Mini Projet 2010-2011

TOP: Techniques and tOols for Programming

Première année

★ Modalités

Ce projet est à réaliser en binôme. Vous indiquerez la composition de votre binôme dans un fichier dénommé `AUTHORS` (placé à la racine du répertoire de votre projet) qui doit contenir les logins UNIX des membres du groupe, un par ligne. L'usage de la plate-forme de travail collaborative de l'école (et du SVN qu'elle offre en particulier) est *très fortement* conseillée : *"version control means you can relax"* – Karl Fogel.

Travail à rendre : Vous devez rendre un mini-rapport de projet (5 pages maximum, format pdf) en plus de vos fichiers sources. Vous y détaillerez les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport) par chaque membre du groupe.

Comment rendre votre projet : Vous devez placer les fichiers nécessaires dans un répertoire sur neptune, vous placer dedans, et invoquer la commande suivante (seuls les fichiers sources et le document pdf sont copiés). Vous pouvez lancer le script autant de fois que vous le souhaitez, seule la dernière soumission est conservée.

```
/home/EqPedag/quinson/bin/rendre_projet TOP
```

Avant le Lundi 14 Mars 2011, à 23h59.

(le script n'acceptera pas de soumission en retard)

Un projet ne compilant pas sera sanctionnée par une note adéquate. Toute soumission par email ou sous une autre forme sera refusée.

Évaluation : Des soutenances individuelles de projet seront organisées la semaine du 21 Mars 2010. Vous serez jugé sur la qualité de votre programme, celle de votre rapport et votre capacité à expliquer son fonctionnement. Vous aurez également à répondre à des questions en rapport avec les TP effectués dans le module, sans rapport avec le projet.

★ Travail personnel et honnêteté

Ne trichez pas ! Ne copiez pas ! Si vous le faites, vous serez lourdement sanctionnés. Nous ne ferons pas de distinction entre copieur et copié. Vous n'avez pas de (bonne) raison de copier. En cas de problème, nous sommes prêt à vous aider. Encore une fois : en cas de doute, envoyez un courriel à vos enseignants, ça ne les dérange pas.

Par tricher, nous entendons notamment :

- Rendre le travail d'un collègue avec votre nom dessus ;
- Obtenir une réponse par Google™ ou autre et mettre votre nom dessus ;
- Récupérer du code et ne changer que les noms de variables et fonctions ou leur ordre avant de mettre votre nom dessus (*"moving chunks of code around is like moving food around on your plate to diguise the fact that you havn't eated all your brussel sprouts"*) ;
- Permettre à un collègue de *s'inspirer* de votre travail. Assurez vous que votre répertoire de travail n'est lisible que par vous même.

Il est plus que très probable que nous détections les tricheries. Chacun a son propre style de programmation, et personne ne code la même chose de la même manière. De plus, il existe des programmes très efficaces pour détecter les similarités douteuses entre copies (MOSS, `http://theory.stanford.edu/~aiken/moss/`). En cas de litige grave, seul un historique progressif de vos travaux (comme en offre le système SVN) constitue une preuve de votre innocence. *Commit soon, commit often.*

En revanche, il est possible (voire conseillé) de discuter du projet et d'échanger des idées avec vos collègues. Mais vous ne pouvez rendre que du code écrit par vous-même. Vous indiquerez dans votre rapport toutes vos sources d'inspiration (comme les sites internet de vulgarisation de l'informatique que vous auriez consulté), en indiquant brièvement ce que vous en avez retiré.

Grand merci à Gérald Oster pour le montage de ce projet.

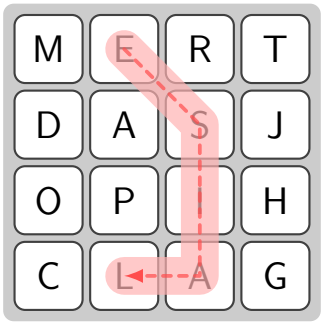
★ Présentation du problème : Boggle

[*extrait de l'encyclopédie Wikipedia* (`http://fr.wikipedia.org/wiki/Boggle`)]

Boggle est un jeu de lettres conçu par Alan Turoff et déposé par Parker Brothers / Hasbro, Inc.

