TP C#11 : Suji wa dokushin ni kagiru
数字は独身に限る

1 Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante : Tout manquement à cette règle sera sanctionné.

```
rendu-tp-sudok_u.zip
+-- sudok_u/
    |-- AUTHORS*
    |-- README
    +-- Sudoku
        |-- Sudoku.sln*
        +-- Sudoku*
            |-- Stuff.cs*
            |-- Program.cs*
            |-- Sudoku.cs*
            |-- IO.cs*
            +-- Tout sauf bin/ et obj/
```

Bien entendu, vous devez remplacer "`sudok_u`" par votre propre login. Les fichiers annotés d'un astérique sont *OBLIGATOIRES*.

N'oubliez pas de vérifier les points suivants avant de rendre :

— Le fichier `AUTHORS` doit être au format habituel (*␣`sudok_u` suivi d'un retour à la ligne, dans le cas ou votre login est `sudok_u`).
— Le fichier `README` (sans extension) doit contenir toutes les informations dont vous voulez nous faire part. Tout bonus non spécifié dans le README ne sera pas pris en compte.
— Pas de dossiers `bin/` ou `obj/` dans le projet.
— **Le code doit compiler, autrement votre note sera multipliée par 0 !**

Table des matières

2 Introduction

Lisez attentivement le sujet avant de commencer tête baissée.
Toutes les fonctions vous sont interdites sauf indication contraire du sujet ou des assistants. N'oubliez pas aussi de gérer tous les cas d'erreur sauf si c'est indiqué.

2.1 Suji wa doku...quoi ?

> « *Le nom* ***sudoku*** *est né de l'abréviation de la règle du jeu japonaise "Suji wa dokushin ni kagiru" (数字は独身に限る), signifiant littéralement "Chiffre limité à un seul" (sous entendu par case et par ligne). Cette abréviation associe les caractères Su (chiffre) et Doku (unique).* »
>
> — Wikipédia

Le sudoku est un jeu en forme de grille carrée composée de neuf cases de côté. Chaque sous-grille de trois cases de côté est appelée « région ». Les quatre règles du Sudoku sont les suivantes :

- Il ne peut y avoir qu'un seul chiffre par case
- Un chiffre ne peut être présent qu'une seule fois par ligne
- Un chiffre ne peut être présent qu'une seule fois par colonne
- Un chiffre ne peut être présent qu'une seule fois par région

Voici un exemple de grille de sudoku que vous pouvez retrouver facilement dans votre quotidien préféré.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

2.2 Objectifs

Cette semaine, le TP a pour objectif de vous faire coder un sudoku, de la génération à la résolution de grille (sympa pour votre grand-mère, vous allez pouvoir lui montrer que l'informatique c'est pratique). Vous allez donc utiliser ce que vous savez dans la manipulation de tableaux, de listes et de fichiers pour en venir à bout. Mais avant de commencer, vous allez devoir coder quelques petites fonctions.

3 Avant de commencer...

Avant de vous lancer dans le coeur du TP de cette semaine, voici pour vous quelques courts exercices de mise en bouche. Il vous est demandé de les mettre dans le fichier *Stuff.cs*.

3.1 TimeAfterTime

```
public static bool TimeAfterTime(ref int days, ref int hours,
                                 ref int mins, ref int sec);
```

Objectif

Cette fonction prend en référence des paramètres correspondants à une durée de temps et va les réorganiser pour qu'ils soient dans les bons intervalles. Par exemple, si vous lui donnez 1664 secondes, vous obtiendrez 27 minutes et 44 secondes.

Cette fonction retournera `true` si la conversion a pu être effectuée. Sinon, `false`.

A ne pas oublier...

Vous devez gérer les valeurs négatives. Par exemple, avec -10 minutes et 2403 secondes, vous aurez 30 minutes et 3 secondes. Si le résultat possède encore des nombres négatifs, la valeur de retour sera `false`.

3.2 Codage par répertoire dynamique

```
public static string Compression(string source);
public static string Decompression(string source);
```

Le codage par répertoire dynamique est une approche répandue de compression de texte qui consiste à remplacer chacun des mots de ce texte par un nombre qui représente sa position dans le répertoire. Un tel codage est dit statique si le répertoire de mots est connu à l'avance. Le principal inconvénient de cette méthode réside dans le fait que ce même répertoire doit aussi être connu pour décompresser le texte.

