**Rapport du projet informatique :**

**Joueur de scrabble**

Réalisé par :

• Yannick SICARD

• Ayoub SERARFI

• Noa LAUVERGNE

Année universitaire : 2021/2022.

Groupe : STH1-B7.

***1) Introduction :***

Le scrabble est un jeu de société et un jeu de lettres, créé en 1946 aux Etats-Unis, qui consiste à former des mots à partir de lettres tirées au hasard et à les placer sur une grille de façon qu’ils s’intègrent aux mots déjà constitués, pour objectif de cumuler des points.

La principale mission de notre projet fut de réaliser la conception d’un programme JAVA qui, à partir d’un dictionnaire déjà donné, permet à l’utilisateur de trouver des mots répondant aux différentes contraintes qu’il peut rencontrer au cours du jeu. Ce qui va lui permettre de gagner la partie adverse.

***2) Langage Java :***

Développé par Sun Microsystems depuis la fin des années 1980, Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C.

Java possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

• Java est indépendant de toute plate-forme : il n'y a pas de compilation spécifique pour chaque plate-forme. Le code reste indépendant de la machine sur laquelle il s'exécute.

• Java est simple : le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple et la surcharge des opérateurs, ...

• Java est fortement type : toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données.

• Java assure la gestion de la mémoire : l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.

• Java est sûr : la sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation. Il ne peut pas y avoir d'accès direct à la mémoire.

• Java est économe : le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.

***3) Partage des tâches :***

Au sein de notre groupe, on a envisagé une répartition de travail sur 3 parties selon le plan du projet :

• Partie 1 - Chercher des mots dans le dictionnaire à partir du tirage.

• Partie 2 - Répondre aux différentes contraintes imposées sur l’utilisateur comme la taille du mot et les lettres déjà placées dans le mot.

• Partie 3 - Perfectionner le programme JAVA en évaluant le score de la partie adverse et ne le pas dépasser par un grand écart.

En plus de ça, on a essayé d’être méthodique et de faire les choses dans l’ordre.

***4) Description du programme :***

Nous allons présenter dans cette partie les principales étapes qu’on a suivi au cours de la réalisation de ce projet.

Tout d’abord, on a essayé de transformer le dictionnaire déjà donné en un nouveau dictionnaire où les accents étaient supprimés et les lettres majuscules ont été transformés à des lettres minuscules. Pour ce faire, on a enlevé les accents manuellement du dictionnaire d’origine. Par suite, on a importé le dictionnaire modifié à notre programme, puis on a essayé de transformer chaque ligne de ce dictionnaire en lettres minuscules grâce à la fonction String.toLowerCase(), pour la recopier de suite dans un autre fichier texte qu’on a nommé ‘’ dictionnaire\_minuscules.txt’’.

Ensuite, il fallait faire rentrer à l’utilisateur son tirage de 7 lettres attachées. Et le lire au clavier sous forme d’un String (chaîne de 7 caractères dans ce cas), en l’occurrence cela fût relativement simple en utilisant la méthode Scanner.

Puis, à l’aide d’une méthode où on a créé 3 tableaux pour chaque ligne du dictionnaire et on a rempli :

• Le 1er et le 3ème tableau : par les lettres contenues dans la ligne.

• Le 2ème tableau : par les lettres contenues dans le tirage.

On a pu calculer le nombre des lettres en commun entre la ligne et le tirage, ainsi que le nombre des lettres en commun entre la ligne et elle-même en utilisant des boucles for et des entiers compteurs count.

En comparant les deux résultats obtenus, on a conclu que :

• Si les lettres en commun entre la ligne et elle-même sont plus nombreux que les lettres en commun entre la ligne et le tirage : donc il existe des lettres étrangères dans la ligne et qui n’appartiennent pas au tirage, d’où le mot contenu dans cette ligne n’est pas jouable.

• Si cas contraire : donc toutes les lettres contenues dans la ligne appartiennent effectivement au tirage, alors on peut recopier le mot dans un nouveau fichier texte temporaire qu’on a nommé ‘’temporaire.txt’’.

Ce fichier, dont nous nous sommes servis par la suite, contient donc tous les mots jouables par le joueur, sans prendre en compte les conditions du jeu.

Juste après, il fallait faire demander à l’utilisateur de saisir la taille du mot demandée et créer un tableau de même taille que celle du mot.

Puis, à l’aide d’une boucle for, on demande à l’utilisateur de saisir pour chaque position caractère du mot recherché soit :

• La lettre imposée par le jeu : s’il y en a une.

• Un 0 : si aucune condition n’est imposée pour cette position.

Ensuite, on fait remplir le tableau qu’on vient de créer par ces valeurs (une lettre ou un 0).

Et puis on commence à traiter le fichier texte temporaire, qu’on a créé précédemment, ligne par ligne tel que :

• Si la longueur de la ligne est différente de la taille rentrée par l’utilisateur :

on passe à l’autre ligne.

• Si la longueur de la ligne est égale à la taille rentrée par l’utilisateur : on passe à la comparaison des lettres contenues dans la ligne avec les lettres contenues dans le tableau, en ignorant les cases où il y a des 0.

Après cela, on recopie chaque mot satisfaisant ces conditions dans un nouveau fichier texte temporaire qu’on a nommé ‘’temporaire2.txt’’.

Ensuite, pour calculer les valeurs des points que vaut chaque ligne de ce texte, on a utilisé une variable points dont la valeur augmentait en fonction des lettres contenues dans chaque ligne.

Suite à cela, il n’y avait plus qu’à afficher la ligne (le mot) et le nombre des points qu’il valait.