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. 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	3	4	5	6	7	8+
Points	1	1	2	3	5	11

Pour la version classique de Boggle, il y a $4,36 \times 10^{22}$ possibilités de dispositions de lettres. Cela correspond à environ 40 000 milliards de milliards de combinaisons.

L'objectif de ce mini-projet est d'écrire un programme permettant à un joueur humain d'affronter un ordinateur. Malheureusement, il y a bien peu de chances que le joueur humain réussisse à battre son adversaire informatisé.

★ Travail à réaliser

De quelle manière vous allez procéder ?

Votre programme commencera par lire un fichier indiquant pour chaque dé quelles sont les lettres qui sont inscrites sur ses faces. Ensuite, le programme réalisera un lancé des dés et positionnera ce jet sur le plateau de jeu.

C'est le joueur humain qui aura l'avantage de commencer la partie. Il devra saisir les mots qu'il trouve (un mot à la fois !). À chaque saisie, votre programme vérifiera que ce mot respecte la contrainte de longueur (au moins 3 caractères de long), que le mot n'a pas encore été proposé (un mot n'est comptabilisé qu'une seule fois, même s'il apparaît plusieurs fois sur le plateau), que le mot appartient bien au dictionnaire de mots connus et bien entendu qu'il est possible de former ce mot à partir des faces visibles des dés positionnés sur le plateau. Si tout ces tests sont valides alors le mot sera ajouté à la liste de mots trouvés par le joueur et le score du joueur sera crédité des points correspondants. Lorsque le joueur humain ne souhaite plus saisir de nouveau mot, il l'indique en saisissant un mot vide (uniquement un retour chariot).

C'est alors au tour du joueur artificiel de jouer. Votre programme va donc chercher tous les mots qui sont dans le dictionnaire et qu'il peut réaliser à partir du plateau. Chaque mot trouvé qui n'aurait pas été trouvé par le joueur humain est ajouté à la liste des mots trouvés par le joueur artificiel, et le score du joueur artificiel est crédité du nombre de points correspondants.

Généralement, le joueur artificiel vaincra sans appel son pendant humain. Mais le joueur humain est libre de recommencer autant qu'il le souhaite ...

Les dés

Les lettres du Boggle ne sont pas placées au hasard sur les faces des dés. En fait, elle sont placées de manière à ce que les lettres les plus communes soient tirées le plus souvent et de manière à ce que l'on ait une bonne combinaison entre le nombre de voyelles et de consonnes.

Pour recréer cette situation, votre programme devra être capable de lire un fichier dont le format est le suivant. Chaque ligne contient soit un commentaire si la ligne commence par un point virgule, soit la description d'un dé. La description d'un dé est un mot de 6 lettres indiquant les 6 lettres à placer sur les 6 faces du dés. Il y a exactement 16 descriptions de dé par fichier. Le contenu suivant vous donne un exemple de fichier valide :

dices-definition.txt

```
; description des dés
BAJOQM
RALESC
LIBART
TOKUEN
ROFIAX
AVEZDN
NULEGY
MEDAPC
SUTELP
HEFSIE
ROMASI
GINEVT
RUEILW
RENISH
TIEAAO
DONEST
```

Lors de la phase d'initialisation, votre programme devra donc lire le fichier, créer les dés correspondants, générer une configuration d'un plateau en plaçant de manière aléatoire chaque dé sur le plateau. Par ailleurs, pour chaque dé, la face visible sera également tirée de manière aléatoire.

Une fois la phase d'initialisation terminée, vous êtes prêt à implémenter les deux types de recherches récursives : l'une pour le tour de jeu de l'utilisateur (recherche d'un mot spécifique), et l'autre pour le tour de jeu de l'ordinateur (recherche exhaustive de tous les mots).

Pour l'utilisateur, la méthode récursive cherche un mot spécifique sur le plateau et s'arrête dès que celui est trouvé. Pour l'ordinateur, la méthode récursive cherche tous les mots contenus dans le dictionnaire qui sont susceptibles d'être sur le plateau.

