M2104 - M2107 : projet

Le rhum de Guybrush

1 Le problème à traiter

Nous sommes à la grande époque de la piraterie.

Lorsqu'il était haut comme trois pommes, Guybrush Threepwood rêvait de devenir pirate. Devenu grand, il réalisa donc son rêve mais ses démêlés avec le pirate fantôme LeChuck, qu'il vainquit pourtant, lui firent comprendre les risques du métier.

Il décida donc de se reconvertir, avec sa fiancée Elaine, au métier plus calme de bouilleur de rhum.

Pour faire du rhum, il faut de la canne à sucre. Guybrush part donc explorer les îles des Caraïbes afin de trouver des parcelles où en planter. Dès qu'une île est découverte, Guybrush en dresse le plan et le confie à un perroquet savant qui le porte à Elaine, restée à la maison.

Malheureusement, le fidèle second du pirate fantôme LeChuck, Largo LaGrande, qui voue une haine profonde à Guybrush, a eu vent du projet et décide de le concurrencer en distillant lui aussi du rhum. Son idée est d'intercepter le perroquet savant pour récupérer les plans des îles découvertes par Guybrush afin d'y planter avant tout le monde sa canne à sucre.

Guybrush et Elaine décident donc d'inventer un système de codage et de décodage des cartes : avant d'attacher un plan de carte à la patte du perroquet, Guybrush le codera avec le système présenté ci-dessous. Elaine réceptionnera la carte ainsi chiffrée et la décodera. Si le perroquet est capturé par Largo LaGrande, celui-ci récupérera une carte incompréhensible puisque codée.

Guybrush et Elaine souhaitent informatiser ce système de codage/décodage. C'est **votre équipe** qui est chargée de la conception de ce logiciel !

2 La structure d'une île

Un île est constituée de plusieurs **parcelles**.

Chaque parcelle est constituée de plusieurs zones carrées appelées **unité**. On donne figure 1 des exemples de parcelles de taille 4.

FIGURE 1 – Quelques exemples de parcelles de taille 4

Une île peut également contenir des blocs de type **Mer** ou **Forêt**, composés d'unités sur lesquelles il est impossible de planter de la canne à sucre. Un bloc **Mer** peut se situer à l'intérieur d'une l'île, auquel cas

il s'agit d'un lac.

Une île s'inscrit toujours dans une grille de taille 10 unités × 10 unités.

On donne figure 2 un exemple de carte valide, qui représente l'île de Scabb.

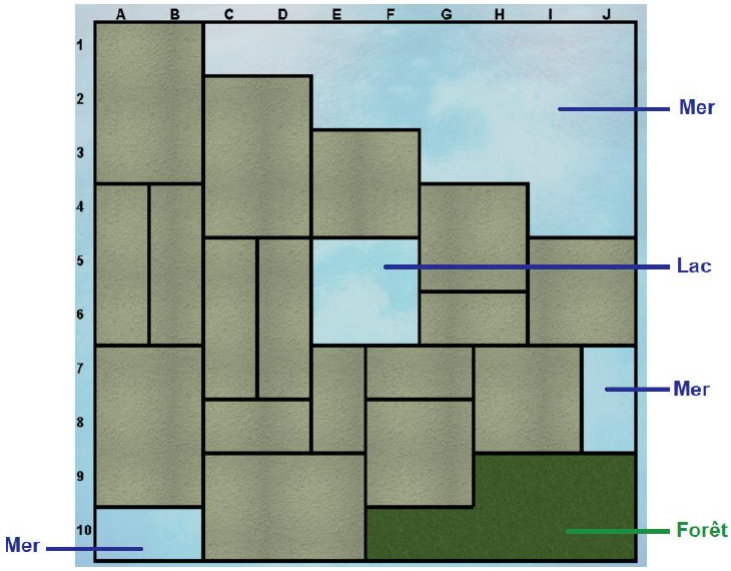

FIGURE 2 – Un exemple d'île : Scabb Island

2.1 Codage d'une carte

Chaque unité est représentée par un nombre entier compris entre 0 et 15. Si ce nombre vaut zéro, cela signifie que l'unité ne possède aucune frontière avec ses quatre voisines. Sinon ce nombre est une combinaison de puissances de 2 et chaque puissance indique qu'il existe une frontière sur un des cotés de l'unité (figure 3) :

