TME 2 : Solutions des mots croisés

Objectifs pédagogiques :

- **interfaces**
- **itérations**
- **structures de données**

2.1 Introduction

Ce TME fait suite au précédent. Nous continuons à travailler sur notre projet `MotCroise` à l'aide d'Eclipse et de GitLab. À la différence de la semaine dernière, toutes les classes ajoutées cette semaine seront placées dans le package `pobj.motx.tme2`.

> **Synchronisation avec le serveur GitLab**
> Depuis le dernier TME, vous avez probablement apporté des modifications au projet sur le serveur GitLab. Avant toute chose, il est nécessaire de synchroniser la copie locale du projet avec cette version plus récente. **Commencez la séance par un *pull*.** En travaillant à partir des dernières versions des fichiers, vous vous éviterez des conflits inutiles entre versions. Pour rappel, la marche à suivre est détaillée dans le document *Mise en place et rendu des TME*.

Nous nous sommes dotés au TME 1 d'une grille et d'une notion d'emplacement de mot. Nous allons dans un premier temps extraire la liste des emplacements de mot d'une grille. Nous allons ensuite définir des dictionnaires pour maintenir des ensembles de mots français (i.e., des chaînes). Chaque emplacement aura son propre dictionnaire, appelé *domaine potentiel*, qui détermine l'ensemble des mots qui peuvent remplir les cases de l'emplacement. Le domaine potentiel d'un emplacement sera ensuite filtré à partir de contraintes spécifiques à cet emplacement. Ainsi, la taille (nombre de cases) de l'emplacement contraint la taille des mots de son domaine potentiel. Par ailleurs, s'il y a des lettres déjà placées sur la grille, nous filtrons le domaine potentiel des emplacements de mot contenant une case avec une lettre fixée pour ne garder que les mots qui ont la lettre demandée à la position donnée. Lors du TME suivant, nous ajouterons également des contraintes liées aux croisements de mots sur la grille, et verrons comment en tenir compte pour réduire encore les domaines potentiels.

Le domaine potentiel peut être restreint à un singleton (un seul mot) si toutes les lettres sont déjà placées, ou même être vide s'il n'y a pas de mot du dictionnaire français qui croise les lettres déjà placées. Les mots croisés sont *irréalisables* si le domaine potentiel d'au moins un emplacement est vide (aucune solution). Ils sont *résolus* si tous les domaines potentiels sont réduits à des singletons (i.e., la grille est remplie et tous les mots sont dans le dictionnaire). Dans cette séance nous allons donc nous donner le moyen de déterminer les emplacements et calculer leurs domaines potentiels. La résolution à proprement parler des mots croisés utilisera cette information. Elle fera l'objet du TME 3.

2.2 Classe `GrillePlaces`

La classe `GrillePlaces` doit explorer une `Grille` pour trouver tous les emplacements des mots qu'elle contient. Rappelons qu'un emplacement de mot est une séquence contiguë de cases, soit horizontale soit verticale, contenant au moins deux cases.

Nous imposons l'ordre des mots au sein d'une `GrillePlaces`, afin de permettre l'exécution des tests : nous trouvons d'abord les mots *horizontaux*, triés par numéro de ligne croissant. Notons qu'il peut très bien y avoir plusieurs mots sur une ligne (contrairement à l'exemple fourni en préambule au TME 1). À numéro de ligne égal, nous rencontrons les mots par colonne de la première case croissante, i.e., nous lisons de gauche à droite. Ensuite, nous trouvons les mots verticaux, triés par colonne croissante, et à colonne égale par numéro de ligne de la première case des mots croisés.

Rappelons par exemple que la grille donnée en préambule au TME 1 comporte au total 12 mots, dont 7 sont horizontaux. La numérotation des cases sur la figure de la première case de chaque mot reflète l'ordre dans lequel nous devons rencontrer les mots dans `GrillePlaces` (0 à 6 pour les mots horizontaux et 7 à 11 pour les mots verticaux).

⇒ **Donnez l'implantation de la classe `GrillePlaces`**, elle comportera au minimum :

- un constructeur `public GrillePlaces (Grille grille)` qui explore la grille fournie et calcule les emplacements de mot qu'elle contient ; nous pourrons stocker les emplacements trouvés dans un attribut de la classe `private List<Emplacement> places`,
- un accesseur `public List<Emplacement> getPlaces()` retournant les emplacements détectés,
- un accesseur `public int getNbHorizontal()` pour obtenir le nombre d'emplacements horizontaux (`getPlaces().size()` permet déjà d'obtenir le nombre total d'emplacements détectés),
- une méthode standard `public String toString()` qui permet d'afficher les emplacements de mot détectés de façon lisible.

