TPC#5 Touché Coulé

Table des matières

1 Préliminaires

N'oubliez pas de lire le sujet en entier avant de commencer le tp.

Pour la bonne réalisation du tp, des fichiers sont à récupérer sur `perso.epita.fr/~querou_a`

Des pénalités pourront être appliquées en cas de non respect des règles suivantes :
- Un fichier AUTHORS doit être présent à la racine du rendu. Il doit contenir une étoile *, un espace, puis votre login (ex : login_x) suivi d'un retour à la ligne.
- La solution doit compiler,
- Le code doit être propre, correctement indenté. N'oubliez pas que `Visual Studio` dispose d'outils de mise en forme automatique, ce qui justifie les lourdes pénalités appliquées en cas d'abus.
- La triche ou l'échange de code sont sévèrement punis.

2 Le Cours

2.1 Les Classes

2.1.1 Généralités

Ca y est, enfin vous allez rentrer dans la cours des grands et utiliser la programmation orienté objet ! Tous cela nous ramène à la question suivante : qu'est-ce qu'un objet ? Si on se réfère à la définition de Wikipedia :

En informatique, un objet est un conteneur symbolique, qui possède sa propre existence et incorpore des informations et des mécanismes en rapport avec une chose tangible du monde réel, et manipulés dans un programme

Les **classes** ne sont que des conteneurs d'objets. C'est dans les classes que le programmeur définit les caractéristiques de l'objet telles que des attributs et des méthodes propre à l'objet. En CSharp, la déclaration de classe se fait comme ci-dessous :

```
public class Human
{
  private int age;       //  Attribut specifique a la classe Human
  private bool sex;      //

  ...

  public int getAge()    //  Methode specifique a la classe Human
  {
    return age;
  }

  ...
}
```

Si vous ne le savez pas déjà une **méthode** est une routine associée à une classe. Pour faire simple, une methode est une fonction à l'intérieur d'un objet qui peut acceder à ses attributs directement.

Un **attribut**, pour faire simple, est une variable dans un objet accessible par toutes les méthodes de la classe.

2.1.2 Instanciation

Une classe ne s'utilise pas telle qu'elle. Il ne suffit pas d'appeller la fonction `getAge()` pour obtenir l'âge. Il faut tout d'abord instancier la classe, chaque instance possède ses propres variables `age` et `sex`. L'instanciation ce fait comme ci-dessous :

```
Human valentine = new Human();
```

Pour accéder a une méthode d'une classe il suffit de faire :

```
type retour = objet.methode(arguments);

//exemple
int ageDeValentine = valentine.getAge();
```

2.1.3 Constructeur

Il existe plusieurs méthodes pour instancier une classe. La méthode par défaut est `new Human()`. Mais il serait utile d'initialiser les variables `age` et `sex` à l'initialisation. Pour cela on utilise un constructeur. Un constructeur est une méthode de la classe qui permet l'instanciation de celle-ci.

Dans le cas de la classe `Human`, on pourrait définir un constructeur comme :

```
public class Human
{
    ...
    Human(int age_, bool sex_)   // Cree une instance de la classe Human
    {                            // et initialise ses champs
        this.age = age_;
        this.sex = sex_;
    }
    ...
}
```

Si les constructeurs existent, les destructeurs existent aussi ! Il permettent de détruire proprement une instance. Cependant vous n'aurez pas à les utiliser avant l'ING1, ne vous en souciez donc pas et profiter du garbage collector.

2.2 Enregistrements

En CSharp une structure (ou enregistrement) est très similaire à une classe. Une structure peut contenir des variables et des méthodes. Il est préférable d'utiliser une structure à la place d'une classe si l'on souhaite s'en servir uniquement pour stocker des variables et ainsi représenter un groupement de variables.
Pour définir une structure en CSharp :

```
public struct Point
{
   int x;
   int y;
}
```

Pour initialiser une structure, il suffit de faire :

```
Point p;          // On cree la structure
p.x = 101010;     // On renseigne le champ x
p.y = 42;         // On renseigne le champ y
```

2.3 Exceptions

Les exceptions représentent des opérations que votre programme n'arrive pas à exécuter. L'exception la plus connue survient lorsque vous essayer de faire :

```
n = i / 0; // Pauvre fou
```

De base, un tel calcul fera crasher votre programme. Cependant il vous est possible de rattraper les exceptions afin de réagir correctement au problème survenu.

Pour rattraper une exception il faut utiliser les mots clés `try catch` (et optionnellement finally).

```
int n = 0;
int i = 100;
try
{
    n = i / 0;  // Lance l'exception DivideByZeroException
}
catch (System.Exception e)
{
    Console.Writeline("Exception occured : {0}", e.Message); // On
        affiche le probleme
}
```

Il vous est aussi possible de créer vos propres exceptions. Une exception étant un objet, vous devez pour cela créer une classe.

```
public class MyException : System.Exception
{
    public MyException() : base() { }
    public MyException(string message) : base(message) { }
}
```

Dans le code ci-dessus nous définissons l'exception `MyException` qui hérite (vous verrez cela plus tard) de la classe System.Exception. Cette classe possède deux constructeurs. Le premier ne prenant pas d'argument, le second prenant une string comme argument. Vous n'avez pas à implémenter le contenu des constructeurs, le mot clé `base` (ça aussi vous le verrez plus tard) s'en occupe à votre place.

