Rapport d’avancement

1er rapport d’avancement (22/11/2022)



**Lucas LEVEQUE, Berachem MARKRIA & Joshua LEMOINE**

11/10/2022

BUT2 Informatique

# INTRODUCTION

Ce premier rapport d’avancement se concentre sur la mise en place du moteur de jeu en C et de la création des divers dictionnaires demandés en JAVA. A ce jour n’ayant aucun sujet pour le développement web ni pour le développement de base de données aucune avancée n’a été effectuée sur ces points.

# DICTIONNAIRES JAVA :

Le code JAVA est découpé en trois classes:

* **DictionaryMaker :** classe qui crée les différents dictionnaires ainsi qu’un fichier contenant les fréquences des lettres
* **DictionnarySearcher :** classe qui se base sur les fichiers créés par DictionnaryMaker pour trouver un mots et ses définitions
* **NormalizedExtractor :** classe qui renvoie tous les mots enregistrés dans les dictionnaires sous forme normalisée. Sa sortie peut être redirigé avec un ‘>’ afin de créer un fichier texte utilisé par la suite dans le C

Les trois fichiers générés par le classe *DictionaryMaker* sont :

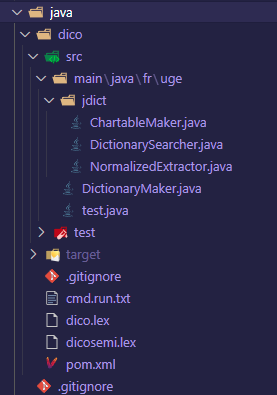
* **XXX.json :** l’ensemble des mots et de leurs définitions sous format json
* **XXXsemi.lex :** l’ensemble des mots sous formes normalisées associées aux offsets de leurs versions non normalisées dans *XXX.json.* Offsets triées par ordre alphabétiques des mots auxquels il correspondent
* **XXX.lex :** l’ensembles des offsets des mots normalisés de *XXXsemi.lex*, triées par ordre alphabetique

\*XXX correspond au paramètre donné à *DictionaryMaker* comme nom des fichiers à générer.

Les commandes utilisables avec ces différents fichier :

* **java fr.uge.DictionaryMaker *path lang XXX* :** avec *path* qui indique le chemin absolu du fichier xml de génération, *lang* la langue voulue, *XXX* le nom des fichiers qui seront générés
* **java fr.uge.jdict.DictionarySearcher *XXX word* :** avec *XXX* le nom des fichiers dans lesquels il faut chercher et *word* le mot à rechercher. Le programme détecte automatiquement s'il est normalisé.
* **java fr.uge.jdict.NormalizedExtractor *XXX* :** avec *XXX* le nom des fichiers dans lesquels il faut chercher les statistiques. Renvoie dans la sortie courante tous les mots normalisés (peut être redirigé par ‘>’).
* **java fr.uge.jdict.CharTableMaker *XXX* :** avec *XXX* le nom des fichiers dans lesquels il faut chercher les statistiques. Renvoie dans la sortie courante chaques lettres et son nombre d’occurence dans les mots du dictionnaires (peut être redirigé par ‘>’).

La seule commande demandée qui ne fonctionne pas encore est : **java fr.uge.jdict.NormalizedExtractor *XXX "types, type, type"***. Permet d’extraire dans la sortie courante tous les mots normalisés appartenant aux types passés en paramètre (adjectif, verbe, nom…)

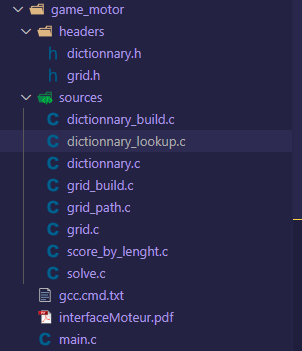


*hiérarchie de la partie JAVA*

# MOTEUR DE JEU EN C :

Le code C est découpé en 8 librairies:

* **dicoToStaticTree.c :** c’est dans cette librairie qu’on définit les structures CSTree et les Static Tree, que l’on utilisera dans les autres librairies. Cette librairie permet de passer du fichier XXX.txt à un arbre
* **dictionnary\_build.c :** qui sert à créer le fichier XXX.lex utilisé dans le reste du moteur de jeu à partir de l’arbre créer dans dicoToStaticTree.c
* **dictionnary\_lookup.c :** contient les fonctions permettant la lecture du fichier XXX.lex pour chercher si un mot souhaité existe dans le fichier lex.
* **grid.c** : regroupe l’ensemble des fonctions annexes liées aux grilles du parcours, à l’affichage, à la récupération des coordonnées des voisins d’une case jusqu’à la conversion de coordonnées 2D en 1D et inversement.
* **grid\_build.c :** fichier qui sert à la création d’une grille avec principalement l’utilisation d’une fonction qui génère pour chaque case de la grille : une lettre aléatoire tirée du fichier de fréquences.
* **grid\_path.c** : emblématique fichier qui renvoie si un mot est présent dans une grille passée en argument. Il contient de nombreuses fonctions qui parcourent récursivement les voisins de cases en cases afin de repérer les lettres du mot dans la grille.
* **solve.c :** Contient les fonctions permettant d’afficher la liste des mots contenu dans le fichier XXX.lex et dans la grille passé en paramètre.
* **score\_by\_lenght.c** : fichier qui prend un ensemble de mots en paramètres et renvoie le score total en lien avec la taille de ceux-ci.



*hiérarchie de la partie C*

# 

# 

# PROBLÈMES RENCONTRÉS

Au niveau du java les problèmes rencontrés ne sont pas nombreux. La partie la plus dure étant de penser la structure des différents fichiers de stockages de sorte à ce que les différentes commandes ne soient pas trop gourmandes en temps de calcul. D'où cette architecture en quatres fichiers qui prend un certain temps à ce générer mais améliore grandement la vitesse de recherche par la suite.

Au niveau du C, la mise en place de l’architecture fût laborieuse à mettre en place. D'autre part, nous avons eu beaucoup de soucis liés aux fuites mémoires qui rendaient difficile et plus longue la réalisation de la partie moteur du jeu.

# FUTUR

Pour le java l’implémentation de la dernière commande obligatoire ainsi que l’ajout de quelques tests au niveau de fichiers passés en paramètre pour éviter tout comportement non voulu est prévu. En fonction du temps restant après ces opérations les parties non obligatoires comme la compression seront envisagées.

Pour le C, il nous reste à corriger certains bugs à corriger liés à des fuites mémoires et d'implémenter d’autres méthodes de calculs de score. Il nous faudra également repenser un peu la structure des fichiers pour leurs futures utilisations , mais cela devrait prendre peu de temps.