Vous pourriez être tenté d'unifier ces deux recherches récursives, mais c'est une très mauvaise idée. **Nous vous demandons explicitement d'implémenter séparément ces deux fonctions récursives**.

Le tour du joueur humain

Une fois que le plateau est affiché, le joueur peut saisir chaque mot qu'il trouve sur le plateau. Le joueur indiquera son souhait de ne plus saisir de mot en saisissant une simple ligne blanche (un simple retour chariot).

À chaque fois qu'un joueur aura saisi un mot, votre programme devra vérifier les conditions suivantes :

- le mot est un mot d'au moins trois lettres ;
- le mot est contenu dans le dictionnaire ;
- le mot apparaît sur le plateau (il est formé d'une séquence de lettres adjacentes et un dé est utilisé au plus une fois) ;
- le mot n'est pas encore déjà contenu dans la liste de mots saisis par l'utilisateur.

Si l'une de ces conditions échoue, le programme doit en informer explicitement l'utilisateur et ne pas lui attribuer de points pour cette saisie. Par contre, si le mot satisfait toutes ces conditions, il doit être ajouté à la liste de mots saisis par l'utilisateur et le score de l'utilisateur doit être mis à jour en conséquence.

La longueur du mot détermine le nombre de point : 1 point pour le mot lui-même et 1 point supplémentaire pour chaque lettre au dessus du nombre minimum de lettre dans un mot. Ainsi, puisque la longueur minimale d'un mot est 3, le mot `geek` rapporte 1 point, le mot `idiot` rapporte 2, et le mot `esialien` rapporte 5 points.

Le tour du joueur artificiel

Lors du tour du joueur artificiel, la tâche de l'ordinateur consiste à trouver tous les mots que le joueur humain aurait oublié, et ce, en cherchant de manière récursive tous les mots qui peuvent débuter par la lettre présente sur un dé du plateau. Lors de cette phase de jeu, les mêmes conditions sur les mots trouvés s'appliquent. Cependant, l'ordinateur n'est pas autorisé à comptabiliser les points des mots qui auraient déjà été trouvés par le joueur humain.

Comme dans tout algorithme de recherche exponentielle, il est important de limiter la recherche au maximum afin de s'assurer que la tâche sera achevée dans un temps raisonnable.

L'une des stratégies les plus importantes consiste à déterminer lorsque la recherche s'engage sur une voie morte afin de l'abandonner au plus vite. Par exemple, si la recherche a construit un chemin menant au préfixe `zx`, vous pouvez utiliser votre dictionnaire pour déterminer qu'il n'y a pas de mot qui commence par ce préfixe. Et donc, vous pouvez arrêter cette recherche pour continuer sur une voie plus prolifique. Si vous n'appliquez pas cette optimisation, votre ordinateur passera un temps non négligeable à vérifier la construction de mots qui n'existent pas tels que `zxgub` ou `zxaep`.

★ Programme de travail

Comme il s'agit principalement de votre premier projet informatique de cette envergure, et afin que vous puissiez travailler de manière efficace vers un objectif bien défini, nous vous proposons de découper votre travail en 5 étapes

Étape 1 : Lecture de dés, affichage du plateau, mélange des dés

Dans cette étape, vous concevez et écrivez les classes qui représentent à un dé et un plateau de jeu. Ajouter les méthodes permettant d'afficher un dé et un plateau. Ajouter les méthodes nécessaires pour lire la définition des dés depuis un fichier. Ajouter les méthodes permettant de générer aléatoirement le plateau en mélangeant les dés.

Classe : `boggle.Dice`

Responsabilités :
- Conserve la valeur (un caractère) de chacune des six faces du dés.
- Conserve quelle est la face visible du dé.
- Permet de consulter la face visible du dé.
- Permet de calculer un nouveau lancer de dé (la face visible est donc mise à jour).

Collaborateurs :
- `java.util.Random`

Classe : `boggle.Board`

Responsabilités :
- Conserve la position de chacun des 16 dés.
- Permet de mélanger les dés (la nouvelle position d'un dé est tirée aléatoirement, un nouveau lancer de ce dé est réalisé pour obtenir une nouvelle valeur).
- Permet d'afficher l'état du plateau.