D'autre part, un codage par répertoire dynamique contourne ce problème en dérivant le contenu du répertoire de mots à partir du texte qui doit être compressé. Au début du procédé, le répertoire est vide. En considérant le texte à partir du début, lorsqu'un mot figure dans le répertoire il est remplacé par le numéro de sa position. S'il n'est pas présent, il est ajouté à la fin et est laissé tel quel dans le texte.

Objectif

Vous allez devoir premièrement appliquer cette compression à la chaîne de caractères passée en entrée et la retourner compressée.

Le paramètre d'entrée ne contiendra que des caractères miniscules non accentués, des espaces et des retours à la ligne. Les espaces et les retours à la lignes ne seront pas convertis (donc doivent être conservés).

La fonction de décompression fera le procédé inverse afin de retrouver la chaîne de départ.

Exemple

Un exemple avec un magnifique haïku :

```
string source = "le grand chien bleu hurle
                 le ciel hurle avec le chien
                 grand bleu orageux";

string comp = Compression(source);

Console.WriteLine(comp);
/* prints "le grand chien bleu hurle
           1 ciel 5 avec 1 3
           2 4 orageux" */

Console.WriteLine(source == Decompression(comp));
/* prints True */
```

Hint

> Comment récupérer les mots chacun après les autres ? Peut-être qu'un tour sur MSDN sur les méthodes de la classe string sera une balade agréable.

4 Sudoku

4.1 Sudoku.cs

Nous allons commencer doucement avec des méthodes simples de manipulation de tableaux.

Objectif

Vous devez créer une classe `Sudoku` et définir ses attributs. Si vous avez bien lu, un sudoku est simplement composé d'une grille de 9x9 cases, contenant des chiffres de 0 à 9 (le 0 représentera une case vide). Nous vous conseillons aussi un attribut de type `Random`, qui va servir pour une des méthodes. A vous de leur choisir le bon niveau d'accessibilité.

Avant d'écrire le contenu du constructeur, vous allez devoir définir les méthodes suivantes :

Init

```
public void Init(int init);
```

Remplit simplement toutes les cases de la grille avec l'entier `init`.

Print

```
public void Print();
```

Affiche simplement dans la console une grille de sudoku de cette manière :

```
+-----------------------+
| 4 9 7 | 8 3 6 | 2 1 5 |
| 1 3 5 | 4 2 7 | 6 9 8 |
| 2 8 6 | 9 5 1 | 3 7 4 |
+-------+-------+-------+
| 9 6 8 | 5 4 2 | 7 3 1 |
| 3 7 4 | 6 1 9 | 5 8 2 |
| 5 2 1 | 3 7 8 | 4 6 9 |
+-------+-------+-------+
| 8 5 3 | 1 6 4 | 9 2 0 |
| 7 4 9 | 2 8 3 | 1 5 6 |
| 6 1 2 | 7 9 5 | 8 4 3 |
+-----------------------+
```

RandomlyFill

```
private void RandomlyFill(int nb);
```

Cette méthode attribue à la grille un nombre `nb` de cases aléatoires supplémentaires. Les cases ont des valeurs aléatoires mais respectent aussi les règles du Sudoku. Votre code sera donc une boucle exécutée `nb` fois avec à chaque fois pour une case aléatoirement sélectionnée :

- on vérifie si elle n'est pas déjà remplie
- on vérifie la colonne de celle-ci pour qu'il n'y ait pas le même chiffre
- on vérifie la ligne de celle-ci pour qu'il n'y ait pas le même chiffre
- on vérifie la région (carré) de celle-ci pour qu'il n'y ait pas le même chiffre
- si toutes ces conditions sont remplies on peut la remplir avec le chiffre aléatoire sélectionné et on décrémente `nb` sinon on refait la boucle sans le décrémenter

Il vous est possible de coder des fonctions annexes. N'oubliez pas de les commenter !

Hint

> N'utilisez cette fonction qu'avec un petit `nb` pour que la grille reste correcte et se remplisse jusqu'au bout ($nb < 60$).

