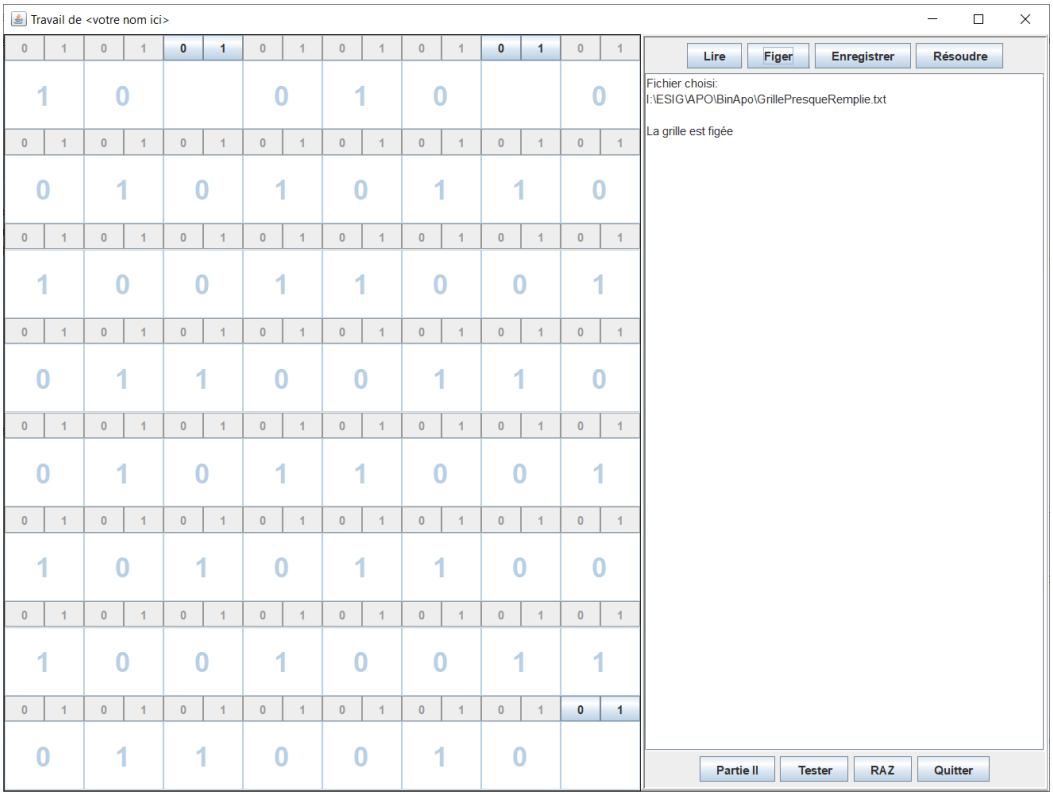

Figure 5 – 3 cases non figées car vides

Clic sur « RAZ »

L'objectif est de remettre à zéro :

— la grille. Pour cela, il faut utiliser la méthode `reset()` (fournie) de la classe `BinApoPanel` sur chacune des cases du tableau `grille`.
— désactiver les boutons « Figer », « Enregistrer » et « Résoudre ».
— effacer le texte du log.

Clic sur « Tester »

Il est apparu un peu compliqué de tester dans la grille si une saisie respectait la règle de base du Binairo : pas 3 symboles identiques de suite.

En revanche si on travaille sur une chaîne de caractères, c'est beaucoup plus facile.

Après un clic sur le bouton « Tester », la boîte de dialogue suivante, assez explicite, apparaît :

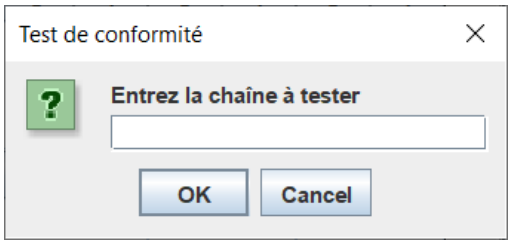

Figure 6

Si l'utilisateur annule sans rien taper, ou si la chaîne n'a pas une longueur d'exactement 3 caractères, ou si un (ou plus) caractère n'est ni « 0 » ni « 1 », c'est une erreur de saisie comme indiquée par ce message :

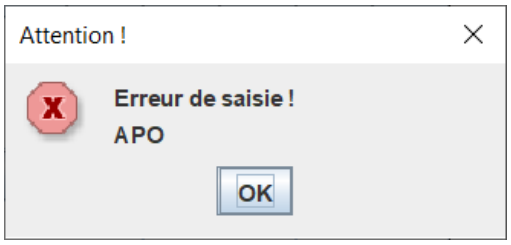

Figure 7a – un exemple d'erreur de saisie

En revanche, si la chaîne respecte ces contraintes mais contient trois fois « 0 » ou trois fois « 1 », c'est une erreur de syntaxe.