Collaborateurs :
- `java.util.Random`
- `boggle.Dice`

Choisir aléatoirement une face d'un dé est trivial en utilisant la méthode `nextInt()` de la classe `java.util.Random`. Ré-arranger les dés est un peu plus difficile. Vous pouvez utiliser l'algorithme suivant qui permet de mélanger des éléments dans un tableau à 2 dimensions.

Algorithme de permutation aléatoire des dés

```
1 for (int row = 0; row < numRows; row++) {
2     for (int col = 0; col < numCols; col++) {
3         Dice aDice = this.field[row][col]; // récupère un élément
4         aDice.roll(); // effectue un lancer de dé
5         int rowDest = row + rand.nextInt(numRows - row);
6         int colDest = col + rand.nextInt(numCols - col);
7         swap(row, col, rowDest, colDest); // change la place des deux dés de place
8     }
9 }
```

Classe : `boggle.Main`

Responsabilités :
- Permet de lire un fichier et de créer les 16 dés qui y sont décrits.
- Permet de créer et de conserver un plateau de jeu.
- Permet d'afficher le plateau de jeu.

Collaborateurs :
- `java.util.Scanner`
- `java.io.FileInputStream`
- `boggle.Dice`
- `boggle.Board`

Étape 2 : Le tour du joueur humain (sans recherche des mots sur le plateau)

Dans la classe représentant votre application (`boggle.Main`), écrivez la boucle permettant à un joueur de saisir les mots qu'il trouve. Si les conditions ne sont pas respectées (mot déjà saisi, longueur minimale non respectée, mot non inclus dans le dictionnaire) rejetez explicitement le mot, sinon comptabiliser les points. Ne faites pas de supposition quand au nombre de mots maximal que le joueur peut trouver. Lors de cette étape, vous ne vous intéressez pas à savoir si le mot se trouve ou non sur le plateau.

Classe : `boggle.Main`	
Responsabilités : – Permet de lire un fichier et de créer les 16 dés qui y sont décrits. – Permet de créer et de conserver un plateau de jeu. – Permet d'afficher le plateau de jeu. – **Permet de savoir si un mot est valide (taille, nouveau mot, mot appartenant au dictionnaire).** – **Permet de calculer le score d'un mot.** – **Permet de créer et de conserver un joueur.** – **Permet à un utilisateur de saisir son nom de joueur.** – **Permet à un joueur de saisir les mots qu'il a trouvés.**	**Collaborateurs :** – `java.util.Scanner` – `java.io.FileInputStream` – `boggle.Dice` – `boggle.Board` – **boggle.Lexicon** – **boggle.Player**

Classe : `boggle.Player`	
Responsabilités : – Conserve le nom d'un joueur. – Conserve le score d'un joueur. – Permet de consulter le nom d'un joueur. – Permet de consulter le score d'un joueur. – Permet de mettre à jour le score d'un joueur. – Conserve la liste des mots trouvés par un joueur.	**Collaborateurs :** – `java.util.ArrayList`

En plus de la classe principale, vous aurez besoin d'une classe représentant le dictionnaire. Le contenu de ce dictionnaire sera chargé à partir d'un fichier qui contiendra sur chaque ligne un et un seul mot écrit en majuscules (non accentué). Un fichier de contenant le dictionnaire officiel du Scrabble (364,370 mots) est disponible à l'adresse `http://www.isc.ro/lists/ods.zip`.

Classe : `boggle.Lexicon`	
Responsabilités : – Permet de lire un fichier et d'ajouter les mots lus. – Permet d'ajouter un mot. – Permet de supprimer un mot. – Permet de savoir si un mot est contenu dans le dictionnaire. – Permet de savoir combien de mots sont dans le dictionnaire.	**Collaborateurs :** – `java.util.ArrayList` – `java.util.Scanner` – `java.io.FileInputStream`

Étape 3 : Trouver un mot sur le plateau

Écrivez la recherche récursive d'un mot sur le plateau afin de vérifier que le joueur humain ne triche pas. Rappelez-vous, un mot valide doit respecter deux règles : i) les lettres doivent être adjacentes, ii) un dé ne peut être utilisé qu'une et une seule fois dans un mot. N'oubliez pas que dés que vous réalisez que vous ne pouvez pas former un mot à partir de cette position, vous devez passer à la position suivante.