Constructeur

Le constructeur va maintenant simplement initialiser les attributs et remplir aléatoirement la grille de sudoku avec un entier demandé à l'utilisateur.

Bonus fortement recommandé

Vous l'avez remarqué, l'algorithme de remplissage donné ci-dessus n'est pas de toute beauté. En bonus, vous pouvez coder ce remplissage aléatoire d'une meilleure manière, n'oubliez pas les explications dans le *README*.

Hint

> Avez-vous bien lu **tout** le sujet ?

4.2 IO.cs

`IO` sera une classe d'intéraction avec les fichiers, qui va nous permettre de récupérer leur contenu.

Objectif

Maintenant que vous pouvez afficher des grilles toutes belles, jetons un oeil à la classe `IO`. Elle va vous permettre de récupérer le contenu d'un fichier que l'utilisateur demandera afin de construire une grille de sudoku avec et la possibilité de sauvegarder une grille dans un fichier.

```
1 private static bool FileToTab(string filename, int[,] tab)
```

Cette fonction ouvre le fichier `filename` et met dans le tableau `tab` les différents chiffres qu'il contient. Il retire des espaces, tabulations et retours ligne. Gère toutes les erreurs. Il retourne `true` si le tableau a correctement été rempli, `false` sinon.

```
1 public static void LoadFile(int[,] tab)
```

Cette fonction appelle la fonction `FileToTab` avec un nom de fichier demandé à l'utilisateur tant que le tableau n'est pas correctement rempli.

```
1 public static bool SaveFile(int[,] tab)
```

Cette fonction demande un nom de fichier à l'utilisateur et essaie de l'ouvrir. Si le fichier existe déjà, demander si on veut réecrire par dessus. Pour tout autre erreur de fichier, retourner faux. Si le fichier est correctement ouvert, écrire chaque ligne du sudoku dans le fichier.

4.3 Retour sur Sudoku.cs

Dernière étape, la plus longue, la résolution de grille de sudoku ! En plus d'utiliser à nouveau vos connaissances sur les tableaux, vous allez pouvoir nous épater avec celles sur les listes. Besoin d'un rappel ? MSDN est votre ami, chérissez-le.

Mais tout d'abord, une petite partie de cours à propos des paramètres optionnels et nommés.

Cours

Les fonctions en C# peuvent utiliser des paramètres dits **optionnels**, c'est-à-dire que si on décide de les appeler sans ce paramètre, un leur est donné par défaut. Les paramètres facultatifs doivent être mis à la fin. Un exemple pour que cela soit clair :

```
1  public static void HelloToSomeone(string someone = "World", int nb = 1)
   {
3      for (int i = 0; i < nb; i++)
           Console.WriteLine("Hello " + someone + "!");
5  }

7  /* In your Main */
   HelloToSomeone("Alex", 2);  // prints "Hello Alex!" 2 times
9  HelloToSomeone();           // prints "Hello World!"
   /* HelloToSomeone(42); */   // ERROR
```

Pour palier la dernière erreur, il est possible de **nommer** explicitement le paramètre. Le nommage explicite peut aussi être utilisé pour des paramètres non facultatifs.

```
HelloToSomeone(nb : 42);    // prints "Hello World!" 42 times
```

Nouveau constructeur

Le constructeur de votre sudoku va maintenant prendre en compte la possibilité que l'utilisateur préfère utiliser une grille de son propre cru. Ce constructeur va avoir un seul paramètre `getFromUser`, un booléen, qui sera optionnel et dont la valeur par défaut sera `false`.

Exemple

Ces deux utilisations doivent fonctionner.

```
1 Sudoku defaultSudoku = new Sudoku();
  /* constructs a new sudoku ramdomly filled */
3
  Sudoku fromUser = new Sudoku(true);
5 /* constructs a new sudoku after asking the user a file to load */
```

Hint

Vous venez de coder des fonctions dans la classe *IO.cs*...

Save

```
1 public bool Save();
```

Cette méthode demande à l'utilisateur s'il veut sauvegarder la grille résolue. Tant que la fonction de sauvegarde que vous avez codé dans IO renvoie false, redemander à l'utilisateur s'il veut sauvegarder sa grille.

