PARTIEL C# - Sujet 3

Vendredi 10 Janvier 2014, 16h-19h

1 Consignes de rendu

A la fin de ce TP, vous serez dans l'obligation de rendre une archive respectant l'architecture suivante :

```
-- rendu-tpcs1-login_x.zip
   |-- login_x/
       |-- AUTHORS
       |-- README
       |-- BONII.txt
       |-- SkillCheck
           |-- SkillCheck.cs
       |-- MySmallSnake
            | -- Properties/
            | -- MySmallSnake.csproj
            | -- Program.cs
            | -- Snake.cs
            | -- Position.cs
       | -- MySmallSnake.sln
```

Bien entendu, vous devez remplacer `login_x` par votre propre login.

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

Fichier AUTHORS

Le fichier nommé `AUTHORS` doit se trouver à la racine de votre rendu. Il devra contenir tous les auteurs du projet (en l'occurrence, VOUS seulement). Le format, pour chaque auteur, est le suivant :

— Le caractère '*' suivi d'un espace.
— Le login de l'étudiant.
— Un retour à la ligne.

Conseils et remarques

Lisez le sujet dans son intégralité avant de commencer. Une fois que vous l'avez lu, relisez-le. Réalisez les exercices dans l'ordre car ils sont en difficulté croissante.
N'oubliez pas de vérifier les points suivants avant de rendre :

— Le fichier `AUTHORS` doit être au format habituel (rappelé ci-dessus).
— Aucun dossier `bin` ou `obj` dans le rendu.
— Les fonctions demandées doivent respecter le prototype donné
— **Le code doit COMPILER**.

Lorsque cela n'est pas précisé, vous êtes totalement libre quant à l'implémentation des fonctions.

2 SkillCheck (60 min - 6 pts)

Cette partie a pour but de valider vos connaissances en C# dans les domaines suivants :

— Création de fonctions récursives et itératives.
— Manipulation des données de types int, double et des chaînes de caractères (string).
— Utilisation des boucles (for et while).
— Passage de variables par référence.
— Gestion des exceptions.

Tous les exercices de cette partie devront être écrits dans le fichier "SkillCheck.cs" dont le namespace est "SkillCheck". Pour ce faire, créez une application console nommée "SkillCheck" (pour le namespace) et renommez le fichier "Program.cs" en "SkillCheck.cs".
Les fonctions devront être toutes appelées dans la fonction main() avec les valeurs de votre choix. L'affichage de chaque fonction doit être séparé par une ligne vide.

Hello world ! (5 min - 0.5 pts)

Vous devez écrire la fonction `hello_world` qui affiche la chaîne de caractères "Hello World !" à la console.
Prototype :

```
static void hello_world();
```

Puissances (15 min - 1.5 pts)

Vous devez écrire deux fonctions `pow(a, n)` qui retournent le résultat de a puissance n.
Une fonction sera récursive et aura le prototype suivant :

```
static int pow_rec(int a, int n);
```

L'autre fonction sera itérative et aura le prototype suivant :

```
static int pow_it(int a, int n);
```

Les deux fonctions devront évidemment renvoyer le même résultat.
Si l'entier n est négatif, vous devrez arrêter l'exécution avec une exception du type `ArgumentOutOfRangeException()`.

Fibonacci (10 min - 1 pt)

Vous devez écrire la fonction `fibo(n)` qui retourne le résultat de fibonacci n.
Prototype :

```
static int fibo(int n);
```

Rappel :

```
# let fibo = function
  | 0 -> 0
  | 1 -> 1
  | n -> fibo(n - 1) + fibo(n - 2);;
```

Si l'entier n est négatif, vous devrez arrêter l'exécution avec une exception du type `ArgumentOutOfRangeException()`.