- 2^0 : il y a une frontière au nord
- 2^1 : il y a une frontière à l'ouest
- 2^2 : il y a une frontière au sud
- 2^3 : il y a une frontière à l'est

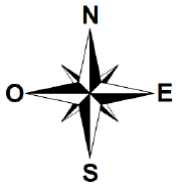

FIGURE 3 – Les quatre points cardinaux

Par exemple, une unité représentée par le nombre 9 possède une frontière au nord et à l'est car $9 = 2^3+2^0$.

Pour différencier les unités de type Mer ou Forêt, pour lesquelles la règle ci-dessus s'applique, on ajoute respectivement 64 et 32 à la combinaison des puissances de 2. Par exemple, une unité représentée par le nombre 70 est de type **Mer** et comporte une frontière à l'ouest et au sud car $70 = 64 + 2^1 + 2^2$.

Ces nombres sont contenus dans une trame dans laquelle ils sont séparés par le caractère « : » tandis que le caractère « | » introduit une nouvelle ligne de la carte.

Ainsi, la trame suivante est correcte et représente la carte de Scabb Island montrée figure 4 (chaque lettre minuscule identifie une parcelle).

```
3:9:71:69:65:65:65:65:65:73|2:8:3:9:70:68:64:64:64:72|6:12:2:8:3:9:70:68:64:72|11:11:6:
12:6:12:3:9:70:76|10:10:11:11:67:73:6:12:3:9|14:14:10:10:70:76:7:13:6:12|3:9:14:14:11:7:
13:3:9:75|2:8:7:13:14:3:9:6:12:78|6:12:3:1:9:6:12:35:33:41|71:77:6:4:12:39:37:36:36:44|
```

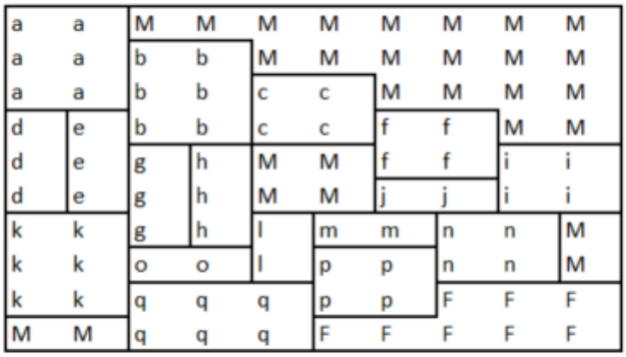

FIGURE 4 – La carte correspondant à la trame donnée en exemple.

C'est cette trame que le perroquet transporte et fournit à Elaine.

2.2 Décodage de la trame

A la réception, Elaine décode la trame pour obtenir une carte claire comme celle montrée figure 4.

3 Les fonctionnalités du logiciel

3.1 Codage d'une carte claire

Le logiciel doit permettre de coder une carte claire afin d'obtenir une carte chiffrée.

La carte claire est contenue dans un fichier texte de dix lignes à l'extension « point clair ». Chaque ligne comporte dix caractères représentant les dix unités de cette ligne.

Les unités d'un même parcelle sont représentées par un lettre minuscule (à partir de *a*). Les unités de type Mer sont représentées par un **M** majuscule et celles de type Foret par un **F** majuscule.

Par exemple, la fgure 5 indique le contenu du fichier texte *Scabb.clair* rédigé par Guybrush et représentant Scabb Island.

La trame chiffrée correspondante est écrite dans un fichier qui possède le même nom que la carte claire, mais avec l'extension « point chiffre ». Ainsi, la carte claire contenue dans *Scabb.clair* est codée par une trame écrite dans *Scabb.chiffre*, lequel fichier sera fourni au perroquet savant pour qu'il le porte à Elaine.