Sans que ce soit imposé, pour factoriser le code de détection de mots, nous suggérons de :

- définir deux méthodes `private List<Case> getLig (int lig), private List<Case> getCol (int col)` qui retournent les cases qui constituent une ligne ou une colonne donnée (sans les copier),
- écrire une méthode `private void cherchePlaces(List<Case> cases)` qui cherche les emplacements dans la liste de cases fournie (une ligne ou une colonne) et qui les ajoute à la liste. Un emplacement de mot est défini dès que nous avons deux cases contiguës non pleines (donc vides ou avec une lettre déjà placée),
- écrire le constructeur `public GrillePlaces (Grille grille)` qui consiste alors à itérer sur les lignes, chercher les emplacements de mot, noter le nombre d'emplacements horizontaux détectés, puis itérer sur les colonnes en cherchant les emplacements verticaux.

Pour écrire `cherchePlaces`, nous pouvons utiliser un `Emplacement` initialement vide. À chaque case rencontrée :

- si elle est non pleine, nous l'ajoutons à l'emplacement,
- sinon, nous examinons la taille de l'emplacement construit,
  - s'il fait au moins deux cases, nous l'ajoutons aux emplacements de mot détectés,
  - dans tous les cas, nous réinitialisons l'emplacement.

N'oubliez pas (à la fin de l'itération) d'ajouter le dernier emplacement trouvé s'il est assez long.

⇒ **Exécutez le jeu de test `pobj.motx.tme2.test.GrillePlacesTest` fourni.**

2.3 Classe `Dictionnaire`

Un dictionnaire est nécessaire pour notre application, il définit l'ensemble des mots *légaux* trouvés dans une grille. Nous considérons qu'un dictionnaire n'est rien d'autre qu'une liste de mots, représentés par des `String`. Nous allons nous donner des méthodes qui permettent de restreindre le dictionnaire aux mots qui satisfont un critère donné : la taille du mot ou la présence d'une lettre donnée à une certaine position.

Nous fournissons l'implantation d'une classe `pobj.motx.tme2.Dictionnaire`, elle comporte :

- un attribut `private List<String> mots` qui stocke une liste de mots,
- une méthode `public void add(String mot)` qui ajoute un mot au dictionnaire,
- une méthode `public int size()` qui retourne la taille (nombre de mots) du dictionnaire,
- une méthode `public String get(int i)` qui retourne le *i-ième* mot du dictionnaire (pour $i <$ `size()`),
- une méthode `public Dictionnaire copy ()` qui retourne une copie du dictionnaire courant,
- une méthode `public int filtreLongueur(int len)` qui modifie le dictionnaire pour ne garder

que les mots de longueur *len*. Cette méthode retourne le nombre de mots qui ont été supprimés du dictionnaire.

Nous ajouterons d'autres méthodes de filtrage sur le modèle de `filtreParLongueur` au fil du TME, par exemple pour trouver les mots qui ont un *a* en troisième lettre.

⇒ **Ajoutez une méthode `public static Dictionnaire loadDictionnaire(String path)`**, elle chargera un dictionnaire depuis un fichier texte. Nous supposons que le fichier d'entrée comporte exactement un mot par ligne.

Pour tester le comportement, nous vous fournissons un dictionnaire dans le fichier `data/frgut.txt`, qui a déjà été traité pour se limiter aux 26 lettres (pas d'accents, espaces, traits d'union...) et qui contient plus de 300000 mots.

⇒ **Exécutez le jeu de test `pobj.motx.tme2.test.DictionnaireTest` fourni**, qui charge `data/frgut.txt` et calcule le nombre de mots de chaque longueur.

Lecture d'un fichier. Il y a plusieurs façons de lire un fichier ligne par ligne en Java. Nous proposons la structure de code relativement simple suivante, où `path` est un nom de fichier. À chaque itération de la boucle *foreach*, la variable `line` chaîne contient une nouvelle ligne du fichier.

```
// Try-with-resource : cette syntaxe permet d'accéder au contenu du fichier ligne par ligne.
try (BufferedReader br = new BufferedReader(new FileReader(path))) {
   for (String line = br.readLine() ; line != null ; line = br.readLine() ) {
      // Utiliser "line".
   }
} catch (IOException e) {
  // Problème d'accès au fichier.
  e.printStackTrace();
}
```

2.4 Classe `GrillePotentiel`

La classe `GrillePotentiel` enrichit la classe `GrillePlace` en associant à chaque emplacement de mot un dictionnaire : son domaine potentiel. Cette classe possédera des méthodes pour manipuler les domaines potentiels. Elle sera enrichie au TME suivant (par héritage) pour tenir compte des croisements.

2.4.1 La base : un dictionnaire par emplacement de mot

Pour cette question, nous nous limitons à des mots croisés sans croisement (!), sur une grille complètement vide (pas de mot déjà placé).

⇒ **Donnez l'implantation de la classe `GrillePotentiel`**, elle comportera au minimum :

- un attribut de type `GrillePlaces` qui stocke la grille actuelle (partiellement remplie) ;
- un attribut de type `Dictionnaire` qui stocke le dictionnaire français complet ;
- un attribut `motsPot` de type `List<Dictionnaire>` qui stocke le domaine de chaque emplacement de la grille, dans le même ordre que les emplacements `places` dans `GrillePlaces`, ainsi qu'un assesseur `getMotsPot` ;
- un constructeur `public GrillePotentiel(GrillePlaces grille, Dictionnaire dicoComplet)` qui initialise les attributs aux valeurs données ; ensuite, il doit initialiser le domaine des emplacements ; commencez par simplement limiter les mots en filtrant le dictionnaire par longueur ;

- une méthode `public boolean isDead()` qui retourne vrai si et seulement si au moins un emplacement a un domaine potentiel vide.

