TP C#4 : Tic-tac-toe.

1 Consignes de rendu

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

```
– rendu-tpcs4-login_x.zip
    |– rendu-tpcs4-login_x/
        |– AUTHORS
        |– README
        |– array/
            |– array.sln
            |– array/
                |– Tout sauf OBJ et BIN
        |– Tic-Tac-Toe/
            |– Tic-Tac-Toe.sln
            |– Tic-Tac-Toe/
                |– Tout sauf OBJ et BIN
```

Bien entendu, vous devez remplacer *login_x* par votre propre login. N'oubliez pas de vérifier les points suivants avant de rendre :

- Le fichier AUTHORS doit être au format habituel : * *login_x$* où le caractère '$' représente un retour à la ligne.
- Le fichier README doit contenir les difficultés que vous avez rencontrées sur ce TP, et indiquera les bonus que vous avez réalisés.
- Pas de dossiers bin ou obj dans le projet.
- **Le code doit compiler !**

2 Introduction

Maintenant que vous savez tous utiliser les variables et les différentes boucles en C#, nous allons étudier dans ce TP la définition et l'utilisation des paramètres d'une fonction ainsi que l'utilisation de tableaux.

Ces deux notions sont très importantes et vous seront utiles tout au long de l'année. Pour cela, il est très important de comprendre ce TP, n'hésitez pas à relire la partie cours si cela vous est nécessaire.

3 Cours

3.1 Passage par référence

Pour l'instant, on vous a appris qu'en C# une fonction utilise (ou non) des paramètres et retourne une valeur (ou pas), mais qu'en est-il quand nous voulons retourner plus d'une valeur, si une seule valeur de retour ne nous suffit pas.

De base, lors du passage en paramètre d'une variable, la fonction qui est appelée, va créer une copie de cette variable pour travailler dessus. Les changements effectués sur cette variable n'impacteront donc pas la variable qui a été passée en paramètre. Ceci est le passage par valeur, le mode par défaut lors du passage d'une variable en paramètre. Maintenant, si on souhaite passer en paramètre une variable et que celle-ci soit affectée par les changements effectués dans la fonction, il faut utiliser un autre mode.

Pour cela, en C#, il existe une fonctionnalité lors de la déclaration des paramètres d'une fonction : Le passage par référence à l'aide du mot clé `"ref"` (pour référence). Cette fonctionnalité permet lors du passage d'une variable en paramètre d'une fonction de dire "Je veux que toutes les modifications faites à cette variable dans cette fonction lui soient également appliquées dans la fonction actuelle". Autrement dit, on souhaite utiliser la même variable dans les deux fonctions et que toutes ses modifications soient effectuées dans les deux fonctions.

Essayons par exemple :

```
1 static void change(ref string a, string b)
2 {
3     a = "The answer is";
4     b = "You will never know";
5 }
6
7 static void Main(string[] args)
```

```
8  {
9      string a = "42";
10     string b = "42";
11     Console.WriteLine(a + " " + b);
12     change(ref a, b);
13     Console.WriteLine(a + " " + b);
14 }
```

Comme vous pouvez le voir, les deux lignes sont différentes, dans la seconde le contenu de la variable 'a' a changé mais pas celui de la variable 'b' alors que dans change() les deux sont modifiées.
C'est en utilisant le mot clé `"ref"` lors de la déclaration de la fonction et lorsqu'on place notre variable en paramètre, que l'on utilise le passage par référence et que l'on spécifie à notre machine que nous voulons que cette variable subisse les changements effectués dans l'autre fonction.

3.2 Les tableaux

Quelques notions

Un tableau est un ensemble de valeurs rangées dans un certain ordre. Tous les tableaux possèdent les propriétés suivantes :

- La taille d'un tableau est FIXE, elle n'est plus modifiable une fois le tableau créé.
- Toutes les valeurs dans un tableau sont du même type.
- On peut accéder et changer le contenu d'une des cases du tableau à tout moment.
- Un tableau peut très bien contenir un autre tableau.
- Les éléments sont placés dans des cases indexées de 0 à n-1 avec n le nombre d'éléments.

