UNIVERSITE LIBRE DE BRUXELLES
Ecole Polytechnique de Bruxelles

Mars 2012

INFO-H-100 : Introduction à la programmation
Projet 2 : Jeu de *Scrabble*

On vous demande de réaliser un jeu de *Scrabble* [1] simplifié en *Python 2.7*.

Fonctionnement réel du jeu

Le *Scrabble* (marque déposée) est un jeu de société et un jeu de lettres dont l'objectif est de réaliser des points en plaçant des mots, au départ de tirages aléatoires de lettres, sur une grille carrée dont certaines cases sont primées.

Le plateau de jeu est une grille carrée de 15 sur 15 (soit 225 cases en tout), dont certaines cases colorées, dites "cases multiplicatrices", valorisent la lettre ou le mot placé dessus. Il y a 102 lettres en tout dans la version francophone. La lettre E est la plus fréquente avec 15 occurrences, mais ne vaut qu'un point, tandis que les "lettres chères" J, K, Q, W, X, Y et Z sont uniques mais valent 8 (J et Q) ou 10 points (K, W, X, Y et Z).

Chaque joueur dispose d'un chevalet contenant ses 7 lettres. Au premier tour et après avoir joué, le joueur tire aléatoirement des lettres dans le sachet afin de compléter son chevalet. Le premier mot doit passer par la case centrale (souvent recouverte d'une étoile) et les mots suivants doivent s'appuyer sur des mots déjà placés.

Le joueur qui parvient à placer 7 lettres en posant un mot reçoit un bonus de 50 points. Ce genre de coup est appelé "scrabble" (en anglais, on dit bonus ou bingo). Le joueur a "scrabblé". La recherche d'un scrabble doit évidemment être la priorité lors de chaque coup.

Si un scrabble se révèle impossible à former (par exemple, sur un tirage tel que "BGIOOUV"), le joueur doit porter son attention sur les cases "mot compte triple" et "mot compte double", si elles sont accessibles. Dans le cas contraire, le joueur cherchera à faire fructifier les lettres chères déjà placées, ou tentera de "maçonner" (placer un mot en le collant à un autre mot déjà placé sur la grille).

Les cases "lettre compte triple" et "lettre compte double" ont également leur intérêt, permettant notamment de placer une lettre chère ou semi-chère en "pivot", doublant ainsi le score octroyé par la case multiplicatrice.

En français, chacun des mots joués doit figurer dans l'édition en vigueur de *L'Officiel du jeu Scrabble* ou *ODS*.

A titre d'exemple, le mot WOK permet de gagner un grand nombre de points au Scrabble, le W et le K valant chacun 10 points. Le placement de ce mot, sur une case "mot compte double" ou "mot compte triple", est surnommé "Coup du Wok" [2].

La section précédente est inspirée de Wikipedia.

1. Voir http://fr.wikipedia.org/wiki/Scrabble
2. http://fr.wikipedia.org/wiki/Wok

Programme à réaliser

On vous demande de réaliser, **uniquement à l'aide de la matière vue au cours et aux séances d'exercices**[3], un jeu de *Scrabble* simplifié à un seul joueur en *Python 2.7*. Votre programme devra se dérouler comme suit :

1. Tout d'abord, votre programme chargera en mémoire le dictionnaire officiel du *Scrabble* ainsi que l'ensemble des lettres et leurs scores respectifs disponibles sous forme de fichiers textes sur la page web des projets[4]. Ces fichiers devront obligatoirement se trouver dans le même répertoire que votre programme. Le format de ces fichiers est détaillé ci-dessous.
2. Ensuite, un tirage aléatoire de 7 lettres dans le sachet sera effectué pour initialiser le chevalet du joueur.
3. Pour placer un mot, le joueur entre le mot complet (même s'il utilise des lettres présentes sur le plateau), sa direction (horizontale ou verticale) ainsi que la position de la première lettre du mot à placer sur le plateau. Les indices des positions commencent à 0. Le premier mot doit obligatoirement passer par la case centrale, en (7,7).
4. Une fois le mot entré par l'utilisateur, le système doit vérifier les conditions suivantes :
   (a) Le mot existe dans le dictionnaire fourni. Les diacritiques sont ignorées.
   (b) Le mot entré ne dépasse pas les bornes du plateau.
   (c) Le mot et sa position sont compatibles avec les lettres actuellement sur le plateau.
   (d) Le joueur possède les lettres nécessaires pour jouer son mot.

