Séance 6: SCRABBLE

Université Paris-Diderot

On souhaite programmer une intelligence artificielle qui joue de manière optimale au Scrabble. La tâche centrale d'une telle IA va être de trouver à chaque tour le mot qui lui donne le plus de points. Pour simplifier, on fait abstraction des contraintes et bonus liés au positionnement du mot sur le plateau. On ne s'intéresse donc qu'au score intrinsèque du mot, calculé à partir de la valeur de chaque lettre. Pour rappel, les valeurs des lettres sont données par le tableau suivant :

Lettres	Points
e, a, i, o, n, r, t, l, s, u	1
d, g	2
b, c, m, p	3
f, h, v, w, y	4
k	5
j, x	8
q, z	10

Par la suite, on supposera que toutes les lettres sont écrites en minuscules.

Remarque:
Vous rédigerez les réponses aux exercices dans le fichier fourni `Exos.java`. Il est important de **tester** au fur et à mesure votre code, à l'aide des tests fournis dans le `main`.

Si vous bloquez sur une fonction d'un exercice, vous pourrez quand même l'utiliser dans les exercices suivants en utilisant l'implémentation compilée dans le fichier `Correction.class`. Pour ce faire, il suffit lorsque vous appelez la fonction de précéder son nom par `Correction.`, par exemple vous écrirez `Correction.fonc(42)` au lieu de `fonc(42)` pour une fonction qui s'appelle `fonc` et qui prend en argument un `int`.

Exercice 1 (Nombre d'occurrences, ⋆)
Écrivez une fonction `nbOccs` *qui prend en entrée deux chaînes de caractères* `lettre` *et* `mot`*, et renvoie le nombre de fois que* `lettre` *apparaît dans* `mot` *si* `lettre` *est de longueur 1, et -1 sinon. Par exemple,*

```
1  nbOccs("s", "sxssq") == 3
```

□

Exercice 2 (Mot jouable, ⋆⋆)
À l'aide de la fonction `nbOccs`*, écrivez une fonction* `jouable` *qui prend en entrée deux chaînes de caractères* `mot` *et* `lettres`*, et renvoie* `true` *si* `mot` *peut être obtenu en utilisant au plus 1 fois chaque occurrence de caractère apparaissant dans* `lettres`*. Par exemple,*

```
1  jouable("pomme", "memspxo") == true
```

□

Exercice 3 (Score, ⋆)
Écrivez une fonction `score` *qui prend en entrée une chaîne de caractères* `mot`*, et renvoie le score associé à* `mot`*. Par exemple,*

```
score("dindon") == 8
```

Remarque:
Il est judicieux ici d'utiliser un tableau d'entiers qui stocke la valeur associée à chaque lettre de l'alphabet. L'idée est qu'une lettre peut-être représentée dans le tableau par sa place dans l'alphabet, *calculable avec la fonction utilitaire* `placeLettre` *fournie dans le fichier* `Exos.java`. *Par exemple,*

```
placeLettre("a") == 0
placeLettre("g") == 6
placeLettre("z") == 25
```

□

Exercice 4 (Meilleur mot, ⋆⋆)
Écrivez une fonction `meilleurMot` *qui prend en entrée un tableau non-vide de chaînes de caractères* `dictionnaire` *et une chaîne de caractères* `lettres`, *et qui renvoie le mot de* `dictionnaire` *jouable avec* `lettres` *qui a le meilleur score. Si plusieurs mots ont le même score, on renverra celui qui apparaît en premier dans* `dictionnaire`. *Par exemple,*

```
meilleurMot({"because", "first", "these", "could", "which"}, "hicquwh
 ") == "which"
```

□

Exercice 5 (Bonus, ⋆⋆⋆)
Dans l'exercice 2, l'utilisation de la fonction `nbOccs` *n'est pas très efficace, car on doit reparcourir lors de chaque appel à* `nbOccs` *l'entièreté du mot.*

Proposez un algorithme plus efficace pour vérifier si un mot est jouable, et implémentez le dans la fonction `jouableEfficace`. *Celui-ci devra se baser sur l'utilisation de tableaux qui associent à chaque lettre son nombre d'occurrences.* □