Classe : `boggle.Board`	
Responsabilités : – Conserve la position de chacun des 16 dés. – Permet de mélanger les dés (la nouvelle position d'un dé est tirée aléatoirement, un nouveau lancer de ce dé est réalisé pour obtenir une nouvelle valeur). – Permet d'afficher l'état du plateau. – **Permet de chercher (récursivement) si l'on peut construire à un mot donné à partir de l'état du plateau (en effectuant un parcours des dés).**	**Collaborateurs :** – `java.util.Random` – `boggle.Dice` – `boggle.Lexicon`

Classe : `boggle.Main`	
Responsabilités : – Permet de lire un fichier et de créer les 16 dés qui y sont décrits. – Permet de créer et de conserver un plateau de jeu. – Permet d'afficher le plateau de jeu. – Permet de savoir si un mot est valide (taille, nouveau mot, mot appartenant au dictionnaire, **pouvant être construit à partir du plateau**). – Permet de calculer le score d'un mot. – Permet de créer et de conserver un joueur. – Permet à un utilisateur de saisir son nom de joueur. – Permet à un joueur de saisir les mots qu'il a trouvés.	**Collaborateurs :** – `java.util.Scanner` – `java.io.FileInputStream` – `boggle.Dice` – `boggle.Board` – `boggle.Lexicon` – `boggle.Player`

Étape 4 : Trouver tous les mots sur un plateau (tour du joueur artificiel)

Il est temps d'implémenter la logique du joueur artificiel et notamment l'algorithme de recherche récursive couplé à un dictionnaire. Cette récursivité est une recherche exhaustive. Vous devez donc explorer toutes les positions du plateau à la recherche des mots qui sont dans le dictionnaire. Cette étape est la plus difficile. Réfléchissez bien à toutes les étapes de la recherche avant de développer votre solution. Pensez également aux optimisations que vous pouvez mettre en place une fois que votre recherche fonctionne. Utilisez un dictionnaire plus petit lors de votre développement et pourquoi pas un plateau contenant moins de dés.

Pour réaliser l'algorithme de recherche, plusieurs approches sont envisageables :

(a) Prendre chaque mot du dictionnaire et utiliser la méthode de recherche développée à l'étape 3. Malheureusement, avec un dictionnaire d'une taille raisonnable (360 000 mots), cette approche est trop coûteuse en temps.

(b) Écrire une méthode récursive qui génère toutes les combinaisons de lettres présentes sur le plateau, et pour chaque combinaison, regarder si celle-ci est contenu dans le dictionnaire. Malheureusement, il y a environ 40 000 milliards de milliards de combinaisons.

En fait, il faut utiliser la seconde approche et arrêter au plus tôt la génération d'une combinaison si celle-ci ne permet pas de générer des mots du dictionnaire. Autrement dit, à chaque génération d'une combinaison, si celle-ci ne peut pas servir de préfixe à un mot du dictionnaire, alors on arrête la descente de la recherche avec retour arrière (*backtracking*).

Pour utiliser cette méthode, vous devez être capable de connaître si un mot est un préfixe d'un ou plusieurs mots du dictionnaire. Pour cela, vous devez donc modifier l'implémentation de votre dictionnaire afin que celui repose sur l'utilisation d'un *PATRICIA trie* (cf. bibliographie).

Vous étudierez sûrement ce genre de structure dans le module SD (Structures de Données). Dans tous les cas, nous vous invitons fortement à lire la page Wikipedia sur le sujet (`http://en.wikipedia.org/wiki/Radix_tree`).

Si cette structure de données ou l'implémentation que vous devez télécharger vous pose un problème, n'hésitez pas à contacter vos enseignants.