   Si les conditions ne sont pas vérifiées, le programme affiche un message d'erreur et retourne en 3.
5. Si ces conditions sont vérifiées, le mot est placé sur la grille et les lettres nécessaires sont retirées du chevalet. Seuls les points de ce mot sont calculés, c'est-à-dire uniquement la somme des scores des lettres du mot entré, y compris les lettres se trouvant déjà sur le plateau de jeu. Ces points sont ajoutés au score du joueur courant. Nous ne vous demandons ni de vérifier les mots perpendiculaires au mot placé, ni de calculer les points de ces mots, ni de prendre en compte les règles supplémentaires comme le "scrabble" ou les cases multiplicatrices.
6. Si le sachet n'est pas vide, le système tire aléatoirement dans le sachet le nombre de lettres consommées par le placement du mot (pour arriver à un total de 7 lettres, ou moins si le nombre de lettres restantes ne le permet pas) et les ajoute au chevalet. Sinon, la partie est terminée.
7. Le plateau de *Scrabble* est affiché ainsi que le chevalet et le score actuel.

Le fichier `french.dic` contient l'entièreté des mots acceptés au *Scrabble*. Chaque ligne de ce fichier contient un mot en majuscule. Ces mots sont classés par taille et par ordre alphabétique. Extrait du fichier :

```
PSYCHISMES
PSYCHOGENE
PTERANODON
```

Les lettres sont contenues dans le fichier `french.let`. Chaque ligne se compose de la manière suivante : la lettre, le nombre de jetons de cette lettre suivis des points de cette lettre. Extrait du fichier :

```
E 15 1
A 9 1
I 8 1
```

3. Si vous voulez utiliser une librairie non vue au cours, veuillez demander l'autorisation **par mail** à vos assistants.
4. http://cs.ulb.ac.be/public/teaching/infoh100/projets

Apports personnels

Les apports personnels seront valorisés à hauteur de 5 points sur 20. **En d'autres mots, un projet parfait sans apport personnel aura une valeur de 15 points sur 20.** Ces apports personnels peuvent être, par exemple, des règles [5] ignorées dans la version proposée ci-dessus.

Voici quelques exemples d'apports personnels :

- Vérifier que le mot entré s'appuie sur un mot existant (difficulté moyenne) ;
- Le mode multijoueur (difficulté moyenne) ;
- Les cases multiplicatrices (difficile) ;
- Le "scrabble" (facile) ;
- Vérifier et comptabiliser les nouveaux mots formés perpendiculairement au mot placé (très difficile) ;
- Les lettres blanches ou jokers (difficile) ;
- Changer les lettres (difficulté moyenne) ;
- Affichage amélioré grâce à l'Art ASCII [6] (facile) ;
- Séparer le code en au moins deux modules (deux fichiers `.py`). Le premier module contiendra les fonctions logiques du projet (le **modèle**, vérifier qu'un mot est dans le dictionnaire, ajouter un mot au plateau de jeu, etc). Le second module contiendra les interactions avec l'utilisateur (la **vue**, c'est à dire l'affichage et les entrées) et importera le premier module. C'est donc ce second module qui sera exécuté pour lancer votre jeu de *Scrabble* (facile) ;
- Interface graphique (difficile ; prendre rendez-vous avec Gary pour explications) ;
- ...