Swap (10 min - 1 pt)

Ecrivez la fonction swap(a, b) prenant en paramètres 2 entiers a et b passés en référence et qui échange les contenus de a et de b.
Prototype :

```
static void swap(int ref a, int ref b);
```

Rotation (20 min - 2 pts)

Ecrivez la fonction rot(char c, int n) qui effectue une rotation de n sur le caractère c. Par exemple, si le caractère passé en argument est 'a' et qu'on lui applique un décalage de 13, la fonction retourne 'n'.
Prototype :

```
static char rot(char c, int n);
```

La rotation ne sera appliquée que sur les caractères alphabétiques.
Conseil : pour tester votre fonction, utilisez `str = rot(rot(str, 13), 13)`

Ecrivez la fonction rotn(string str, int n) qui effectue une rotation de n sur tous les caractères de la chaine donnée en paramètre.
Prototype :

```
static string rotn(string str, int n);
```

Conseil :
Utilisez un objet de type StringBuilder (contenu dans l'espace de noms `System.Text`) pour construire petit à petit une chaîne de caractères. Les deux méthodes à retenir sont `Append` qui ajoute une chaîne de caractères à la suite des autres, et `ToString` qui sert à obtenir la chaîne de caractères complète.

Si l'entier n est négatif, vous devrez arrêter l'exécution avec une exception du type `ArgumentOutOfRangeException()`.

3 Partie II : MySmallSnake

Cette partie a pour but de valider vos connaissances en C# dans les domaines suivants :

— Utilisation avancée de la console sous Windows.
— Utilisation de boucles, fonctions et variables.

Nous allons créer ensemble un jeu vidéo ! Au terme de cette partie, nous aurons un jeu en console représentant un serpent qui bouge à l'écran, contrôlé par le joueur (nous !) et qui mangera des pommes à l'infini. Bien sûr, mode console oblige, tout cela sera sous forme de caractères ! Par ailleurs, pour garder une difficulté raisonnable, la queue du serpent ne sera pas à faire... sauf si vous le souhaitez (voir partie Bonii).

3.1 Bien démarrer

Commençons par créer un nouveau projet console nommé **"MySmallSnake"**. Ajoutons une boucle `while(true)` qui ne fait rien dans le main. Parfait ! Maintenant lorsqu'on lance le programme, la console ne se ferme pas !

3.2 La gestion du Serpent

Afin de garder notre projet lisible et propre, nous allons travailler dans différents fichiers. Pour faire cela, nous allons d'abord changer la ligne :

```
class Program
```

pour qu'elle devienne

```
partial class Program
```

Cela nous permettra de répartir le programme sur plusieur fichiers.

Maintenant, créons la classe **"Snake.cs"**. À nouveau, changez la ligne

```
class Snake
```

pour qu'elle devienne

```
partial class Program
```

C'est donc dans **"Snake.cs"** que vous écrirez toutes les fonctions de gestion du serpent.

Le serpent devra constamment avancer dans une seule direction. Sa direction pourra changer si l'utilisateur appuie sur une touche. Créez donc trois variables :

— `snakePosX` qui sera un entier représentant la position du joueur en X
— `snakePosY` qui sera un entier représentant la position du joueur en Y
— `snakeDir` qui sera une string qui pourra prendre les valeurs "up", "down", "right" ou "left" selon les déplacements du joueur.

Le serpent a ainsi une direction et une position qui lui est propre.
Profitons-en pour écrire une fonction qui initialisera la position au milieu de l'écran, et la direction à droite.

```
static void snake_init(int x, int y);
```

Maintenant, vous allez devoir créer plusieurs fonctions qui seront appelées dans la boucle de jeu et qui manipuleront le serpent.

```
static void snake_go();
static void snake_turn(ConsoleKey key);
static bool snake_collision(int positionX, int positionY);
static void snake_draw();
```

La fonction `snake_go()` fait avancer la position du serpent en fonction de sa direction.

La fonction `snake_turn(ConsoleKey key)` modifie la direction selon la touche sur laquelle l'utilisateur a appuyé (passée en argument).