Enfin si dans votre code vous repérez un comportement anormal et souhaiter le gérer grace aux exceptions vous pouvez utiliser le mot clé `throw` :

```
throw new MyException("Bam... Exception");
```

3 Touché Coulé

Lors de ce TP vous allez devoir créer un jeu de touché coulé. Vous devriez normalement tous savoir comment une partie se déroule mais au cas où, vous pouvez toujours aller voir wikipedia (http ://fr.wikipedia.org/wiki/Bataille_navale_(jeu)).

Rassurez vous, la version qui vous est demandée est simplifiée mais libre à vous d'implémenter les bonus les plus fous pour impressionner vos ACDCs !

Un squelette vous est fourni pour ce TP, vous le remplirez au fur et à mesure. En suivant les instructions pas à pas vous ne devriez pas rencontrer de problème mais surtout ne sautez pas d'étapes, au quel cas vous serez livré à vous même.

3.1 Terrain.cs

Avant de commencer, pensez à copier le fichier map.txt dans le dossier debug

Tout d'abord nous allons commencer en implémentant le terrain, c'est à dire les maps sur lesquelles vous allez jouer. Pour ce faire vous allez remplacer les FIXME dans le fichier Terrain.cs et ce sera ainsi pour toute la suite du TP.

Une matrice est un tableau à deux dimensions, pour un jeu comme le touché coulé, elle est carrée avec une dimension de 10 de largeur.

```
private char [,] matrice;
```

SlotType :
Définissez des valeurs pour l'enum `SlotType` qui correspondront à une case non valide, de l'eau, un case d'un bateau touchée et une case d'un bateau non touchée.

```
public Terrain(string path)
```

Ceci est le constructeur de la classe Terrain. Il serait peut etre intéressant ici, d'initialiser la matrice en chargeant le fichier représentant la map.

```
SlotType get_slot_type(char c)
```

Selon le caractère passé en argument, vous retournerez sa valeur de type SlotType.

```
public void display_full()
```

Cette fonction affiche la totalité de la map en console, ce qui devrait ressembler à cela :

```
   A B C D E F G H I J
1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2  ~ O ~ ~ ~ ~ ~ ~ ~ ~
3  ~ O ~ ~ ~ O ~ ~ ~ ~
4  ~ X ~ ~ ~ O ~ ~ ~ ~
5  ~ O ~ ~ ~ O ~ ~ ~ ~
6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7  ~ ~ O O ~ ~ ~ ~ ~ ~
8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
10 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

```
public void display_hidden();
```

Cette fois-ci, affichez la map mais en remplacant les `0` par des ~, c'est quand même mieux de ne pas afficher la position des bateaux en jeu ...

```
public void load_file(string path);
```

Chargez le fichier se trouvant dans path, lisez le caractère par caractère et remplissez la matrice en fonction.

```
public bool game_over();
```

On parcourt la matrice pour vérifier si le jeu est fini.

```
public bool touche(int x, int y);
```

Cette fonction vérifie si le bateau a été touché et met à jour la matrice en fonction.

```
public bool coule(int x, int y);
```

Cette fonction vérifie si un bateau a été coulé en partant de la position (x,y). Vous pouvez l'implémenter de la manière dont vous le souhaitez. La fonction coule_rec est un indice mais n'est en aucun cas obligatoire.

3.2 Player.cs

```
public Player(string name, string path);
```

Constructeur de la classe player, instancie le joueur en mettant à jour son nom et en instanciant la carte du joueur adverse.

```
public string get_name();
```

La variable contenant le nom du joueur est protegée (private), on crée donc un getter pour récuperer cette valeur.

```
public void show_board();
```

Affiche la carte de l'adversaire du player.

```
public bool play(int x, int y);
```

Dans cette fonction, vous appellerez les fonctions nécessaires pour mettre à jour la matrice, vérifier si vous avez touché un bateau adverse et même coulé!
Retourne si le joueur a gagné ou non.

3.3 Game.cs

```
public struct Point;
```

Structure représentant un point avec les coordonnées x et y.

```
public void game_loop();
```

Implémente la boucle de jeu, référez vous au squelette pour la remplir.

4 Bonus

Idées de bonus :
- Génération de terrain
- IA
- Interface Graphique (Facile avec XNA)
- Modélisation 3D (Moins facile, même avec XNA)
- Réseau (via SMS ? Certains l'ont bien fait pour les échecs :
http ://www.twilio.com/engineering/2012/11/08/adventures-in-unicode-sms)