Licence Sciences du Langage **Linguistique Informatique – L5NS6M2A**
Université Bordeaux 3 **2012 – 2013**

Projet: Programmation du jeu Boggle

Jeu de Boggle

Boggle est un jeu de lettres

Commençons par le présenter en citant la page Wikipedia :

(`http://fr.wikipedia.org/wiki/Boggle` consultée le 22/10/2012)

"Le jeu commence par le mélange d'un plateau (carré) de 16 dés à 6 faces, généralement en le secouant. Chaque dé possède une lettre différente sur chacune de ses faces. Les dés sont rangés sur le plateau 4 par 4, et seule leur face supérieure est visible. Après cette opération, un compte à rebours de 3 minutes est lancé et tous les joueurs commencent à jouer.

Chaque joueur cherche des mots pouvant être formés à partir de lettres adjacentes du plateau. Par « adjacentes », il est sous-entendu horizontalement, verticalement, ou en diagonale. Les mots doivent être de 3 lettres au minimum, peuvent être au singulier ou au pluriel, conjugués ou non, mais ne doivent pas utiliser plusieurs fois le même dé pour le même mot. Les joueurs écrivent tous les mots qu'ils ont trouvés sur leur feuille personnelle. Après les 3 minutes de recherche, les joueurs doivent arrêter d'écrire et le jeu entre dans la phase de calcul des points.

[...]

Lors du calcul des points, chaque joueur lit à haute voix les mots trouvés. Si deux joueurs ou plus ont trouvé le même mot, il est rayé des listes le contenant. Tous les joueurs doivent vérifier la validité d'un mot. Un dictionnaire est utilisé pour accepter ou refuser un mot (c'est souvent le dictionnaire du Scrabble (ODS)). Après avoir éliminé les mots communs aux listes des joueurs, les points sont attribués suivant la taille des mots trouvés. Le gagnant est le joueur ayant le plus grand nombre de points."

Taille du mot	Points
3	1
4	1
5	2
6	3
7	5
8 et +	11

Projet de programmation

Nous allons développer un programme informatique qui permet de trouver des solutions au jeu de Boggle. On envisage plusieurs applications possibles à ce programme : partenaire artificiel, aide à la recherche pour un enfant ou un apprenant, solutions optimales, etc.

Pour cela, nous allons vous conduire pas-à-pas dans ce projet où le jeu Boggle est un prétexte à l'initiation informatique dans le cadre du Traitement Automatique des Langues.

Modalités pratiques

Le projet est réalisé par groupes de 2 ou 3 personnes où chacune des personnes dit clairement quelle a été sa contribution personnelle.

Le travail est à rendre le 11 décembre 2012 dernier délai sous la forme d'un courriel adressé à `lionel.clement@labri.fr` et `jkirman@labri.fr`

1. dont l'objet est "Projet L2 SDL"
2. adressé par l'une des personnes du groupe seulement
3. contenant deux fichiers attachés :
   (a) `nom-prenom.pdf`, un document PDF de quelques pages qui explique le travail
   (b) `nom-prenom.py`, le code Python réalisé
4. Un accusé de réception sera envoyé systématiquement en retour. Cet AR fera foi de l'envoi.

Questions

1. Formalisation du tirage

   Chaque dé contient 6 faces, et chaque face tirée au sort est visible de façon aléatoire dans l'une des 16 cases du jeu. Il y a donc un très grand nombre de grilles différentes que nous ne pouvons pas toutes lister. Nous allons écrire un programme qui permet de produire une grille de façon aléatoire.

   On représente un dé sous la forme d'une liste de 6 caractères et une grille sous la forme d'une liste de 16 listes de 6 caractères.

   En utilisant la fonction `random.randint(i, k)` qui renvoie un nombre entier aléatoire compris entre i et k, écrire un algorithme qui produit une grille aléatoire.

   Les 16 dés sont les suivants : [E, T, U, K, N, O] ; [E, V, G, T, I, N] ; [D, E, C, A, M, P] ; [I, E, L, R, U, W] ; [E, H, I, F, S, E] ; [R, E, C, A, L, S] ; [E, N, T, D, O, S] ; [O, F, X, R, I, A] ; [N, A, V, E, D, Z] ; [E, I, O, A, T, A] ; [G, L, E, N, Y, U] ; [B, M, A, Q, J, O] ; [T, L, I, B, R, A] ; [S, P, U, L, T, E] ; [A, I, M, S, O, R] ; [E, N, H, R, I, S].

