TP C#3 : Boucles et débugguer

1 Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive de rendu respectant l'architecture suivante :

```
tpcs03-login_x.zip
    |– tpcs03-login_x/
        |– AUTHORS
        |– README
        |– src/
            |– boucles/
            |– sudoku/
            |– debugger/
```

Vous devez, bien entendu, remplacer login_x par votre propre login. Avant de rendre, n'oubliez pas de vérifier :

- Que le fichier AUTHORS est bien au format habituel (une *, un espace, votre login et un retour à la ligne, dans un fichier sans extension)
- Que vous avez bien supprimé les dossiers bin et obj
- **Que votre code compile**

2 Bouuuuucles !

2.1 La boucle while

2.1.1 Qu'est ce que c'est ?

Comme son nom l'indique, la boucle ***while*** exécute des instructions tant qu'une condition est vraie :

```
while(condition)
{
    instruction1;
    instruction2;
}
```

La boucle while :

```
int i = 0;
while(i != 10)
{
    Console.Writeline("Hello World !");
    i = i + 1;
}
```

Par exemple cette boucle va afficher ***10 Hello world*** sur la console.

Console.Writeline() et ***Console.Readline()*** sont deux fonctions qui permettent à l'utilisateur d'interagir avec le programme. La première va écrire sur la console et la deuxième lire sur celle-ci

```
Console.Writeline("Enter your name :");
string name = Console.Readline();
Console.Writeline("Hello " + name);
```

Essayez de comprendre ce code et la façon d'utiliser ces fonctions.

2.1.2 Exercice 1 : Un phare !

Implémentez une fonction qui affichera un phare de n étages, avec n >= 1.

Le prototype :

```
static void drawme(int size);
```

Exemple :

```
drawme(1);

   .n.
  /___\
  |  |o|
IIIIIII
  |   /|
  |  / |
  |/__|

 drawme(3);

   .n.
  /___\
  |  |o|
IIIIIII
  |   /|
  |  / |
  |/   |
  |   /|
  |  / |
  |/   |
  |   /|
  |  / |
  |/__|
```

Attentions aux boucles infinies ! N'oubliez pas d'incrémenter ou de décrémenter votre compteur.

2.1.3 Exercice 2 : puissance

Implémentez une fonction qui calcule la puissance n d'un nombre entier. Attention aux puissances négatives !

Le prototype :

```
static int pow(int a, int n); //a^n
```

Exemples :

```
pow(2, 2) = 4
pow(9, 1) = 9
pow(42, 0) = 1
```

2.1.4 Exercice 3 : la boucle do while

```
int i = 0;
do {
  Console.Writeline("Hello world n!" + i);
  ++i
} while (i != 10)
```

La boucle ***do while*** est presque la même que la boucle ***while***. Parfois vous avez besoin d'exécuter au moins une fois les instructions présentes dans la boucle même si votre condition est fausse. Dans la boucle ***while***, la condition ne sera testée qu'après l'exécution des instructions. Écrivez une fonction qui demande à l'utilisateur d'écrire "ok" tant que celui-ci n'aura pas répondu "ok".

Le prototype :

```
static void ask();
```

Exemple :

```
Please write "ok"
Answer: no //The user write no
Please write "ok":
Answer: not now //The user write not now
Please write "ok":
Answer: ok //The user write ok
Thank you, bye
```

2.2 La boucle for

2.2.1 Qu'est-ce que c'est ?

Imaginez que vous devez écrire 42 fois la même chose :

```
Console.Writeline("Hello world !" + i);
```

Bien sûr, vous pourriez utiliser la boucle ***while***. Mais il y a une autre boucle, qui pourrait être plus claire et plus courte à écrire : la boucle ***for***.

```
for (initialization; condition; incrementation)
{
  instruction;
}
```

Et voici comment elle marchera avec notre exemple :

```
for (int i = 0; i < 42; ++i)
{
  Console.Writeline("Hello world !" + i);
}
```

2.2.2 Exercice 4 : la boucle simple

D'abord déclarez un tableau A :

```
int[] A = {2, 3, 5, 7, 11, 13, 17};
```

Le prototype :

```
static void ex4(int[] tab));
```

Utilisez une boucle ***for*** afin d'afficher tous les éléments de ce tableau avec ***Console.WriteLine()***. Vous pouvez accéder au n-ième élément avec A[n - 1].