Ainsi, pour Scabb Island, à partir du fichier texte *Scabb.clair*, le logiciel produit le fichier texte *Scabb.chiffre* dont le contenu est montré figure 6.

```
aaMMMMMMMM
aabbMMMMMM
aabbccMMMM
debbccffMM
deghMMffii
deghMMjjii
kkghlmmnnM
kkoolppnnM
kkqqqppFFF
MMqqqFFFFF
```

FIGURE 5 – Le fichier *Scabb.clair*, représentant l'île de Scabb.

```
3:9:71:69:65:65:65:65:65:73|2:8:3:9:70:68:64:64:64:72|6:12:2:8:3:9:70:68:64:72|11:11:
6:12:6:12:3:9:70:76|10:10:11:11:67:73:6:12:3:9|14:14:10:10:70:76:7:13:6:12|3:9:14:14:
11:7:13:3:9:75|2:8:7:13:14:3:9:6:12:78|6:12:3:1:9:6:12:35:33:41|71:77:6:4:12:39:37:
36:36:44|
```

FIGURE 6 – Le fichier *Scabb.chiffre*, représentant l'île de Scabb chiffrée.

Indications de programmation

Pour utiliser les méthodes .Net liées à la gestion des fichiers, il faut placer en début de fichier source la ligne suivante :

```
using System.IO;
```

Les objets de la classe *StreamReader* et *StreamWriter* permettent respectivement de lire et d'écrire dans un fichier texte. La procédure est la suivante :

- ouverture du fichier avec le constructeur
- lecture ou écriture dans le fichier (méthodes *ReadLine* et *WriteLine*)
- fermeture du fichier et libération des ressources associées (méthode *Close*)

Les méthodes statiques *Path.GetFileName* et *Path.GetDirectoryName* permettent d'extraire d'un chemin d'accès à un fichier respectivement le nom du fichier et le nom du dossier qui contient ce fichier.

La méthode *Split* de la classe *string* permet de découper une chaîne de caractères suivant un séparateur donné.

Ne pas oublier de gérer les éventuelles erreurs d'exécution (fichier inexistant ou problème de droit). Consulter l'annexe **C**.

3.2 Décodage d'une carte chiffrée

Le logiciel doit permettre de lire une carte chiffrée contenue dans un fichier « point chiffre »pour la charger en mémoire.

La trame correspondant à la carte chiffrée est contenue dans un fichier texte d'une ligne à l'extension « point chiffre ».

Indication informatique En C#, un caractère est représenté par son code ASCII. Par exemple, la ligne de code suivante est correcte :

```
int raoul = 'a' - 8;
```

L'entier *raoul* contient 89 car le code ASCII du caractère *a* minuscule vaut 97.

3.3 Affichage de la carte de l'île

Le logiciel doit permettre d'afficher la carte d'une île.

Les unités d'une même parcelle seront représentées par une même lettre de l'alphabet et affichées en blanc. Les blocs de type Mer seront représentés par un M majuscule de couleur bleue et les blocs de type Forêt par un F majuscule de couleur verte.

La figure 7 présente l'affichage obtenu pour Scabb Island.

```
a a M M M M M M M M
a a b b M M M M M M
a a b b c c M M M M
d e b b c c f f M M
d e g h M M f f i i
d e g h M M j j i i
k k g h l m m n n M
k k o o l p p n n M
k k q q q p p F F F
M M q q q F F F F F
```

FIGURE 7 – Rendu à l'écran de l'affichage de Scabb Island

Indication informatique La couleur d'écriture sur la console est définie ou obtenue avec l'accesseur *Console.ForegroundColor*. Les couleurs d'affichage disponibles sont contenues dans l'énumération *ConsoleColor*.

3.4 Informations sur une carte

3.4.1 Affichage de la liste des parcelles

Ecrire une méthode qui affiche la liste des parcelles, avec le nombre d'unités qui les composent et les coordonnées de ces unités.

Exemple de sortie écran pour Scabb Island :

```
PARCELLE a - 6 unites
(0,0)  (0,1)  (1,0)  (1,1)  (2,0)  (2,1)

PARCELLE b - 6 unites
```

```
(1,2)  (1,3)  (2,2)  (2,3)  (3,2)  (3,3)

PARCELLE c - 4 unites
(2,4)  (2,5)  (3,4)  (3,5)

...
```

3.4.2 Taille d'une parcelle

Écrire une méthode qui retourne la taille (c'est-à-dire le nombre d'unités) de la parcelle passée en paramètre.