Déclaration d'un tableau

Pour pouvoir utiliser un tableau, il faut commencer par le déclarer, pour cela, on dispose de plusieurs méthodes :

```
1  int[] array;
2  /* Cette variable contiendra un tableau d'entier.*/
3
4  array = new int[5];
5  // On cree un tableau qui contiendra 5 valeurs qui ne sont pas definies
6
7  int[] bis = {1, 2, 3, 4, 5};
8  // On cree un tableau qui contiendra 5 valeurs qui sont definies
9
10 int[][] tabtab = new int[5][]; // tableau de tableau
11 int[][] tabtab2 = { bis, bis };
12 int[,] dim2 = new int[5, 1]; // tableau a 2 dimensions
13 int[,] dim2bis = { { 1, 2, 3 }, { 4, 5, 6 } };
```

On peut soit déclarer un tableau en initialisant chacun des champs, comme indiqué dans la première méthode. Soit uniquement préciser sa taille sans fixer de valeur pour le moment. Dans les deux cas la taille est définie. Dans la première méthode, la taille est le nombre d'éléments, dans la seconde c'est le nombre spécifié. De plus si on utilise la seconde méthode, tout les champs sont initialisés à 0. Les deux dernières lignes de l'exemple sont des déclarations de tableaux à plusieurs dimension, les deux types de tableaux se comportent de la même façon seule leur utilisation diffère.

Accéder aux valeurs

Pour accéder à une case du tableau, on procède de la façon suivante :

```
1  bis[0]; // retourne 1
2  tabtab2[1][2]; // retourne 6
3  dim2bis[1,2]; // retourne egalement 6
```

On précise dans les crochetss à quelle case du tableau on souhaite accéder. ATTENTION : tenter d'accéder à une case en dehors du tableau fera planter le programme.

Cette méthode permet également de changer le contenu d'une case :

```
1 array[0] = 100;
2 Console.WriteLine("array[0] = {0}", array[0]);
```

Les strings

Les string sont des tableaux de char. Elles possèdent certes de plus grandes fonctionnalités, mais tout ce qui marche sur un tableau, marche sur une string, y compris l'accès aux cases.

4 Exercices

4.1

Vous allez à présent travailler sur la solution *array.sln*. Complétez les fonctions qui se trouvent dans *Function.cs*.

```
|– array/
      |– array.sln
      |– array/
            |– Functions.cs
            |– Other files
```

La fonction Real_length

Cette fonction prend en paramètre une chaîne de caractères sous la forme d'un tableau de caractères et sa longueur passée par référence. Elle modifie cette longueur pour que les espaces à la fin de la chaîne de caractères ne soient plus pris en compte dans la longueur.

```
1 static void real_length(char[] str, ref int length);
```

Exemple :

```
1 int length = 11;
2 char[] a = { 'H', 'e', 'l', 'l', 'o',' ',' ',' ',' ',' ',' ',' '};
```

```
real_length(a, ref length);
// length = 5
```

La fonction count_words

Cette fonction compte le nombre de mots dans une chaîne de caractères. Elle prend en paramètre une chaîne de caractères sous la forme d'un tableau. Les mots sont séparés par des espaces. **Il peut y avoir plusieurs espaces à la suite.**

```
static int count_words(char[] str);
```

La fonction substring

Cette fonction copie une partie du tableau compris entre deux index. Elle prend en paramètre une chaîne de caractère sous la forme d'un tableau, un index de début et un index de fin. Elle renvoie un tableau des caractères contenant la chaîne entre l'indice de début inclus et l'indice de fin exclus.

```
static char[] substring(char[] str, int start, int end)
```

Exemple :

```
char[] a = { 'H', 'e', 'l', 'l', 'o'};
char [] s = substring(a, 1, 4);
//s = { 'e', 'l', 'l'}
```

La fonction get_next_word

Cette fonction retourne le mot suivant l'index en paramètre. Elle prend en paramètres une chaîne de caractères et un index de début du mot. Le mot retourné ne doit pas contenir d'espaces. A la fin s̈tartd̈oit contenir l'index de début mot suivant.

```
static char[] get_next_word(char[] str, ref int start);
```