2.2.3 Exercice 5 : la double boucle

D'abord déclarez un double tableau B :

```
int[,] B = new int[3, 3]
                {{1, 2, 3},
                {1, 2, 3},
                {1, 2, 3}};
```

Utilisez deux boucles afin d'afficher tous les éléments de ce tableau avec ***Console.Write***, dans le format suivant :

```
 _____________
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
 _____________
```

Le prototype :

```
static void ex5(int[,] tab));
```

Vous pouvez accéder aux éléments avec B[i, j].

Maintenant, avec le même tableau, affichez ceci :

```
 _____________
| 1 | 2 | 3 |
| 1 | 2 | 3 |
| 1 | 2 | 3 |
 _____________
```

Le prototype :

```
static void inverse_ex5(int[,] tab));
```

3 Un résolveur de sudoku !

3.1 Validation

3.1.1 Vérification d'une ligne

D'abord écrivez une fonction qui vérifie si un élément k manque sur une ligne d'un double tableau de taille [9][9].

```
static bool missing_line(int k, int i, int[,] grille);
```

3.1.2 Vérification d'une colonne

Ensuite écrivez une fonction qui vérifie si un élément k manque sur une colonne d'un double tableau de taille [9][9]..

```
static bool missing_column(int k, int i, int[,] grille);
```

3.1.3 Vérification d'un bloc

Ensuite écrivez une fonction qui vérifie si un élément k manque d'un bloc (i,j) d'un double tableau de taille [9][9].

```
static bool missing_blk(int k, int i, int j, int[,] grille);
```

3.2 Retour sur trace

Le retour sur trace est un algorithme qui est utilisé pour revenir en arrière sur des décisions prisent qui étaient incorrectes. Pour résoudre nos grilles, c'est cet algorithme que nous allons utiliser.

3.2.1 La fonction principale

Cette fonction résout notre grille récursivement en essayant toutes les options possibles et en revenant en arrière s'il n'est plus possible d'avancer.
Exemple :
On appelle d'abord notre fonction sur la case (0,0). Si le numéro dans cette case est différent de 0, on rappelle notre fonction sur la case suivante car il n'y a rien à faire. Sinon, on cherche un nombre qui pourrait être présent sur la case grâce à nos trois fonctions précédentes. Quand on en a trouvé un, on regarde la valeur de retour de l'appel de notre fonction sur la case suivante. Si celle ci est vraie, alors on a gagné, sinon, on essaye de trouver un autre nombre qui pourrait correspondre. Si on en trouve pas, on retourne faux, sinon on retourne vrai.

```
static bool solve(int[,] grille, int i, int j);
```

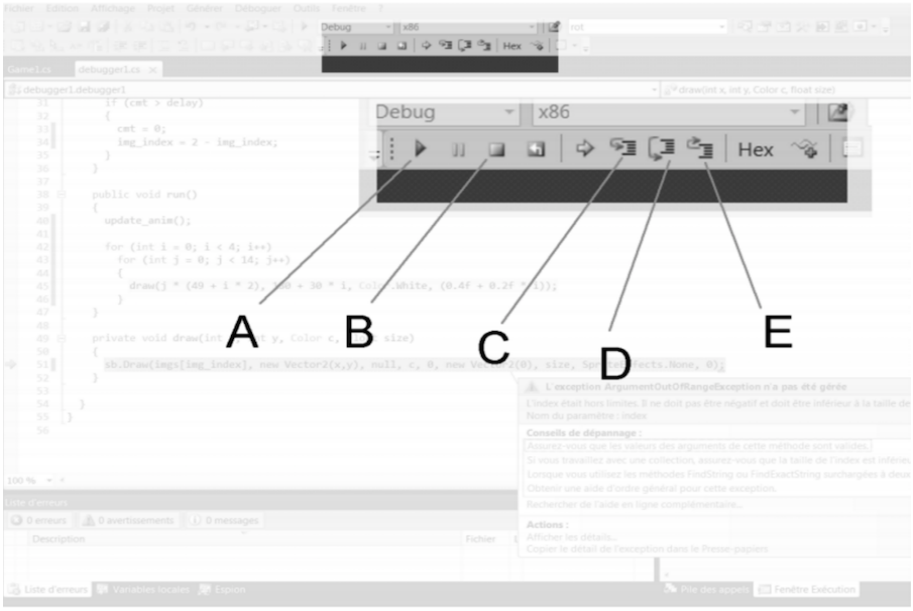
3.3 Un test !

