UNIVERSITE LIBRE DE BRUXELLES
École Polytechnique de Bruxelles

novembre 2013

INFO-H-100 : Introduction à la programmation
Projet 1 : Boggle

Le but du projet est d'implémenter en *Python 3* une version simplifiée, à un joueur, du Boggle [1].

Règle simplifiées

Le jeu se déroule de la manière suivante :

Le jeu de « Boggle » en version simplifiée est un jeu à un joueur. Le jeu commence par le mélange d'un plateau (carré 4 fois 4) 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, et seule leur face supérieure est visible.

Après cette opération, le joueur cherche le mot le plus long pouvant être formé à partir de lettres adjacentes du plateau. Par « adjacentes », il est sous-entendu horizontalement ou verticalement. Les mots doivent être de 3 lettres au minimum, ne peuvent pas utiliser plusieurs fois le même dé, et doivent exister dans un dictionnaire fourni (voir plus bas).

Les points sont attribués suivant la taille des mots trouvés comme suit : si le mot a une taille de 3 ou 4 le joueur reçoit 1 point, pour des mots de 5, 6 ou 7 le joueur reçoit respectivement 2, 3 ou 5 points, finalement pour les mots de longueur 8 et plus le joueur reçoit 11 points.

À réaliser

Tout d'abord, votre programme chargera en mémoire le fichier `dico.txt` contenant les mots du dictionnaire. Pour effectuer cela, vous utiliserez la fonction `load_dico()` qui est fournie dans le module mis à votre disposition (voir plus loin). Le fichier dictionnaire devra obligatoirement se trouver dans le même répertoire que l'exécutable de votre projet.

Ensuite, le jeu commence. Il est composé de 5 tours, chaque tour se déroule comme suit :

- Le plateau est mélangé et présenté à l'utilisateur.
- Le candidat propose un mot (la saisie doit être vérifiée et ne contenir que des alphabétiques sans diacritiques).
- Le système vérifie alors les conditions suivantes :
  1. Le mot existe dans le dictionnaire fourni (`if word in dico:`).
  2. Le mot est composé uniquement des lettres adjacentes (horizontalement : à gauche ou à droite ou verticalement : en haut ou en bas) à la disposition du joueur. Chaque dé ne peut être utilisé qu'une seule fois. Pour simplifier cet algorithme et ne pas revenir en arrière lors de la recherche d'une correspondance, votre programme peut ne pas trouver un mot si, pour une lettre du plateau, il y a deux lettres adjacentes identiques. (voir exemple plus bas). Cette erreur est acceptée.
- Si ces conditions sont vérifiées, le candidat gagne les points correspondant à la taille du mot proposé (comme expliqué dans l'introduction ci-dessus). Sinon, il perd ce nombre de points.

Si le joueur se trompe, l'ordinateur lui propose de recommencer. Le joueur peut néanmoins aussi bien entrer des lettres en majuscule qu'en minuscule.

Vous devez gérer les saisies de l'utilisateur et les tester afin d'assurer la cohérence entre la donnée récupérée et le type de données requis. Par contre, les préconditions des fonctions, qui seront spécifiées dans les *docstrings*, ne doivent pas être explicitement revérifiées au début de vos fonctions. Ces conditions sont supposées remplies lors de l'utilisation de ces fonctions (voir consignes et règles de bonne pratique).

1. Voir `http://fr.wikipedia.org/wiki/Boggle` pour les règles complètes.

Exemple de partie (à des fins de lisibilité, les entrées de l'utilisateur sont soulignées et les mots mis en gras dans la grille)[2]

```
E Z F R
H K T U
A B J O
U O N V
Quel est votre mot: bonjour
Vous gagnez 5 points

V Q F X
M J S O
L C A L
T U L B
Quel est votre mot: salut
Mot introuvable. Vous ne gagnez pas de points

N T Z T
H X Q H
O K B T
W D Y G
Quel est votre mot: ho
Proposition invalide. Recommencez
Quel est votre mot: ho21
Proposition invalide. Recommencez
Quel est votre mot: howdy
Mot absent du dictionnaire. Vous ne gagnez pas de points
...
```

Dans le deuxième exemple, le plateau contient le mot SALUT (en gras) mais le programme ne l'a pas trouvé car, pour la lettre `A`, il y a deux `L` adjacents. Cette erreur est acceptée car le programme ne doit pas, dans ce cas, explorer les deux possibilités.