2. Implantation du tirage

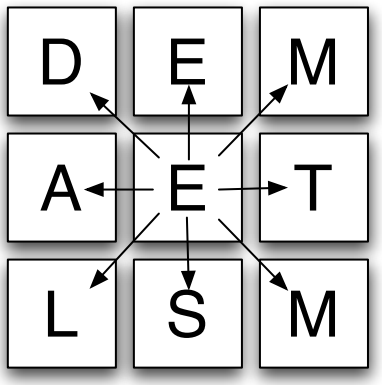
   En Python, la fonction `random.randint(i, k)` renvoie un nombre entier aléatoire compris entre i et k (inclus).

   ```
   import random
   random.randint(1, 6)
   ```

   Écrire une fonction `tirage` sans argument qui renvoie une grille Boggle de façon aléatoire.

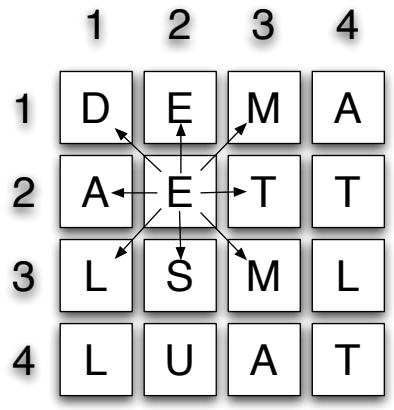
3. Cases adjacentes

   Les voisins d'une case sont représentés par les flèches suivantes

   On représentera une case sur une grille grâce aux numéros de ligne et de colonne.

Ecrire un algorithme qui donne, pour une case donnée, l'ensemble des cases adjacentes. Exemple :



adjacents de $(2,2) = [(1,1),(1,2),(1,3),(2,1),(2,3),(3,1),(3,2),(3,3)]$

On prendra garde de ne pas dépasser la grille pour les cases adjacentes des cases qui se trouvent en bordure.

4. Ecrire une fonction qui fournit une liste de cases adjacentes pour une case donnée
   On représentera une case par un couple d'entiers
5. Trouver un mot dans une liste
   Soit $[m_1, m_2, \ldots, m_k]$ une liste de mots de la langue française de plus de trois lettres.
   Écrire un algorithme qui recherche un mot dans cette liste.
   Pour cela on procédera ainsi :
   (a) On suppose que la liste de mots est triée en ordre alphabétique
   (b) On cherche le mot par fourchettes : on teste d'abord pour savoir si le mot est celui qui figure exactement au milieu de la liste
   (c) Si non il est soit avant, soit après dans la liste.
   S'il est avant, on recherche le mot dans la première moitié, s'il est après on le recherche dans la seconde moitié de la liste.

   (a) Formaliser le problème en notant
   m le mot recherché,
   $[m_i, m_{i+1}, \ldots, m_j]$ une liste de mots triés par ordre alphabétique allant du i^e mot au j^e mot,
   $a < b$ si a est avant b dans l'ordre alphabétique
   (b) Écrire un algorithme en utilisant les notations formelles qui permet de dire si oui ou non un mot donné est dans la liste $[m_0, m_1, \ldots, m_k]$
   (c) Implanter ce programme en code Python sous la forme d'une fonction qui renvoie une valeur booléenne.
6. Trouver un chemin
   Etant donné une case du jeu, on considère l'ensemble des chemins possibles, c'est-à-dire les listes de cases $[c_1, c_2, \ldots, c_k]$ où c_{i+1} fait partie des cases adjacentes de c_i. On rappelle qu'une case ne peut être visitée qu'une seule fois.
   (a) Écrire un algorithme qui permet de calculer un chemin à partir d'une case donnée
   (b) Implanter ce programme en code Python sous la forme d'une fonction qui renvoie une liste de listes de cases

7. Trouver tous les chemins valides

   On remarquera que le nombre de chemins est extrêmement important. Nous allons donc modifier le dernier programme pour que les seuls chemins valides soient les chemins qui correspondent à des préfixes de mots qui existent.

   (a) Écrire un algorithme qui permet de calculer un chemin $[c_1, c2, \ldots, c_k]$ si et seulement s'il existe un mot qui contient le préfixe $a_1 a_2 \ldots a_k$ dans le lexique (la lettre a_i se trouve dans la case c_i).

   (b) Compléter cet algorithme pour qu'il affiche tous les mots d'une grille donnée