Exemple :

```
int start = 5;
char[] c = { 'H', 'e', 'l', 'l', 'o',' ','w','o','r','l','d',' '};
s = get_next_word(c,ref start);
//s = {'w','o','r','l','d'};
//start = 11
```

La fonction split_string

Elle prend en paramètre une chaîne de caractères et retourne un tableau contenant chaque mot de cette chaîne. Aucun mot ne doit contenir d'espaces.

```
static char[][] split_string(char[] str);
```

La fonction print_string

Elle prend en paramètre une chaîne de caractères contenu dans un tableau de mots et affiche les mots séparés par des '/'.

```
static void print_array(char[][] tab);
```

```
b={'H','e','l','l','o',' ','h','o','w',' ','a','r','e',' ','y','o','u'};
print_array(split_string(b));
//affiche "Hello/how/are/you"
```

4.2 Tic-tac-toe

4.3 Introduction

Le Tic-tac-toe, plus connu sous le nom de "morpion", est un jeu assez simple à comprendre, de plus pour y jouer on a besoin d'une grille (ça ressemble beaucoup à un tableau non ?). Dans cet exercice nous allons vous demander de recréer une version du Tic-tac-toe, il n'y aura pas de taille précise, et pour gagner il faudra aligner n (avec n la taille d'une ligne) signes, en ligne, en colonne ou en diagonale. Pour cela vous allez devoir compléter plusieurs fonctions présentées ci-dessous.

La fonction init_matrix

Cette fonction créée le tableau à 2 dimensions qui nous servira de grille de jeu. La grille du Tic-tac-toe étant carré le tableau aura le même nombre d'éléments n dans les 2 dimensions. Ce tableau sera initialisé à 0 pour chacune de ses cases pour signifier que personne n'a joué.

```
1 static int[][] init_matrix(int n)
```

La fonction play

Cette fonction permet de remplir une case du tableau lors du tour d'un joueur. La case est donc remplie avec un "1" ou un "2" selon le joueur. La matrice est supposée de la bonne taille dans les 2 dimensions. En revanche, les coordonnées données en paramètres doivent être valides et la fonction retourne si une case a été remplie.

```
1 static bool play(int player, int x, int y, ref int[][] matrix)
```

La fonction print_matrix

Cette fonction affiche le tableau à 2 dimensions qui nous servira de grille de jeu. L'affichage doit ressembler à cet exemple :

```
-------
| |o|x|
|-+-+-|
|x|o|x|
|-+-+-|
| |o| |
-------
```

Un "o" correspond à une case cocher par le joueur 1 et un "x" au joueur 2 ;

```
1 static void print_matrix(int[][] array)
```

Les fonctions check_rows et check_columns

Ces fonctions vérifient que personne n'a gagné en remplissant une ligne entièrement (check_rows) ou une colonne (check_colums). Elles retournent le numéro du joueur gagnant ou 0 sinon.

```
1 static int check_columns(int[][] matrix);
2 static int check_rows(int[][] matrix);
```

La fonction check_diagonals

Cette fonction vérifie que personne n'a gagné en remplissant une des 2 diagonales. Elle retourne le numéro du joueur gagnant ou 0 sinon.

```
1 static int check_diagonals(int[][] matrix);
```

La fonction check_end

Cette fonction vérifie que la grille n'est pas pleine.

```
1 static bool check_end(int[][] matrix);
```

La fonction game

Cette fonction réalise une partie entre 2 joueurs. Elle demande les coordonnées des joueurs et affiche la grille à chaque tour. De plus, elle affiche le joueur gagnant à la fin de la partie ou match nul.

```
1 static void game ();
```

Bonus

Cette fonction réalise une partie avec un seul joueur. Elle demande les coordonnées au joueur 1 et remplit une case aléatoire pour le joueur 2 sauf si il peut gagner en sélectionnant une case. Dans ce cas, cette case est à sélectionner.

```
1 static void auto_game ();
```

Brace yourselves, Deadline is coming.