Escape From WestWorld

Consignes de rendu

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

```
prenom.nom.zip
|-- prenom.nom/
    |-- AUTHORS
    |-- README
    |-- WarmUp/
        |-- WarmUp.sln
        |-- WarmUp/
            |-- Everything except bin/ and obj/
    |--EscapeFromWestWorld/
        |--EscapeFromWestWorld.sln
        |--EscapeFromWestWorld/
            |--Everything except bin/ and obj/
```

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

— Remplacez `prenom.nom` par votre propre login et n'oubliez pas le fichier `AUTHORS`.
— Les fichiers `AUTHORS` et `README` sont obligatoires.
— Pas de dossiers `bin` ou `obj` dans le projet.
— Respectez scrupuleusement les prototypes demandés.
— Retirez tous les tests de votre code.
— **Le code doit compiler !**

AUTHORS

Ce fichier doit contenir une ligne formatée comme il suit : une étoile (*), un espace, votre login et un retour à la ligne. Voici un exemple (où `$` est un retour à la ligne et ␣ un espace) :

```
*␣firstname.lastname$
```

Notez que le nom du fichier est `AUTHORS` sans extension. Pour créer simplement un fichier `AUTHORS` valide, vous pouvez taper la commande suivante dans un terminal :

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

README

Vous devez écrire dans ce fichier tout commentaire sur le TP, votre travail, ou plus généralement vos forces / faiblesses, vous devez lister et expliquer tous les boni que vous aurez implémentés. Un `README` vide sera considéré comme une archive invalide (malus).

1 Introduction

Lisez bien la totalité du sujet avant de commencer. Chaque fonction est évaluée indépendamment, si vous bloquez sur un exercice, n'hésitez pas à passer à la suite pour y revenir plus tard.

2 Warm Up

Les champs `size` testés seront toujours corrects.

2.1 Does it look like anything to you ?

Implémentez la fonction `HelloWestWorlds` qui affiche sur la sortie standard `n` fois le message suivant suivi d'un retour à la ligne :

```
Hello WestWorld!
```

```
1  public static void HelloWestWorlds(int n);
```

Si `n` est négatif, vous devrez lancer une `ArgumentException`.

2.2 GetAt

Implémentez la fonction `GetAt`, retournant la valeur contenue à l'indice `index` du tableau `tab`. En cas d'erreur, votre fonction doit lancer une `ArgumentException`.

```
1  public static int GetAt(int index, int[] tab, int size);
```

2.3 SetAt

Implémentez la fonction `SetAt`, remplacant la valeur contenue à l'indice `index` du tableau `tab` par la valeur `elm`. En cas d'erreur, vous devrez lancer une `ArgumentException`.

```
1  public static void SetAt(int index, int elm, int[] tab, int size);
```

2.4 Get Max

Implémentez la fonction `GetMax`, revoyant l'index de la valeur maximum du tableau `tab`. En cas d'erreur, vous devrez lancer une `ArgumentException`.

```
1  public static int GetMax(int[] tab, int size);
```

2.5 Sorted ?

Implémentez la fonction `IsSorted`, renvoyant vrai si le tableau `tab` est trié en décroissant. En cas d'erreur, vous devrez lancer une `ArgumentException`.

```
1  public static bool IsSorted(int[] tab, int size);
```

2.6 Sort

Implémentez la fonction `Sort` qui trie le tableau `tab` en ordre décroissant selon la méthode de votre choix.

```
1  public static void Sort(int[] tab, int size);
```

2.7 Print Array

Implémentez la fonction `PrintTab`, qui affiche le tableau `tab` sur la sortie standard. Les éléments sont suivis par un espace sauf le dernier, suivi d'un retour à la ligne. En cas d'erreur, vous devrez lancer une `ArgumentException`.

```
1  public static void PrintTab(int[] tab, int size);
```

2.8 Conway

Implémentez la fonction `PrintConway`, qui affiche le rang `n` de la suite de Conway sur la sortie standard. En cas d'erreur, vous devrez lancer une `ArgumentException`.

Chaque terme de la suite se construit en annonçant le terme précédent, c'est-à-dire en indiquant combien de fois chacun de ses chiffres se répète. Le premier terme est `1`, le second `11`, car il y a un 1. Puis `21` car il y a deux 1 à la suite etc.

```
1  public static void PrintConway(int n);
```

Voici un exemple, obtenu grâce à l'appel suivant :

```
1  public static int main()
2  {
3      for (int i = 0; i < 10; ++i)
4          PrintConway(i);
5      return 0;
6  }
```

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
```

3 Escape From WestWorld

WestWorld ouvre dans moins de 3 heures et un des scénarios n'est toujours pas prêt. Le scénario comporte une course poursuite au milieu du désert. Vous êtes responsable de la création de l'environnement. Au travail !

Le jeu consiste donc à diriger (grâce aux flèches directionnelles) une diligence sur un chemin à 3 voies, des obstacles apparaissent aléatoirement sur les voies. Si la diligence heurte un des obstacles, c'est perdu !

Une fois terminé, le jeu devrait ressembler à ceci :

