TP C#3: Once upon a game

Assignment

Archive

You must submit a zip file with the following architecture :

```
rendu-tp-firstname.lastname.zip
|-- firstname.lastname/
    |-- AUTHORS
    |-- README
    |-- Basics/
        |-- Basics.sln
        |-- Basics/
            |-- Everything except bin/ and obj/
    |-- Takuzu/
        |-- Takuzu.sln
        |-- Takuzu/
            |-- Everything except bin/ and obj/
```

- You must obviously replace *firstname.lastname* with your login (the format of which usually is *firstname.lastname*).
- The code **MUST** compile.
- In this practical, you are allowed to implement methods other than those specified. They will be considered bonuses. However, you must keep the archive's size as small as possible.

AUTHORS

This file must contain the following : a star (*), a space, your login and a newline.
Here is an example (where $ represents the newline and ␣ a blank space):

```
*␣firstname.lastname$
```

Please note that the filename is AUTHORS with **NO** extension. To create your `AUTHORS` file simply, you can type the following command in a terminal:

```
echo "* firstname.lastname" > AUTHORS
```

README

In this file, you can write any and all comments you might have about the practical, your work, or more generally about your strengths and weaknesses. You must list and explain all the bonuses you have implemented. An empty `README` file will be considered as an invalid archive (malus).

1 Introduction

1.1 Objectives

Now that you know how to use variables and some loops in C#, we are going to study in this Practical the definition and use of the parameters of a function and of arrays.

Both concepts are really important and will be useful all year. That is why it is really important that you understand this Practical. Do not hesitate to reread the lecture notes if you encounter difficulty.

2 Courses

2.1 assing Reference-Type Parameters

For now, you have learnt that in C#, a function uses (or not) some parameters and returns a value (or not).
What if we want to return more than one value, if only one return value is not enough.
When you use a variable as a parameter of a function, it will create a copy of the variable that will be used while working on the function. The changes that are applied in the function that is called won't appear in the variable that we used as a parameter. This is called a Value-Type parameter, the default mode when you use a variable as a parameter.
Now, if we want the current variable to change after using it as a parameter of a function, and the changes applied to this variable in the called function.

In C#, there is a functionality when parameters of a function are declared : using Ref variable and not a copy of it in the function that we call. We use Reference-Type Parameters with the key word "ref" (for reference). This functionality allows the user to say "I want any modification of the variable in the function called to be applied on this variable" when using a variable as a parameter of a function.
In other words, we want to use a variable in both functions and all modifications should be done in both functions.

For example :

```
static void change(ref string a, string b)
{
    a = "These violent deadlines";
    b = "have violent ends.";
}

static void Main(string[] args)
{
    string a = "ACDC 2020";
    string b = "are the best.";
    Console.WriteLine(a + " " + b);
    change(ref a, b);
    Console.WriteLine(a + " " + b);
}
```

```
>./main
ACDC 2020 are the best.
These violent deadlines are the best.
```

As you can see, both lines are different. In the second one, the variable 'a' has changed, but not 'b', whereas both were changed in the "change" function. It is by using the key word "ref" when declaring a parameter and when using a variable as parameter of a function that we use Passing Reference-Type Parameters ; and that we specify to the computer that we want this variable modified when a change is applied in the other function.

2.2 Array

2.2.1 Some concepts

An array is a set of values arranged in a specific order. All arrays have the following characteristics :

- An array is always passed to a function by reference, no need to put "ref" in the prototype or the call of the function.
- The size of an array is fixed and cannot be modified after the creation of the array.
- All values in an array have the same type.
- We can access and change the value of a case at any moment.
- An array can contain another array.
- Values are placed in cells from the index 0 to n-1 where n is the size of the array.

2.2.2 Declaration of an array

To use an array we need to start by declaring it. For this we have some possibilities :

```
int[] array;
/* This variable will contain an array of int*/

array = new int[5];
// We create an array that will contain 5 values that we don't know.

int[] bis = {1, 2, 3, 4, 5};
//  We create an array that will contain 5 values that we already know.
int[][] tabtab = new int[5][];// array of array
int[][] tabtab2 = { bis, bis };
int[,] dim2 = new int[5, 1];// two dimensional array
int[,] dim2bis = { { 1, 2, 3 }, { 4, 5, 6 } };

public static Replace(int[] tab, int i, int val)
{
    tab[i] = val;
}
Replace(bis, 2, 12);
//bis = {2, 1, 12, 4, 5} now.
```

We can declare an array either with its value as in the first method or by giving to the computer only the size that we want for the array without giving any value to put in the array. In both cases, size is fixed and cannot be changed afterwards. In the first method, the size is the number of values that we put in the array. If we use the second method all cells of the array are set to 0. The last four lines are declarations of multi-dimensional array, and both types work in the same way, only their use changes.

2.2.3 Access a value

To access a value, we procced like this :

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

We put between brackets the index of the cell that we want to access.
WARNING : trying to access a space outside of the array will make your program crash !
This method allows the user to change the content of a cell :

2.2.4 Strings

Strings are arrays of "char" , it means that they have the same behaviour as an array, plus some features.

3 Exercises

3.1 Variables

We are going to see some basic uses of references in C#. It will allow you to better understand these concepts, which are very useful.

3.1.1 Let's swap

Write a function that swaps 2 integers. As we told you, it's basic.

```
public static void Swap(ref int a, ref int b);
//exemple:
int a = 1;
int b = 5;
Swap(ref a, ref b);
//a = 5 and b = 1 now.
```

3.1.2 Let's work on float !

Code a function that returns the floor part of a *float* and replace the value of the variable by its decimal part.

```
public static int Trunc(ref float f);
```

3.1.3 RotChar

You wanted it? They did it! *RotChar* is back. You have to swap uppercase with uppercase, lowercase with lowercase and digit with digit. You know the story, your function must follow the following prototype:

```
public static void RotChar(ref char c, int n);
```

WARNING: 'n' can be negative.