La grille pour essayer vos fonctions :

```
int grid[9][9] =
{{9, 4, 6, 0, 8, 7, 3, 0, 1},
 {0, 0, 3, 0, 0, 0, 0, 0, 2},
 {0, 0, 0, 0, 0, 3, 0, 8, 0},
 {0, 1, 9, 3, 0, 0, 8, 0, 6},
 {0, 0, 7, 6, 0, 2, 5, 0, 0},
 {2, 0, 5, 0, 0, 4, 1, 3, 0},
 {0, 9, 0, 5, 0, 0, 0, 0, 0},
 {6, 0, 0, 0, 0, 0, 4, 0, 0},
 {1, 0, 2, 7, 4, 0, 6, 9, 5},
};
```

4 Debugger

Sur les gros projets, vous allez passer beaucoup de temps à débugger, il est donc important de bien connaître l'interface et les options de debug. Ouvrez la solution debugger.sln et lancez l'exécution du programme avec F5 (la flèche verte). Le programme va "se planter" et vous remarquerez que vous disposez désormais de nouvelles options dans votre menu.



4.1 A : continuer

Cette option vous permet de demander au débugger d'ignorer l'erreur et de passer à la suite, commentez la ligne en jaune et cliquez sur cette flèche, vous verrez alors que le programme continue son exécution.

4.2 B : arrêter le débuggage

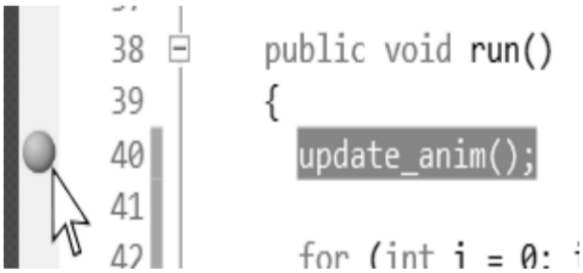
Cette option vous permet de demander au débugger de s'arrêter.

4.3 C, D et E : pas à pas

Pour comprendre ces options, nous allons faire appel à quelque chose que l'on appelle le breakpoint (point d'arrêt en anglais).

Fermez la session de débuggage en appuyant sur le bouton (B).

Maintenant cliquez sur la partie vide de votre interface à gauche sur la ligne updateAnim().



Un petit bouton rouge apparaît, c'est le breakpoint.

Quand votre programme exécutera cette ligne, il s'arrêtera, comme s'il avait rencontré un bug. Relancez alors l'exécution avec F5 et vous le verrez s'arrêter sur cette ligne.

Maintenant vous pouvez avancer pas à pas avec le bouton (C), avancer pas à pas mais sans entrer dans une sous fonction (D) ou bien avancer jusqu'à la sortie de la fonction (E).

Passez un peu de temps à manipuler ces options du debugger, afin de bien comprendre comment elles fonctionnent.

4.4 Exercice : à vous de debugger

Essayez de trouver l'erreur dans debugger.cs et corrigez le !

Indice : Si ça marche, vous devriez voir des trolls danser...