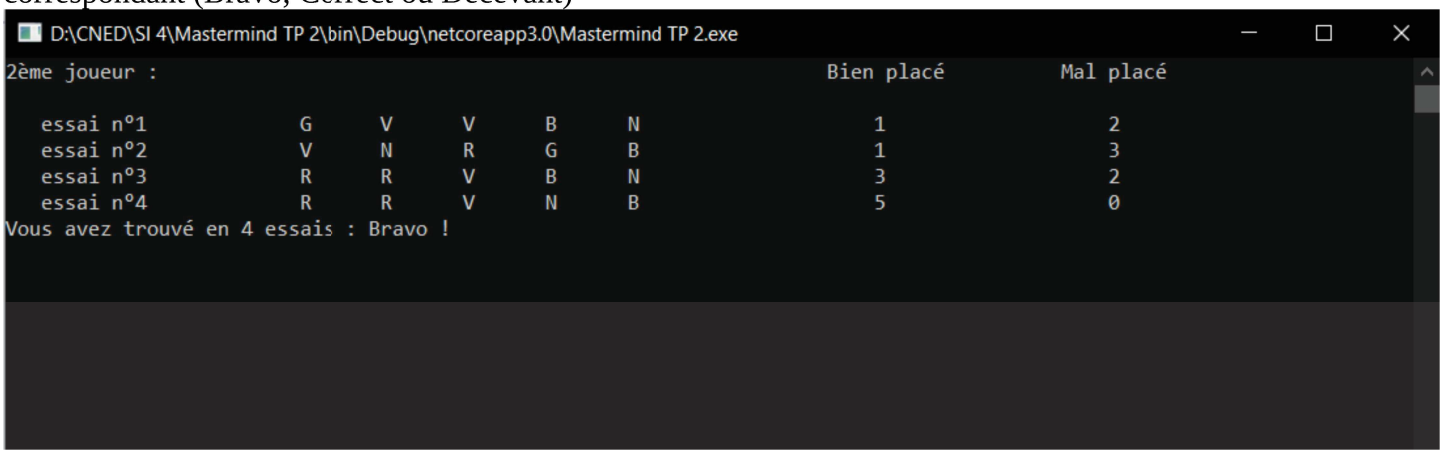
Fiche de Synthèse : Mastermind *Novembre 2019*

Développement :
Il s'agit d'une adaptation du celèbre jeu Mastermind, développé en mode console.
Déroulement du jeu : un 1er joueur saisi 5 lettres (sur 7 au choix) et ensuite un 2ème joueur doit trouver la bonne combinaison. A chaque essai, le jeu lui indique le nombre de lettres bien placées, et le nombre de lettres mal placées.
Lorsqu'il a trouvé, le nombre d'essais qui ont été nécessaires est affiché, ainsi que le commentaire correspondant (Bravo, Correct ou Decevant)



Cadre :
Il s'agit d'un TP proposé dans le cadre du cours "Bases de la Programmation ", par Elisabeth Martins Da Silva.

Support :
Développé en C# avec Visual Studio Community 2019.

Contraintes :
C'est un TP qui illustre la programmation procédurale modulaire.

Difficultés rencontrées :
Pas de difficultés.

Description détaillée du développement

J'ai suivi les étapes suivantes pour le développement:
- saisie de la formule à trouver
- passage au second joueur, saisie des essais
- calcul des biens et mal placés
- fin du jeu

1. Saisie de la formule à trouver

Le joueur 1 doit donc saisir la combinaison à faire deviner : il doit choisir 5 lettres parmis 7 (symbolisant des couleurs), le curseur doit se placer au bon endroit, et bien sûr la combinaison gagnante doit être enregistrée dans un tableau de " char ".
Tout ceci peut se faire en développant les fonctions ***controleChar(Char caractere)*** qui vérifie que le caractère saisi est valide, et ***saisie (int numLigne, int numColonne)*** qui remet le curseur sur le caractère s'il est pas validé par la précédente fonction.