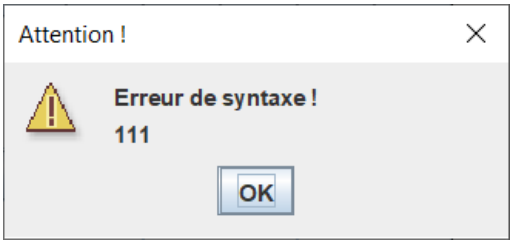

Figure 7b – un exemple d'erreur de syntaxe

Enfin, si la chaîne respecte toutes les contraintes :

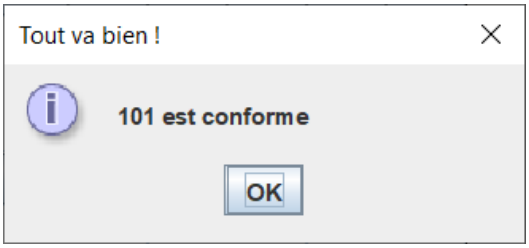

Figure 7c – un exemple de chaîne conforme

Clic sur « Quitter »

L'application se ferme, tout aussi proprement qu'avec la case de fermeture

Clic sur « Partie II »

Cette Partie II correspond aux exercices de parcours récursif des répertoires. Elle peut être réalisée totalement indépendamment de l'interface graphique en affichant les résultats par `System.out.println` (sans pénalité). Cependant les exemples de sortie ci-dessous correspondent à des affichages dans le log à la suite d'un clic sur « Partie II ».

Vous travaillerez sur l'arborescence de répertoire ci-dessous (qui se trouve dans le répertoire de projet `BinApo`). Une constante `REP_DEPART` est également définie dans la classe `BinApoFen`. Notez bien que contrairement aux noms utilisés, cette arborescence ne contient ***aucune*** grille de Binairo (ou d'autre chose d'ailleurs) car il n'y a que des fichiers vides (0 octet !).

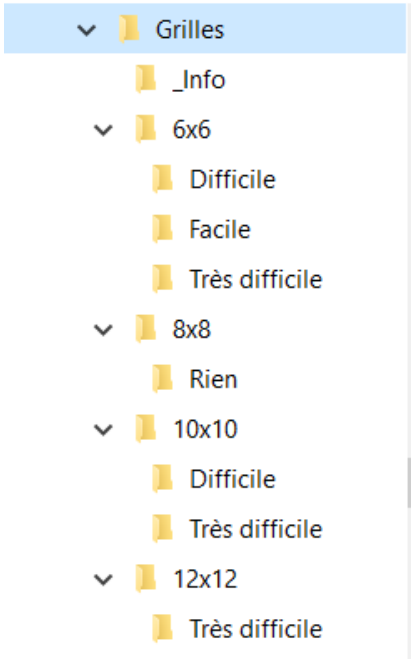

Figure 8

L'objectif est triple :

— Il faut compter le nombre de grilles présentes dans cette arborescence. Une grille est définie par un nom de fichier qui commence par « Grille » (cf. méthode `StartsWith` de la classe `String`) et une extension « txt » (cf. méthode `findExtension` fournie dans `BinApoFen`).

— Il faut compter combien il y a de répertoires totalement vides. Un répertoire totalement vide est un répertoire qui ne contient ni fichier, ni sous-répertoire.

— Il faut lister les répertoires totalement vides rencontrés. L'ordre peut être quelconque.

Vous pouvez traiter les deux derniers points séparément mais il serait plus simple de construire la liste et de donner sa taille pour faire d'une pierre deux coups, ou, ici, faire d'un parcours récursif deux réponses !

Ci-dessous les résultats attendus :

```
Nombre de grilles : 16
Nombre de répertoires totalement vides : 3
  - Grilles\6x6\Très difficile
  - Grilles\8x8\Rien
  - Grilles\_Info
```

Rappel sur les tableaux à deux dimensions (peut-être inutile !)

Le tableau `grille` est déjà déclaré. Vous n'aurez donc besoin que de :

— accéder à un élément du tableau : la syntaxe est `grille[numLigne][numColonne]`
Notez la double paire de crochets (pas de virgule !).
N'oubliez pas que ce tableau contient des éléments qui sont, chacun, des instances de `BinApoPanel`.

— parcourir tous les éléments du tableau : une bonne double boucle imbriquée avec une boucle sur les lignes et l'autre sur les colonnes fera l'affaire ! Utilisez la constante pour les bornes.