La partie 3 consiste à simuler l’évolution d’une partie, on ajoute à chaque tour le nombre de points marqués au score total d’Antoine et de Gisèle. Il faut aussi qu’Antoine ait moins de 20% d’avance sur Gisèle pour ne pas se faire démasquer.

Pour simuler le score de Gisèle, nous avons choisi le niveau facile en attribuant un nombre de points aléatoire. Le nombre de points que peut inscrire Gisèle est de 1 à 10, afin de rester dans des scores raisonnables étant donné qu’on ne prend pas en compte les mots compte double, lettre compte triple… Pour le score d’Antoine, on exécute les parties 1 et 2 afin de trouver les mots jouables ainsi que leurs points. L’utilisateur choisit alors quels mots il souhaite jouer et rentre le nombre de points que vaut le mot choisi. Ce nombre de points est ajouté à son score total.

Ensuite il a fallu rajouter la condition de l’écart faible. Si le score d’Antoine est supérieur de plus de 20% de celui de Gisèle, Antoine va placer des mots valant moins de points pour ne pas éveiller de soupçons. Pour intégrer cela dans le programme, on va comparer les deux scores totaux. Ensuite la structure conditionnelle if nous permet de tester si oui ou non le score d’Antoine est trop élevé par rapport à celui de Gisèle. A partir de là, on distingue deux cas, soit le score est supérieur trop grand, soit il n’est pas trop élevé ou inférieur à celui de Gisèle. Dans ce second cas, rien ne se passe et la partie continue en suggérant des mots à l’utilisateur qui rentre le plus grand nombre de points jouables afin de gagner la partie. Dans l’autre cas, un message s’affiche pour le prévenir que son score est grand. La liste de mots s’affiche alors de la même manière mais il est invité à sélectionner un mot valant moins de points pour réduire son avantage. L’affichage de la liste de mots jouables se fait grâce à la partie 1 et 2 qui est intégrée à la partie 3 sous la forme d’une fonction nommée « trouverlesmots » et qui est placée juste avant que l’utilisateur rentre le score d’Antoine de sorte à ce qu’il puisse voir quels mots il lui est possible de jouer.

Il est aussi intéressant de noter que lors du premier tour du jeu, la condition des 20% ne s’applique pas puisque les deux participants commencent avec un score de 0. Pour cela, nous avons réécrit la partie permettant de lire les points marqués et de les additionner au score total en dehors de la boucle if. Enfin nous avons intégré la boucle if dans une nouvelle boucle while. Celle-ci permet d’arrêter ou de continuer la partie quand l’utilisateur le souhaite. Après que le tour d’Antoine et Gisèle soit passé, un message s’affiche et demande si l’on souhaite arrêter ou continuer. L’utilisateur rentre alors 1 pour continuer et 0 pour arrêter.

Au cours de la réalisation de ce projet, on a utilisé plusieurs méthodes et fonctions dans notre code et qu’on n’a pas pu citer dans la description précédente. Par exemple :

• BufferedReader : qui permet de lire des caractères à partir d'un flux tamponné.

• BufferedWriter : qui est utilisée pour simplifier l'écriture de texte dans un flux de sortie de caractères.

• trim() : qui retourne la chaîne débarrassée des blancs, tabulations et espaces inutiles en début et fin de chaîne.

• write() : qui écrit le caractère en paramètre dans le flux.

• System.getProperty("line.separator") : pour obtenir le caractère de nouvelle ligne.

• close() : qui ferme le flux et libère les ressources qui lui étaient associées.

• nextLine() : qui permet de lire une ligne de texte jusqu'au prochain retour à la ligne.

• char [] = new char[t] : pour créer un tableau de taille t.

• length() : qui renvoie le nombre de caractères présents dans une chaîne de caractères.

• charAt() : qui renvoie le caractère à l'index spécifié dans une chaîne.

***5) Auto-évaluation :***

D’une part, on pense que nous nous sommes plutôt bien sorti puisque dans l’ensemble le programme répond à la problématique de départ, même s’il subsiste quelques erreurs. La plus importante d’entre elles étant qu’il est impossible de rentrer un tirage contenant des lettres en double ou en triple. Par ailleurs, bien qu’on soit parvenu à faire afficher le nombre de points que vaut chaque mot, on n’est pas parvenu à les classer dans l’ordre de celui rapportant le plus de points à celui en rapportant le moins.

Au cours de la réalisation de ce projet, on a beaucoup progressé puisqu’avant d’en arriver à cette version finale qui répond correctement à la problématique, on a fait beaucoup d’essais en employant d’autres méthodes. En plus des compétences utilisées dans ce programme, on a appris à nous servir par exemple de la méthode indexOf, la méthode contains, les boucles Try/catch…

Finalement, l’ensemble du programme aurait pu être améliorée en proposant des saisies contrôlées quand l’utilisateur rentre son tirage ou un nombre. En effet, il est facile de se tromper au moment des contraintes lorsqu’on ne souhaite pas de contraintes et qu’on enchaîne les 0. La seule saisie contrôlée se trouve au niveau de la boucle while ou les seules valeurs possibles sont 0 et 1 alors que la variable accepte tous les entiers.

Pour conclure, on a réussi à tenir compte des retours qui nous ont été fait lors de la soutenance et cela nous a permis de terminer le programme. On est aussi satisfait de notre travail de groupe car nous partions de connaissances assez faibles en codage en début d’année, et nous avons réussi à créer un programme certes pas des plus optimal mais qui reste fonctionnel et abouti.