Classe : `boggle.LexiconTrie`	
Responsabilités : – Permet de lire un fichier et d'ajouter les mots lus. – Permet d'ajouter un mot. – Permet de supprimer un mot. – Permet de savoir si un mot est contenu dans le dictionnaire. – Permet de savoir combien de mots sont dans le dictionnaire. – Permet de savoir si une chaîne de caractère est un préfixe pour un ou plusieurs mots du dictionnaire.	**Collaborateurs :** – `java.util.Scanner` – `java.io.FileInputStream` – `org.ardverk.collection.PatriciaTrie`

Classe : `boggle.Board`	
Responsabilités : – Conserve la position de chacun des 16 dés. – Permet de mélanger les dés (la nouvelle position d'un dé est tirée aléatoirement, un nouveau lancer de ce dé est réalisé pour obtenir une nouvelle valeur). – Permet d'afficher l'état du plateau. – Permet de chercher (récursivement) si l'on peut construire à un mot donné à partir de l'état du plateau (en effectuant un parcours des dés). – **Permet de chercher (récursivement) tous les mots que l'on peut construire à partir de l'état du plateau (en utilisant le dictionnaire).**	**Collaborateurs :** – `java.util.Random` – `boggle.Dice` – **boggle.LexiconTrie**

Classe : `boggle.Main`	
Responsabilités : – Permet de lire un fichier et de créer les 16 dés qui y sont décrits. – Permet de créer et de conserver un plateau de jeu. – Permet d'afficher le plateau de jeu. – Permet de savoir si un mot est valide (taille, nouveau mot, mot appartenant au dictionnaire, pouvant être construit à partir du plateau). – Permet de calculer le score d'un mot. – Permet de créer et de conserver un joueur. – Permet à un utilisateur de saisir son nom de joueur. – Permet à un joueur de saisir les mots qu'il a trouvés. – **Permet de faire jouer le joueur artificiel.** – **Permet de calculer le score du joueur artificiel.**	**Collaborateurs :** – `java.util.Scanner` – `java.io.FileInputStream` – `boggle.Dice` – `boggle.Board` – **boggle.LexiconTrie** – `boggle.Player`

Étape 5 : Boucle pour jouer plusieurs fois, peaufinage

Maintenant qu'il est possible à un joueur de jouer une partie complète, il est temps de permettre à un joueur de jouer autant de partie qu'il le souhaite.

Vous pouvez maintenant peaufiner votre programme : vérifiez que vous avez bien géré toutes les saisies possibles et cherchez à optimiser les différentes routines.

★ Bonus (tâches facultatives)

> *To infinity, and beyond !* [Buzz Lightyear, Toy Story]

L'état du code à la fin de la section précédente constitue le minimum à atteindre pour avoir une note potable (entre 12 et 15 selon le degré de finition et le rapport). Pour aller (vers l'infini et) au delà, il vous faudra implémenter certaines des extensions suivantes.

Quelles que soient les extensions que vous choisissez, il devra toujours être possible d'utiliser la version originale et classique du jeu.

Esthétisme *(difficulté : faible)* : Améliorez l'aspect esthétique de votre application (un petit peu d'ASCII Art (`http://fr.wikipedia.org/wiki/Art_ASCII`) et voire même quelque code d'échappement ANSI pour mettre de la couleur).

Jeu à deux *(difficulté : faible)* : Offrez la possibilité à deux joueurs humains de se confronter lors d'une partie. L'ordinateur indiquera les mots non trouvés à la fin de la partie.

Hall of Fame *(difficulté : faible)* : Votre programme sauvegardera et affichera les meilleurs scores (le nom des joueurs associés).

Taille variable *(difficulté : assez faible)* : Modifier votre programme afin que les dimensions du plateau puissent être changées facilement, afin par exemple de pouvoir jouer sur une grille (42×42).

Extension 3D *(difficulté : moyenne)* : Modifier votre programme afin que le plateau ne soit plus uniquement un carré (4×4), mais un cube ($4 \times 4 \times 4$).

Limite de temps de jeu *(difficulté : moyenne)* : Ajouter la possibilité de limiter le temps de jeu de chaque joueur. Par exemple, le joueur humain n'a que 5 minutes pour trouver et proposer ses mots (idem pour l'ordinateur).