⇒ **Exécutez le test `pobj.motx.tme2.test.GrillePotentielTest` fourni**, il testera une grille `split.grl` vide sans croisement.

2.4.2 Lettres placées

Nous supposons à présent que certaines cases peuvent avoir un contenu.

⇒ **Ajoutez dans la classe `Dictionnaire` une méthode `public int filtreParLettre(char c, int i)`**. Elle modifiera le dictionnaire pour ne garder que les mots dont la $i^{\text{ème}}$ lettre est égale au caractère de l'argument c. La méthode retourne le nombre de mots qui ont été supprimés du dictionnaire. Exécutez les tests fournis dans `pobj.motx.tme2.test.DictionnaireTest2`.

NB : Pour accéder au $i^{\text{ème}}$ caractère d'une `String s`, nous utilisons `s.charAt(i)`.

⇒ **À l'aide de cette méthode, raffinez pendant la construction de `GrillePotentiel` le domaine potentiel des emplacements de mot pour respecter les lettres déjà placées.** Nous pourrons itérer sur les cases (classe `Case`) constituant l'emplacement de mot pour voir si elles ont un contenu.

⇒ **Exécutez le test `pobj.motx.tme2.test.GrillePotentielTest2` fourni**, il testera une grille `easy2.grl` avec quelques lettres placées.

2.4.3 Placer un mot

Lors de la résolution des mots croisés, nous allons fixer la valeur de certains mots à un candidat donné, pris dans leur domaine potentiel. Cependant, nous allons avoir besoin d'explorer plusieurs possibilités, car ces affectations ne produisent pas toutes des grilles correctes. Pour préparer ce travail, nous proposons de développer la classe `GrillePotentiel` pour pouvoir fixer un mot dans un emplacement *sans modifier l'instance courante de la grille* : l'affectation doit donc retourner une nouvelle grille (une copie) qui diffère de la grille originale sur laquelle l'affectation est invoquée par le fait qu'un mot de plus y est placé. Ainsi, l'instance originale, sans le mot fixé, restera disponible pour explorer d'autres possibilités[1].

Nous rappelons que la classe `Grille` du TME 1 comporte une méthode `public Grille copy()` qui retourne une copie à l'identique de la grille courante.

⇒ **Ajoutez dans la classe `GrillePlaces`, une méthode `public GrillePlaces fixer(int m, String soluce)`**, elle retournera une *nouvelle* grille où les cases constituant l'emplacement de mot d'indice m (dans la liste des emplacements de mot de la grille telle que retournée par `getPlaces()`) ont pour contenu les lettres de `soluce`. Nous commencerons par faire une copie de la `Grille` stockée, que nous modifierons à l'aide de `setChar(char c)` sur les cases adaptées. Nous retournons enfin une nouvelle `GrillePlaces` initialisée à partir de la nouvelle grille.

⇒ **Ajoutez dans la classe `GrillePotentiel`, une méthode `public GrillePotentiel fixer(int m, String soluce)`**, elle initialisera une nouvelle `GrillePotentiel` avec la grille résultant de l'affectation.

⇒ **Exécutez le test `pobj.motx.tme2.test.GrillePotentielTest3` fourni**, il construira la grille du précédent test en partant d'une grille vierge.

2.5 Rendu de TME (OBLIGATOIRE)

N'oubliez pas de faire un rendu de TME en fin de séance, et éventuellement un deuxième rendu avant la prochaine séance si vous n'avez pas fini ce TME.

[1] Ces copies induisent une perte d'efficacité. Nous proposerons des améliorations en « bonus » au TME 3 pour minimiser ce problème.

Vous suivrez les mêmes instructions que la semaine dernière : vous propagerez vos dernières modifications locales vers le serveur GitLab, toujours sur le projet `MotCroise` privé à votre binôme et à vous ; vous créerez ensuite une *release* avec pour nom de *tag* « rendu-initial-tme2 » (ou « rendu-final-tme2 ») et vous répondrez aux questions ci-dessous dans le champ « Release notes ».

Vous vous assurerez cette fois que tous les tests unitaires des TME 1 et 2 passent correctement dans l'onglet d'intégration continue « Build > Pipelines ».

⇒ **Répondez aux questions suivantes dans votre rendu :**

- Pourquoi dans la méthode `copy` du dictionnaire est-il inutile de copier les `String` sous-jacentes (en effet les deux dictionnaires référencent à la fin les mêmes `String`) ?
- Copiez (ou attachez) dans le rendu la trace d'exécution de la suite de tests `pobj.motx.tme2.test.TME2Tests` fournie.