Ainsi, je peux maintenant écrire ***rempliT(int numeroLigne***), une procédure qui utilise les fonctions précédentes, déplace le curseur sur la colonne suivante quand c'est OK, et enregistre dans un tableau au fur et à mesure.

```
// Procédure remplissage tableau combinaison
2 références
static Char[] rempliT(int numeroLigne)
{
    Char[] tableau = new Char[5];

    int numCol = 25;
    for (int i = 0; i < 5; i++)
    {
        Char couleurAjoutee = saisie(numeroLigne, numCol);
        numCol += 7;
        tableau[i] = couleurAjoutee;
    }
    return tableau;
}
```

Ainsi, dans le programme principal, tout est fait, il n'y a plus qu'à utiliser :

```
// 1er écran du jeu :
Console.Write("1er joueur : ");
Char[] combinaisonG = rempliT(0);

// effacement de l'écran :
Console.Clear();
```

2. Passage au second joueur, saisie des essais

C'est au tour du joueur 2 : il faut lui afficher une 1ère ligne avec les en-têtes de colonnes.
Et démarrer une boucle, qui affichera le n° de l'essai et lui demandera de saisir son essai.
Rien de " neuf " finalement, j'ai bien travaillé puisque j'ai ma petite procédure rempliT(), je peux donc la ré-utiliser pour enregistrer la saisie du joueur 2 :

```
// déclarations :
int bienPlace, malPlace, essais = 0;

// 2ème joueur :
Console.Write("2ème joueur : ");
Console.SetCursorPosition(70, 0);
Console.Write("Bien placé");
Console.SetCursorPosition(90, 0);
Console.Write("Mal placé");

// le jeu, qui dure tant que la combinaison n'est pas trouvée, i.e 5 bienPlacé
do
{
    essais += 1;

    // Copie du tableau de la combinaison gagnante
    Char[] combiCopie = new Char[5];
    Array.Copy(combinaisonG, combiCopie, 5);

    // Placement du curseur pour les essais + on remplit le tableau des combinaisons proposées
    Console.SetCursorPosition(3, (essais + 1));
    Console.Write("essai n°" + (essais));
    Char[] combinaisonE = rempliT((essais + 1));
```

3. Calcul des biens et mal placés

Il y a une petite difficulté pour ce calcul : en effet, les caractères qui sont considérés comme “ bien placés ” ne doivent pas être recomptés dans le nombre de “ mal placés ”
Pour que ça n’arrive pas, ma solution était, quand un caractère était compté comme étant “ bien placé ”, alors il soit remplacé dans les 2 tableaux :

```
// Calcul des bien placés :
1 référence
static int bienPlaceFct(Char[] tableG, Char[] tableP)
{
    int bienMis = 0;
    for (int i=0; i < 5; i++)
    {
        if (tableG[i] == tableP[i])
        {
            bienMis += 1;
            tableG[i] = 'X'; // je remplace les lettres trouvées par des X, pour ne pas les recompter ensuite
            tableP[i] = 'Y'; // idem avec la combinaison proposée
        }
    }
    return bienMis;
}
```

J’ai fait la même chose bien sûr avec les mals placés.
Du coup, mon petit programme avance bien :

```
            // Décomptes des biens et mals placés à l'aide des fonctions
            bienPlace = bienPlaceFct(combiCopie, combinaisonE);
            malPlace = malPlaceFct(combiCopie, combinaisonE);

            // Affichage
            Console.SetCursorPosition(74, (essais + 1));
            Console.Write(bienPlace);
            Console.SetCursorPosition(94, (essais + 1));
            Console.Write(malPlace);

        } while (bienPlace != 5);
```

4. Fin du jeu

Une fois que la bonne combinaison de lettres/couleurs a été trouvée, il ne reste plus qu’à afficher, en-dessous, le nombres d’essais (on le connais, puisqu’on les as comptés au fur et à mesure), et le compliment qui va avec :

```
// Il faut revenir à la ligne, puis afficher le compliment correspondant :
Console.WriteLine();
Console.WriteLine(compliment(essais));

Console.ReadLine();
```

Bien sûr, une procédure ***compliment(int essais)*** a été écrite au préalable.