```
|###|   |   |
|###|   |   |
|   |   |###|
|   |   |###|
|   |   |   |
|   |   |   |
|   |   |###|    // Un obstacle
|   |   |###|
|   |###|   |
|   |###|   |
|   |   |   |
|   |   |   |
|   |   |###|
|   |   |###|
|###|   |   |
|###|O|O|   |
|   | | |   |     // La carriole
|   |O-O|   |
```

3.1 Architecture du jeu

Le jeu est divisé en différentes classes, ayant chacune une utilité bien spécifique :

La classe `Game` fait office de *game manager*. Elle possède 6 attributs privés :
- `BoardMaxLines` : Le nombre maximal de lignes du plateau. Vous ne devez pas modifier sa valeur par défaut.
- `BoardMaxColumns` : Le nombre maximal de colonnes du plateau. Vous ne devez pas modifier sa valeur par défaut.
- `board` : Tableau de caractères à deux dimensions représentant le plateau.
- `car` : Un objet de type `Car`, représentant la carriole.
- `rocks` : Une file d'objets `Rock`, représentant les obstacles qui apparaîtront sur le plateau.
- `count` : Entier représentant

Les classes `Car` et `Rock` héritent de la classe abstraite `GameObject`. Ces 2 classes possèdent des attributs indiquant leur position. `column` pour `Car`, et `column` et `line` pour `Rock`.

Les objets de classe `Car` ou `Rock` peuvent être affichés sur le plateau (placés dans le tableau `board`).

Nous vous conseillons de lire tout le code fourni avant de commencer à coder.

3.2 Game

3.2.1 Constructeur

Implémentez le constructeur de la classe `Game`, initialisant tous ses attributs.

```
1 public Game();
```

3.2.2 Loop

Implémentez la méthode `Loop`, qui appelle la méthode `SetBoard`, affiche le plateau et rajoute des obstacles tant que le jeu n'est pas terminé.

```
1 public void Loop();
```

3.2.3 End

Implémentez la méthode `End`, qui clear la console, puis affiche le message suivant sur la sortie standard : "Game Over".

```
1 public void End();
```

3.2.4 EmptyBoard

Implémentez la méthode `EmptyBoard`, qui initialise un tableau de jeu vide, sans carriole ni obstacle.

```
1 public void EmptyBoard();
```

3.2.5 SetBoard

Implémentez la méthode `SetBoard`, qui rajoute les obstacles et la carriole sur le plateau.

```
1 public void SetBoard();
```

3.2.6 PrintBoard

Implémentez la méthode `PrintBoard`, qui affiche le plateau sur la sortie standard.

```
1 public void PrintBoard();
```

3.2.7 MoveObjects

Implémentez la méthode `MoveObjects`, qui déplace la voiture et les roches. Attention à ne pas oublier de détruire les roches ayant quitté le plateau.

```
1 public void MoveObjects();
```

3.2.8 IsOver

Implémentez la méthode `IsOver`, qui retourne vrai si le jeu est terminé, c'est à dire si une collision a lieu entre un obstacle et la carriole.

```
1 public bool IsEnd();
```

3.2.9 SpawnRock

Implémentez la méthode `SpawnRock`, qui crée un nouvel obstacle dnas une colonne aléatoire, puis l'enfile dans la file d'obstacles.

```
1 public void SpawnRock();
```

3.3 Car

3.3.1 Print

Implémentez la méthode `Print`, qui ajoute la voiture sur le plateau. Attention à bien respecter l'exemple !

```
1 public void Print(char[,] board);
```

3.3.2 Move

Implémentez la méthode `Move`, qui déplace la carriole. Celle-ci ne peut que se déplacer vers la droite ou la gauche.

```
1 public void Move();
```

3.4 Rock

3.4.1 Constructeur

Implémentez le constructeur de la classe `Rock`, qui se contente d'attribuer la valeur 17 à l'attribut `line`.

```
1 public Rock(Column column);
```

3.4.2 Print

Implémentez la méthode `Print`, qui ajoute la roche sur le plateau. Attention à bien respecter l'exemple et les cas où la roche n'est pas entièrement sur le plateau !

```
1 public void Print(char[,] board);
```

3.4.3 Move

Implémentez la méthode `Move`, qui déplace la roche d'une ligne. La méthode retourne vrai si le nouvel attribut `line` est supérieur ou égal à -1.

```
1 public bool Move();
```

3.5 GameObject

3.5.1 Constructeur

Implémentez le constructeur de la classe `GameObject`, qui initialise l'attribut `column` de la classe.

```
1  protected GameObject(Column column);
```

3.5.2 Setter

Implémentez un setter pour l'attribut `column`.

3.6 Fonctions autorisées

Vous avez accès à l'ensemble des méthodes de la classe `Queue`. Vous aurez sans doute besoin des méthodes `Enqueue` et `Dequeue` qui respectivement, enfilent et défilent un élément de la file.

These violent deadlines have violent ends.