Petit rappel sur les listes

Lorsque vous initialisez une liste, il est possible d'indiquer directement des éléments à ajouter, comme pour un tableau, grâce à des accolades.

```
1 List<string> namae = new List<string> { "Arekkusu", "Rec9", "Poni" };
  /* the list namae contains now "Arekkusu", "Rec9" and "Poni" */
```

GetPossibleNumbers

Voici trois méthodes que vous devez implémenter pour la classe `Sudoku` :

```
  void GetLinePossibleNumbers(ref List<int> numList, int line);
2 void GetColumnPossibleNumbers(ref List<int> numList, int column);
  void GetRegionPossibleNumbers(ref List<int> numList, int line,
4                                                      int column);
```

Chacune de ces fonctions va trouver tous les chiffres possibles pour les cases vides de la grille de sudoku parmi les chiffres contenus dans `numList`, respectivement par ligne, colonne et région. Vous devez donc procéder à la suppression de chiffres présents dans la liste `numList` donnée en paramètre.

Attention pour `GetRegionPossibleNumbers`, les paramètres `line` et `column` correspondent à la position d'une case, il faut donc chercher parmi les cases de la région à laquelle elle appartient.

GetMinList

```
void GetMinList(ref List<int> minList, ref int xIndex, ref int yIndex);
```

Cette fonction va trouver la case vide ayant le moins de chiffres possibles pour la remplir. Elle retourne ensuite par référence les index de x et y de la case de la grille, puis la liste de ses possibilités.

Hint

Des pistes :
- Création et initialisation adaptée de la liste
- On teste chaque case pour trouver la plus petite liste

Les fonctions codées précédemment aimeraient peut-être qu'on les utilise...

IsFinished

Un peu plus facile ?

```
1 public bool IsFinished();
```

Cette fonction va parcourir toute la grille et vérifier si le tableau est correctement rempli en respectant les règles du sudoku. Elle renvoie bien sûr ce résultat.

Hint

Souvenez vous que les fonctions d'avant sont très serviables.
Pensez aux conditions pour lesquelles la grille est correctement établie.

Solve

Et voici ce que vous attendiez tous... le Solveur !

```
1 public bool Solve();
```

Bon vous aurez sans doute compris que cette fonction va résoudre notre grille si elle est correcte. Comme nous sommes sympathiques, voici un micro-algorithme :
- Retourner vrai si la grille est finie.
- Sauvegarder l'état actuel de la grille.
- Tant que des cases n'ont qu'une seule possibilité, les remplir avec celle-ci.
- Pour tous les chiffres possibles de la case qui en a le moins : remplir cette case avec l'un d'entre eux, utiliser la récursivité, retourner le résultat si vrai, sinon continuer avec le chiffre suivant et ainsi de suite.
- Tester à nouveau si la grille est finie au cas où on ne passe pas par l'étape précédente. On retourne `true` si c'est vrai sinon on rétablit la grille sauvegardée et on retourne `false`.

Essayez d'utiliser cette fonction sur la grille vide pour générer une grille générique. Autre truc à tester, la fonction sur une grille remplie par la fonction `RandomlyFill` (avec comme paramètre quelque chose entre 0 ou 20) : vous aurez le droit à votre belle grille aléatoire ! Sachez que la grille que génère `RandomlyFill` est correcte mais n'a pas forcément de solution.

Si vous avez besoin d'aide sur les fonctions de cet exercice, n'hésitez pas à demander à vos ACDC.

4.4 Program.cs

Il vous reste juste un dernier effort à faire : vous avez en main une classe `Sudoku` qui marche comme sur des roulettes (oui, on peut marcher avec des roulettes), il vous faut maintenant remplir le `Main` de votre programme afin que l'utilisateur puisse utiliser tranquillement votre solveur.

Objectif

Voilà ce qui est demander au minimum : Remplir votre `Main` avec ce qu'il faut pour que l'utilisateur puisse choisir de charger une grille depuis un fichier ou en générer un aléatoirement, résoudre la grille et sauvegarder la grille ainsi résolue.

Bonus

Impressionez-nous ! Des jolies couleurs, une interface de jeu pour le sudoku...

The bullshit is strong with this one...