La fonction `snake_collision(int positionX, int positionY)` compare la position passée en argument et celle du joueur et renvoie vrai si les deux position sont égales. Faux, sinon.

La fonction `snake_draw()` dessine le serpent à sa position : elle doit déplacer le curseur puis écrire le caractère correspondant, qui devrait être 'X'.

3.3 L'initialisation du jeu

L'initialisation du jeu doit se trouver hors de la boucle de jeu. Elle va nous permettre d'initialiser les différents éléments et notamment, d'assigner une position à la pomme.
Pour cela, retournons donc dans le fichier `Program.cs` et créons d'abord deux variables :

```
static int applePosX;
static int applePosY;
```

Pour que le jeu soit ludique, il faut que la pomme soit positionnée de manière aléatoire. Pour faire cela, vous allez devoir utiliser la classe Random :

```
static Random rand = new Random();
```

Il vous suffit ensuite d'utiliser `rand.Next(int maxValue)` pour générer des nombres aléatoires.

Finalement, n'oubliez pas d'initialiser le joueur en faisant un simple appel à la fonction associée au serpent : `snake_init`.
Conseil :
Utilisez la valeur maximale de la fonction `rand.Next()` pour éviter de placer la pomme hors de la fenêtre de la console.

3.4 La boucle de jeu

Maintenant que vous avez tout ce qu'il faut, il ne reste plus qu'à assembler les pièces du puzzle. Pour cela, il faut tout regrouper dans la boucle de jeu puisque c'est là que tout se passera.
Conseil :
Vous pouvez diviser la boucle de jeu en plusieurs fonctions comme, par exemple : `update()`, `draw()`, etc.

Les différentes étapes à effectuer sont :

3.4.1 Vérifier s'il y a collision entre le joueur et la pomme

Utilisez la fonction associée au serpent : `snake_collision`. S'il y a effectivement collision, il vous faudra déplacer la pomme à un autre endroit aléatoire : réinitialiser l'élément, en somme !

3.4.2 Faire avancer le joueur

Utilisez la fonction associée au serpent.

3.4.3 Récupérer les entrées du joueur

Pour cela, vous devez utiliser les méthodes et les attributs de la classe console disponible : `Console.KeyAvailable` et `Console.ReadKey(true)` vous seront utiles ! Lisez bien la documentation disponible dans Visual Studio. Vous pouvez alors utiliser la fonction associée au serpent.

3.4.4 Dessiner le jeu

Videz la console, utilisez fonction associée au serpent pour dessiner le joueur puis, de la même manière, dessinez la pomme.

3.4.5 Attendre

En effet, si vous lancez le jeu tel quel, vous ne verrez pas grand-chose, si ce n'est des caractères se déplaçant très vite (trop vite, d'ailleurs). Pour résoudre ce problème, il suffit de mettre le jeu en pause pendant un court laps de temps. Pour cela, utilisons la fonction `Sleep(int milliseconds Timeout)` qui, comme son nom l'indique, endort le processus.

```
System.Threading.Thread.Sleep(int milliseconds Timeout);
```

Réglez le Timeout à 50 millisecondes pour un résultat convenable.

3.5 Bonii

Voilà ! Votre jeu est fini... Mais l'est-il vraiment ?

En effet, il reste bien trop simple alors à vous d'ajouter le nécessaire pour le compléter. Pour cette partie, vous avez carte blanche : faîtes ce que bon vous semble, tant que ça n'empiète pas sur la partie obligatoire.
Chaque bonus que vous réalisez doit être courtement expliqué dans un fichier "BONII.txt".

Quelques idées de bonii :

- Limiter les déplacements du joueur à l'intérieur de la fenêtre pour qu'il ne puisse pas sortir.
- Ajouter un score, affiché dans la console, qui augmente à chaque fois qu'une pomme est ramassée.
- Multiplier les pommes dans le jeu !
- Ajouter un game over (quand le joueur touche les limites, par exemple).
- Ajouter la queue du serpent !