Le projet est fourni avec un module Python contenant certaines fonctions et documentées. Elles se trouvent dans le fichier `Boggle_cpy32.pyc` pour la version 3.2 de Python (une version 3.3 est fournie également). Les fonctions fournies dans ce module sont les suivantes :
- `load_dico(file_name)`
- `get_dices()`
- `show_grid(grid)`
- `ask_yes_no(message)`

Pour les utiliser, consultez leur documentation en utilisant la fonction `help()` dans IDLE, par exemple.

Voici un exemple de code pour importer le module et utiliser ses fonctions :

```
import Boggle_cpy32 as Boggle_mod

Boggle_mod.get_dices()
```

Pour le reste, on vous demande de découper votre programme en fonctions dont voici quelques prototypes à implémenter obligatoirement :
- `play_boggle()` : lance une partie (de 5 tours) de Boggle et affiche le nombre de points obtenus au total.
- `play_one_round(dico)` : reçoit le dictionnaire lu du fichier `dico.txt` et joue un tour de Boggle en affichant un plateau et en demandant au joueur son mot. Renvoie le nombre de points obtenus.
- `shake()` : renvoie, sous la forme d'une matrice (liste de listes) 4 x 4, le plateau des seize dés mélangé obtenus au départ de DICES.
- `word_on_board(word, board)` : renvoie `True` si le mot (word) apparaît (n'importe où) sur le plateau (board) et `False` sinon.
- `word_in_place(word, board, line, col)` : renvoir `True` si le mot (word) apparaît sur le plateau (board) en commençant en position (line, col) et `False` sinon.

Notez qu'à l'exception de `play_boggle()` et de `play_one_round(dico)`, aucune des fonctions présentées ci-dessus n'interagit avec l'utilisateur.
Ce découpage n'est pas complet et vous êtes invités à définir d'autres fonctions afin de rendre votre programme et vos fonctions le plus modulaire possible. Chaque fonction doit avoir une tâche **unique** et bien définie. En particulier, une fonction ne peut jamais dépasser 10 lignes (maximum strict).

2. La mise en gras et le soulignement sont là pour clarifier l'exemple ; dans votre programme, il s'agira de texte "normal".

Consignes

Remise du projet

Le projet est à réaliser **seul** et à remettre **au plus tard le 6 décembre 2013 à 13h00**. Nous vous demandons de vous limiter **uniquement** à la matière vue au cours des 8 premiers TPs.

Vous devez remettre une version électronique de votre code sur un dépôt Git que vous devez créer dans le groupe suivant : `http://wit-projects.ulb.ac.be/rhodecode/INFO-H-100/Projets_2013/Projet-1`.

Un tutoriel expliquant comment créer un dépôt et y soumettre votre code est disponible sur la page du cours.

Gestion des erreurs

- Dans le cadre de ce cours, vous pouvez supposer que le programmeur sait ce qu'il fait. Les fonctions qui documentent clairement leurs hypothèses dans leur *docstring* (soit toutes les fonctions que vous écrivez pour les projets, voir paragraphe suivant) ne doivent plus vérifier elles-mêmes que ces conditions sont bien remplies : c'est la responsabilité de l'appelant de passer des arguments corrects.
- Par contre, on ne peut jamais faire confiance à l'utilisateur du programme. Vous devrez donc toujours vérifier les entrées de l'utilisateur. Toute fonction dont le rôle est de récupérer une valeur entrée par l'utilisateur doit clairement documenter les promesses qu'elle fait sur sa valeur de retour (par exemple, toujours renvoyer un entier pair supérieur à 0), et **doit** effectivement renvoyer le bon type de valeur dans le bon domaine. Si l'utilisateur n'entre pas des données valides, la stratégie préférée sera d'afficher un message expliquant ce qui ne va pas, et de lui redemander.

Évaluation et derniers conseils

Les critères d'évaluation sont les suivants :
- Le code **résout exactement le problème demandé sans ajout ni apport personnel**, qualité de l'algorithmique, qualité et pertinence de la découpe en fonction, syntaxe correcte, indentation parfaite, variables bien nommées.
- Le code doit **respecter les conventions et les règles de bonne pratique** vues au cours et aux séances d'exercices (elles se trouvent sur une feuille à disposition sur le site des TP.[3]). **En particulier, vous devez être attentif à ce que chaque fonction soit correctement documentée par une *docstring***.
- Le respect des consignes.

Votre code devra être clair, simple, concis et bien documenté.

Les étudiants-assistants sont disponibles les mardis et jeudis à 12h30 dans la salle Socrate afin de répondre à vos questions.

3. `http://wit-projects.ulb.ac.be/rhodecode/INFO-H-100/Docs/BonnesPratiques_Python/archive/tip.zip`