Interface graphique *(difficulté : élevée)* : Proposer une interface utilisateur graphique (en utilisant la bibliothèque Swing, par exemple).

Klingon Boggle *(difficulté : for geek only) ;b* : Proposer la possibilité de jouer au Boggle Klingon (`http://bigbangtheory.wikia.com/wiki/Klingon_Boggle`).

Imaginer vos propres extensions ! ! ! Vous êtes bien entendu libres d'implémenter toute autre extension ou option à ce jeu (qui ne voudrait pas jouer au Boggle en utilisant le Kinect™ ?).

★ Bibliographie

L'Officiel du Scrabble : 364,370 mots : `http://www.isc.ro/lists/ods.zip`.

La documentation de l'API Java : `http://download.oracle.com/javase/6/docs/api/`.

Radix tree (Wikipedia) : `http://en.wikipedia.org/wiki/Radix_tree`.

Une implémentation de la structure de données PATRICIA-Trie () : `http://code.google.com/p/patricia-trie/`.

Vous pourrez trouver des informations complémentaires à l'adresse suivante :
`http://www.loria.fr/~quinson/Teaching/TOP/`

★ Indications

Dans ce projet vous devez mettre en applications les connaissances que vous avez acquises dans le module POO ainsi que dans le module TOP.

Vous devrez donc réaliser un programme en respectant les principes de la programmation objet. Il vous est donc fortement conseillé de chercher à modéliser le problème en manipulant des classes et des objets. Par exemple, vous serez amenés à définir une classe pour le programme principal, une pour chaque dés, une pour le plateau de jeu, une pour représenter la liste des mots trouvés, une pour le dictionnaire, une pour un joueur, etc.

Dernier conseil, n'attendez pas la dernière minute pour démarrer ce projet. Il n'est pas compliqué, mais parfois on peut rapidement passer beaucoup de temps à trouver une erreur bête. N'hésitez pas à contacter vos enseignants pour vous débloquer.

★ Cas d'utilisation

Vous trouverez ci-dessous à titre d'exemple, l'affichage donné lors l'utilisation du programme que vous devez réaliser.

```
 ______                     __
|   __ \.-----..-----..-----.|  |.-----.
|   __ <|  _  ||  _  ||  _  ||  ||  -__|
|______/|_____||___  ||___  ||__||_____|
                |_____||_____|
                   ESIAL/TOP Project'11

Loading lexicon, Please wait.
Please enter your name: Sheldon Cooper

        /---\/---\/---\/---\
        | r || r || g || t |
        \---/\---/\---/\---/
        /---\/---\/---\/---\
        | s || h || m || i |
        \---/\---/\---/\---/
        /---\/---\/---\/---\
        | g || a || d || d |
        \---/\---/\---/\---/
        /---\/---\/---\/---\
        | e || t || s || q |
        \---/\---/\---/\---/

Player:Sheldon Cooper        Score:0
Please enter a word: tas
Player:Sheldon Cooper        Score:1
Please enter a word: gate
Player:Sheldon Cooper        Score:2
Please enter a word: mats
Player:Sheldon Cooper        Score:3
Please enter a word: bidule
Sorry, but you cannot make this word from the board.
Please enter a word: de
Sorry, but you must enter at least a 3-characters word.
Please enter a word: gate
Sorry, but this word is already in your list.
Please enter a word:
Congratulations Sheldon Cooper, your score is 3
The computer will now play...
I just found 'git'
I just found 'sas'
I just found 'sate'
I just found 'sage'
I just found 'samit'
I just found 'hast'
I just found 'haste'
I just found 'hate'
I just found 'mit'
I just found 'mas'
I just found 'mat'
I just found 'mats'
I just found 'mate'
I just found 'mage'
I just found 'mahdi'
I just found 'image'
I just found 'gate'
I just found 'admit'
I just found 'age'
I just found 'ami'
I just found 'date'
I just found 'dam'
I just found 'dit'
I just found 'eta'
I just found 'team'
I just found 'tas'
I just found 'tag'
I just found 'tags'
I just found 'stage'
Player:computer        Score:32
Bazinga!
```