Délivrables

1. Le code source sera rendu au format électronique sur le serveur *Mercurial* (voir plus bas).
   Chaque fonction devra être correctement documentée. C'est-à-dire pour chaque fonction : son prototype, une description succincte de la tâche qu'elle remplit, ses paramètres et leur domaine (le type et les valeurs possibles), le type de ses valeurs de retour et leur domaine et un exemple d'utilisation. Par exemple [7] :

```
def daysInMonth(month, year):
    """Calcule le nombre de jours d'un mois.

    Arguments:
        month (int) : le mois considere, entre 1 et 12.
        year (int) : l'annee consideree.

    Valeurs de retour:
        int. Retourne le nombre de jours associes au mois et a l'annee consideres.
        Retourne -1 si les parametres ne respectent pas les domaines.

    Exemples:
        >>> print daysInMonth(2, 2012)
        29
        >>> print daysInMonth(13,2012)
        -1
    """
    <<code de la fonction>>
```

   Rappelez-vous que la *docstring* est le texte qui sera affiché lorsque vous tapez `help(daysInMonth)` dans l'interpréteur ; il s'agit donc de la documentation nécessaire à un **programmeur** pour *utiliser* la fonction. Ce texte ne s'adresse pas à l'utilisateur du *programme* (le joueur), et n'explique pas le fonctionnement *interne* de la fonction ; il explique uniquement **comment utiliser** la fonction.

5. http://www.fisf.net/index.php?Itemid=52&id=29&option=com_content&task=view
6. http://fr.wikipedia.org/wiki/Art_ASCII
7. basé sur http://packages.python.org/an_example_pypi_project/sphinx.html#full-code-example

2. De plus, un rapport dactylographié vous est demandé. Celui-ci devra comprendre :
   (a) Une page de garde indiquant clairement vos **noms, numéros de matricules et votre série**.
   (b) **Une explication claire et justifiée de vos choix algorithmiques et de structures de données pour les fonctions qui le nécessitent**.
   (c) Un manuel d'utilisation illustré de votre programme fini de maximum 2 pages.

   L'ensemble du rapport ne devrait pas faire plus de 5 pages.

Consignes

Le projet se fera obligatoirement par **groupes de deux étudiants de la même série**.

1. Le code source sera rendu via le serveur *Mercurial*, de la même manière que pour le premier projet. On vous demande cette fois de créer votre dépôt dans le groupe `INFO-H-100»2011»2`. Le nom du dépôt devra être `<netid1>-<netid2>`, où les deux *netids* doivent être dans l'ordre alphabétique. Par exemple, `boverhae-gaverhae`. [8]
2. Le rapport au format papier (uniquement des feuilles et des agrafes, pas de plastique ni de reliure) devra être rendu au bureau UB4.131.
3. Une copie du rapport au format PDF devra se trouver dans le dépôt Mercurial.

Ces délivrables devront être rendus pour le 30 avril 2012 à 12h30 [9]. Les groupes remettant leur projet après cette limite seront pénalisés.

Vous devrez défendre votre projet devant les assistants lors des semaines 23 et 24.

L'évaluation de ce projet prendra en compte des points suivants :
– La résolution du problème énoncé.
– Le respect scrupuleux de l'énoncé.
– Le respect scrupuleux des conventions [10] et des règles de bonne pratique [11] publiées sur le site web des projets.
– La qualité du rapport : intérêt, présentation, concision, grammaire et orthographe.
– Votre défense en salle machine devant les assistants.
– L'efficacité de vos algorithmes et structures de données, en restant bien sûr dans le cadre de la matière vue aux cours et aux séances d'exercices.

Les élèves assistants sont disponibles les midis [12] dans la salle Socrate afin de répondre à vos questions.

BON TRAVAIL !

8. Bien que vous puissiez utiliser Mercurial uniquement pour remettre votre projet à la dernière minute, il s'agit à la base d'un logiciel conçu pour faciliter le travail en groupe. Vous pouvez donc l'utiliser pour communiquer entre vous, comme expliqué en annexe.
9. Pour les délivrables électroniques, l'heure prise en compte sera celle du serveur. N'attendez pas la dernière minute !
10. http://cs.ulb.ac.be/public/teaching/infoh100/conventions
11. http://cs.ulb.ac.be/public/teaching/infoh100/bonne_pratique
12. Leur horaire est disponible ici : http://cs.ulb.ac.be/public/teaching/infoh100#horaire_des_guidances

Annexe : Utilisation en groupe de *Mercurial*

Il existe plusieurs manières d'utiliser *Mercurial* pour faciliter le travail en groupe. La plus simple consiste à utiliser un dépôt commun sur le serveur, ce qui nécessite évidemment que les deux membres du groupe aient accès au dépôt. Le dépôt doit quand même être créé en cochant la case `Private`, comme pour le projet 1, puisqu'on ne veut pas que tout le monde y ait accès. Une fois le dépôt créé sur le serveur *Rhodecode*, par contre, il est possible de le partager avec d'autres utilisateurs via les `settings` du projet.