Exemple de sortie pour la parcelle *s* (qui n'existe pas) de Scabb Island :

```
Parcelle s : inexistante
Taille de la parcelle s : 0 unites
```

Exemple de sortie pour la parcelle *b* de Scabb Island :

```
Taille de la parcelle b : 6 unites
```

3.4.3 Affichage des parcelles dont la taille est supérieure à une borne donnée

Écrire une méthode qui affiche les parcelles dont la taille est supérieure ou égale à une taille donnée.

Exemple de sortie écran avec une borne de 4 pour Scabb Island :

```
Parcelles de taille supérieure à 4 :
Parcelle a: 6 unites
Parcelle b: 6 unites
Parcelle c: 4 unites
Parcelle f: 4 unites
Parcelle i: 4 unites
Parcelle k: 6 unites
Parcelle n: 4 unites
Parcelle p: 4 unites
Parcelle q: 6 unites
```

Exemple de sortie écran avec une borne de 12 pour Scabb Island :

```
Parcelles de taille supérieure à 12 :
Aucune parcelle
```

3.4.4 Taille moyenne des parcelles

Écrire une méthode qui retourne la taille moyenne des parcelles. Le résultat est arrondi à deux chiffres après la virgule.

Exemple de sortie écran pour Scabb Island :

```
Aire moyenne : 3,76
```

4 Modalités de réalisation du projet

4.1 Configuration

Ce projet est à réaliser en équipe de quatre étudiants maximum. Le langage de programmation imposé est C#. Une modélisation objet est requise. Un outil de versionnage de type Git devra être utilisé.

4.2 ENT

Une section dédiée à ce projet tutoré est présente sur l'ENT de la matière M2104. On pourra y trouver :
- des exemples de cartes claires ou chiffrées
- des renseignements supplémentaires
- les dates cruciales liées au projet
- les espaces de téléversement des éléments notés du projet

Un espace de discussion sera par ailleurs créé sur le Discord du département.

4.3 L'évaluation

L'évaluation portera sur les éléments suivants :
- Le code source du projet. Les critères suivants seront pris en compte :
  - La décomposition judicieuse en classes/méthodes/sous-méthodes
  - Le respect des principes de programmation objet et des bonnes pratiques de programmation (chapitre 8 du cours de M2104)
  - La clarté et la lisibilité du programme (commentaires pertinents, variables bien nommées, indentation correcte...)
  - Le versionnage régulier du programme sur le dépôt Git par **tous** les membres de l'équipe
- La documentation du projet, obtenue à partir de la documentation du code source et mise en forme avec le logiciel *Doxygen*, sous forme d'un fichier *pdf* ou d'un site web en ligne sur Internet.
- Un rapport au format *pdf* qui contiendra :
  - une page de garde complète. Seuls les étudiants figurant sur cette page seront notés
  - une introduction
  - une explication succincte de la modélisation objet retenue
  - une conclusion, qui intégrera les apports de ce projet, les difficultés rencontrées et les éventuels axes d'amélioration du programme
  - un lien vers le dépôt *git* du projet, afin que votre enseignant puisse tester le projet

La note finale de chaque membre de l'équipe sera obtenue comme suit :
- Le projet obtiendra une note globale sur 20
- Cette note sera multipliée par le nombre de membres de l'équipe
- Les membres de l'équipe se partageront ce total de points en fonction de l'investissement de chacun.

Exemple Prenons le cas d'un projet réalisé en quadrinôme et noté 14.5/20. Les quatre membres de l'équipe se partageront un total de $4 \times 14.5 = 58$ points. Une répartition correcte peut être :

- Développeur 1 : 16/20
- Développeur 2 : 16/20
- Développeur 3 : 14/20
- Développeur 4 : 12/20

La note de ce projet comptera dans le module **M2107** « projet tutoré du semestre 2 ».

Attention Le partage de code à la manière des logiciels libres est très noble. Cependant, dans le cas d'un travail noté comme ce projet, cela s'apparente à une action de fraude, comme si un étudiant en salle d'examens passait sa copie à son voisin. Les codes identiques seront donc sanctionnés.