3.2 Arrays

Now that you have seen how arrays work, it is time to use them.

3.2.1 Let's search a bit

Write the function *Search*, which takes an array of integers *arr* as argument and returns the position of the first occurrence of the element *e* in it (-1 if *e* is not in the array).

```
public static int Search(int[] arr, int e);
```

3.2.2 King of the hill

Write the function *KingofTheHill* that takes an array of integers *arr* and returns the king of the hill.
Hills are represented this way :

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

A sequence of integers increasing then decreasing, the king of the hill is the integer at the top, 9 in our example.

Be aware that a simple search for maximum will not garantee you a complete mark.

3.2.3 The clone army

Write the function *CloneArray* that takes an array of integers *arr* and returns a copy of it.

```
static int[] CloneArray(int[] arr)
```

3.2.4 Some bubbles and sorting

Implement the bubble sort :

```
void BubbleSort(int[] arr);
```

The BubbleSort is a basic sorting algorithm, it works this way :

- Traverse the whole array
- Compare the current element and the next
- If they don't respect the sorting condition, swap them
- Restart until the swapping stops.

Congratulation, your array is now sorted.

For more information : `https://en.wikipedia.org/wiki/Bubble_sort`

Here, the array has to be sorted in increasing order.

```
  int[] arr = {2, 4, 6, 1, 4, 9, 0, 5, 2};
  BubbleSort(arr);
  /* arr = {0, 1, 2, 2, 4, 4, 5, 6, 9} */
```

Bonus: implement another sorting algorithm. You must specify the name and why it is more efficient than bubble sort in the README.

3.3 Takuzu

The Takuzu is a Belgian logic based game created by Frank Coussement and Peter De Schepper in 2009. It may be from 6x6 grid to 14x14 in general, but it can have a different number of columns and rows as long as there is an even number of them. each grid only contains 0 and 1 and must be completed by following three rules :

- As many 0s and 1s in each completed row and each completed column
- No more than 2 identical digits side by side
- No two rows nor columns can be identical

More informations : `https://en.wikipedia.org/wiki/Takuzu`

We will represent a grid by a two dimension array, with empty cells worth '-1'
Thus the following code :

```csharp
int[,] grid =
{
  {-1, -1,  0, -1, -1, -1, -1, -1},
  { 1, -1,  1, -1, -1, -1, -1, -1},
  {-1, -1, -1, -1,  0,  0, -1, -1},
  {-1, -1, -1,  1, -1, -1, -1, -1},
  {-1, -1,  0, -1, -1, -1, -1,  0},
  {-1, -1, -1, -1,  0, -1, -1,  0},
  { 0, -1, -1,  0,  0, -1, -1, -1},
  { 0, -1, -1, -1, -1,  0,  0, -1}
};
Takuzu.PrintGrid(grid);
```

Displays this :

```
   0 1 2 3 4 5 6 7
0 | | |0| | | | | |
1 |1| |1| | | | | |
2 | | | | |0|0| | |
3 | | | |1| | | | |
4 | | |0| | | | |0|
5 | | | | |0| | |0|
6 |0| | |0|0| | | |
7 |0| | | | |0|0| |
```

3.3.1 Gutenberg

Write the function :

```csharp
void PrintGrid(int[,] grid);
```

This displays the grid following this format :

- the number of the column on top of each column
- the number of each row to the left of each row, followed by a space

- each row begins and ends by a '|'
- each cell is separated by a '|'
- empty cells are represented by a space

3.3.2 Is it valid ?

Write the following functions :

```
bool IsRowValid(int[,] grid, int row);
bool IsColumnValid(int[,] grid, int col);
```

They check that the row *row* (respectively the column *col*) of the grid *grid* can still respect the first two conditions.
Then write the function :

```
bool IsGridValid(int[,] grid);
```

that checks that each row and each column is valid and unique.

3.3.3 Let's play a bit

Write the function :

```
bool PutCell(int[,] grid, int x, int y, int val);
```

That puts the value *val* at the position (x,y). The function does not modify the grid and returns *false* if :

- *val* is not a valid value
- the coordinates x or y are not in the boundaries of the array
- the grid is not valid after the modification.

The function returns *true* and modifies the grid in all other cases.

3.3.4 Let's play a lot

Write the function :

```
void Game(int[,] start);
```

That starts a game of Takuzu from the grid *start*.
It takes place in the following way :

- Display the grid
- Ask for coordinates
- Ask for a value

```
   0 1 2 3 4 5 6 7
0 | | |0| | | | | |
1 |1| |1| | | | | |
2 | | | | |0|0| | |
3 | | | |1| | | | |
4 | | |0| | | | |0|
5 | | | | |0| | |0|
6 |0| | |0|0| | | |
7 |0| | | | |0|0| |
x : 1
y : 1
value : 0
   0 1 2 3 4 5 6 7
0 | | |0| | | | | |
1 |1|0|1| | | | | |
2 | | | | |0|0| | |
3 | | | |1| | | | |
4 | | |0| | | | |0|
5 | | | | |0| | |0|
6 |0| | |0|0| | | |
7 |0| | | | |0|0| |
x :
```

The grid is not modified if the coordinates match a cell already set in the original grid or if *PutCell* returns *false*.
A message is displayed when the grid is finished. Be creative!

3.3.5 Bonus

Write the function :

```csharp
void AI(int[,] grid);
```

That finishes the grid. Don't be afraid to think by yourselves. However, you must explain your algorithm in the README.

These violent deadlines have violent ends.