En résumé, la démarche est la suivante, en appelant les deux membres du groupe Alice et Bob.

1. Alice crée le dépôt sur le serveur *Rhodecode* en utilisant son `netid` et celui de Bob. Cette création se passe de la même manière que pour le projet 1, à ceci près qu'il faut cette fois-ci sélectionner un groupe (`INFO-H-100/2011/2`).
2. Une fois le dépôt créé, Alice peut le cloner sur son ordinateur et l'utiliser en suivant les mêmes instructions que pour le projet 1. Il faut cependant encore donner l'accès à Bob.
3. Pour donner l'accès à Bob, Alice doit aller sur le site web du serveur Rhodecode (`http://informa2.ulb.ac.be`) et, là, aller chercher les informations sur son projet (en cliquant sur `INFO-H-100`, puis sur `2011`, puis sur `2`, et enfin sur le nom de son projet).
4. Une fois sur la page de résumé de son projet, Alice doit encore cliquer sur `Options`, en haut à droite, puis sur `Settings` dans le menu qui apparaît.
5. Sur la page de configuration du projet, tout en bas à gauche, se trouve le menu pour donner accès à quelqu'un d'autre. Alice doit alors cocher la troisième case (`write`) et entrer le `netid` de Bob dans la case de texte, puis cliquer sur `Save`.
6. Bob a maintenant accès au dépôt d'Alice comme si c'était le sien, et peut donc également cloner le dépôt et travailler sur le code.

Une fois l'accès configuré pour les deux étudiants, il faut encore effectivement utiliser le serveur comme "point de rencontre". Pour coordonner l'effort, il faut que les deux étudiants synchronisent régulièrement leur version avec le serveur.

Si Alice et Bob ont tous les deux modifiés le même fichier sur leurs ordinateurs respectifs, la synchronisation peut poser problème. En général, Mercurial arrivera à s'en sortir tant qu'Alice et Bob ont modifié des morceaux différents du fichier. Prenons un cas pratique.

Si Alice modifie le fichier `main.py` sur son ordinateur, pour le synchroniser avec le serveur, elle devra utiliser la commande `hg push`. Mais cela ne fait qu'envoyer ses modifications sur le serveur. Pour également recevoir les modifications du serveur [13], Alice doit également utiliser les deux commandes `hg pull` (pour télécharger les changements) et `hg update` (pour intégrer les changements du serveur dans son propre projet). Au moment où elle exécute la commande `hg push`, le serveur va refuser le changement s'il est incompatible avec la version qu'il abrite [14]. La commande `hg pull` fonctionne toujours, puisqu'elle ne fait que télécharger les changements, sans essayer de les appliquer. La commande `hg update` va par contre rater dans les mêmes circonstances que `hg push` : quand les deux versions sont incompatibles. Mercurial va alors demander à Alice de regarder les deux versions et de choisir celle qu'elle veut conserver.

Une fois ce choix fait, Alice peut remettre le code du serveur à jour avec `hg push`.

Pour plus de détails, vous pouvez consulter le site `http://hginit.com/02.html`, en ignorant les parties qui expliquent comment mettre en place le dépôt central (puisque c'est fait pour vous par le serveur *Rhodecode*).

13. Cette partie est nouvelle : comme dans le cas du premier projet, une seule personne avait accès au serveur, le cas de télécharger des modifications depuis le serveur vers l'ordinateur local ne se présentait pas.
14. Cela signifie que Bob a mis à jour la version sur le serveur et qu'il a modifié les mêmes sections du fichier qu'Alice.