M2104 - M2107 : projet

Annexes

A Création du dépôt distant Git à partir d'un dépôt local

Un dépôt local contient un projet donné. Le but de cet section est de le pousser sur un dépôt distant, situé sur Internet.

Dans un premier temps, on demande à Git de créer un lien vers le dépôt sur le serveur distant (GitLab dans le cas ci-dessous) à l'aide de la commande *git remote add* :

```
$ git remote add origin https://gitlab.com/VotreIdentifiant/NomDuDepot
```

— *origin* est le nom du lien vers le dépôt distant. Ce nom peut être choisi librement

Ceci étant fait, on peut pousser le projet du dépôt local vers le dépôt distant avec *git push* :

```
$ git push -u origin master
```

On peut vérifier avec l'interface web de GitLab que le projet et son historique ont bien été poussé vers le serveur distant (Menu *Repository* à droite, onglets *Files* et *Commits*).

→ Maintenant il est possible à tous les membres du projet d'en créer une copie locale (un clone) – si l'autorisation leur a été donnée bien entendu !

B Documentation technique : le logiciel libre Doxygen

Doxygen est un générateur de documentation sous licence libre capable de produire une documentation logicielle à partir du code source d'un programme. Pour le cas du langage C#, il examine tous les fichiers source et repère la documentation introduite par les triples slashes « `///` ».

Le site officiel de *Doxygen* est : `https://www.doxygen.nl/index.html`.

C La gestion des exceptions

Une **exception** désigne une erreur pouvant se produire lors de l'exécution d'un programme. En C# il existe deux sources d'exception :

— les exceptions systèmes : division par zéro, accès en dehors d'un tableau, utilisation d'une référence *null*. Ces exceptions sont le résultat d'une erreur de programmation (bug).

— les exceptions générées volontairement par le programmeur pour signaler un problème.

C.1 Gérer une exception

Pour gérer une exception, on utilise le bloc *try... catch*. Le bloc *try* (essayer) comporte le code susceptible de générer des exceptions et le bloc *catch* (attraper) contient le code de gestion des exceptions.

```
int a, b, c;
...
try
{
  a = b / c;  // erreur de division par 0 si c vaut 0
}
catch (Exception e)
{
  Console.WriteLine("Erreur! {0}", e.Message);
}
```

Le bloc *catch* récupère un objet de type *Exception*, dont l'attribut *Message* décrit l'erreur survenue.

Il existe différentes classes d'exceptions qui héritent de la classe *Exception* et il est possible d'affecter des blocs de traitement différents pour chaque exception.

```
int a, b, c;
int [] t;
...
try
{
  t[a] = b / c;
}
catch (AritmeticException e)
{
  Console.WriteLine("Erreur sur une opération arithmétique! {0}", e.Message);
}
catch (IndexOutOfRangeException e)
{
  Console.WriteLine("Accès en dehors des bornes d'un tableau! {0}", e.Message);
}
catch (Exception)
{
  Console.WriteLine("Erreur!");
}
```

Il y a donc des exceptions plus ou moins spécialisées, ce qui permet de ne gérer que certaines exceptions. Le type *Exception* désigne n'importe quelle exception et attrapera donc toutes les exceptions possibles.

C.2 Bloc finally

Chaque bloc *try* peut être associé à un bloc *finally*. Les instructions placées dans le bloc *finally* seront exécutées même si une exception se produit dans le bloc *try*, que cette exception soit gérée ou non. Ce

bloc est très utile pour gérer la libération des ressources tenues. Par exemple :

```
StreamWriter sw;    // pour écrire dans un fichier
try
{
  sw = new StreamWriter("monfichier.txt");
  sw.WriteLine("toto");
}
catch (Exception e)
{
  Console.WriteLine("Erreur : {0}", e.Message);
}
finally
{
  // ce bloc est toujours exécuté : c'est le bon endroit pour fermer le fichier !
  // si sw vaut null, c'est que le fichier n'a jamais été ouvert
  if (sw != null)
    sw.Close(); // fermeture du fichier
}
```