Rapport IN104 - Jeu Wordle

Augustin Descourvières et Gaëtan Cloitre

Avril-Mai 2022

Introduction

Ce projet consiste au développement du jeu *Wordle* sur console en utilisant le langage de programmation C. L'objectif de ce projet d'IN104 est de nous permettre de pratiquer et d'approfondir nos connaissances en programmation utilisant la langage C vue dans les autres cours d'IN, de prendre en main l'outil *Github* et ainsi apprendre à travailler en équipe sur un projet d'informatique. Nous allons ici vous présenter la structure de notre projet et nos choix d'implémentation.

Lors de L'exécution du programme dans le terminal, l'utilisateur peut choisir entre 3 modes de jeu (jeu classique, première IA, seconde IA). A la fin d'une partie, l'utilisateur récupère la main dans le terminal. Libre à lui de relancer le programme si il veut rejouer au jeu classique ou tester les IAs.

Sommaire

1 Dictionnaire

La première étape de ce projet a été de construire le dictionnaire de mot de cinq lettres de la langue française ainsi que les différentes fonctions permettant de travailler sur ce dictionnaire et récupérer les informations dont nous avions besoin pour programmer le jeu.

1.1 Choix du dictionnaire en ligne

Nous avons fait le choix de partir d'un dictionnaire de la langue française, afin de pouvoir proposer une version du jeu française, tandis que la version originale est en anglais. De plus, afin de faciliter le traitement, nous avons cherché sur internet le dictionnaire du jeu *SCRABBLE* en majuscule. Le choix de le prendre en majuscule nous permet de ne pas avoir à traiter le cas des lettres accentuées. C'est à partir de ce dictionnaire ainsi crée que nous avons pu développer notre version du jeu *Wordle* et les différentes IAs.

1.2 Stockage des mots

Lors de traitement du dictionnaire, nous nous sommes posés la question de quelle serait la structure permettant de stocker le dictionnaire des mots de 5 lettres. Nous avons d'abord tenter de construire un tableau de chaîne de caractères, avec l'avantage de pouvoir accéder facilement avec son indice, et de pouvoir parcourir l'ensemble des mots de manière efficace. Pour se faire, nous avons défini une *struct* :

```
struct dico_en_tableau {
int taille;
char** tableau_mots;
};
```

Cependant, défini de cette façon, le programme ne compilait pas et nous n'étions pas à l'aise avec les techniques vue en IN103. Nous nous sommes donc rabattu sur une solution utilisant l'écriture et la lecture de fichier texte notions dont nous maîtrisions plus. Nous avons en revanche perdu la facilité d'accéder à un élément, car désormais, il faut ouvrir le fichier et parcourir toutes les lignes avant d'accéder à la ligne voulue.

1.3 Implémentation

Lors de l'écriture de nos fonctions, nous avons pris soin, à chaque fois qu'il fallait renseigner le longueur des mots, de le faire avec la variable *LONGUEUR*, défini comme suit :

```
#define LONGUEUR 5
```

Cela nous permettrait très facilement de changer les règles du jeu, pour pouvoir proposer une version du jeu ou l'on doit deviner des mots de longueur autre que 5.

Ensuite, nous avons défini plusieurs fonctions nous permettant de passer d'un dictionnaire de la langue française à un dictionnaire de la langue française ne contenant que des mots de cinq lettres. Nous avons alors notre base de donnée à partir de laquelle nous allons pouvoir récupérer aléatoirement un mot à deviner. D'autres fonctions que nous avons définies permettant de vérifier si un mot existe dans le dictionnaire ou alors quel est le nombre de mots d'un fichier texte nous serons utile par la suite, notamment pour les IAs.

Enfin, comme nous avons fait appelle à la fonction *rand*, nous avons pris soin d'inclure la librairie *time.h*

2 Affichage

Notre affichage est pour l'instant simple mais il semble assez efficace et robuste aux multiples situations auxquelles il doit faire face. Il est cependant améliorable, notamment au niveau de l'esthétisme de l'affichage graphique. Pour faciliter le jeu de l'utilisateur, nous avons décidé d'afficher un clavier avec les lettres possiblement dans le mot (à l'image du jeu "officiel" en ligne).

2.1 Choix d'affichages

Pour réaliser une affichage facile à comprendre pour le joueur et assez rapide à implémenter, nous avons décider de représenter les différentes situations comme ceci :

- si la lettre est bien placée, on affiche **x**
- si la lettre est présente mais au mauvais endroit, on affiche **o**
- si la lettre n'est pas présente dans le mot à deviner, on affiche **-**

2.2 Implémentation

Le programme consiste à tester pour chaque lettre du mot de l'utilisateur si cette lettre est présente dans le mot à deviner, si elle est présente dans le mot à deviner mais pas à la bonne position ou si elle est finalement absente du mot à deviner.
Intrinsèquement, comme on commence par tester à partir de l'indice 0 dans la chaîne de caractère, on donne donc la priorité aux informations des premières lettres plutôt que sur les informations des lettres bien placés.

Par exemple, si le mot à deviner est LILAS et que le mot testé est ALLER, alors il s'affichera à l'écran : "ooo–", au lieu de "oox–" si l'on mettait la priorité sur les lettres bien placé.

Pour avoir le même affichage que dans le jeu officiel, nous avons codé dans l'IA n°2 une façon de prioriser les lettres bien placées. Au delà de l'aspect de l'affichage, il nous fallait stocker les informations que l'on récupéraient lors de chaque test pour pouvoir ensuite les exploiter, possibilité que la solution d'affichage présenté ici ne permettait pas.

Notre affichage est implémenté en une seule fonction (à laquelle s'ajoute la fonction "recherche" permettant de vérifier le mot de l'utilisateur). Il aurait peut-être été plus logique de coder l'affichage grâce à plusieurs fonctions mais nous avons eu la bonne surprise de faire marcher notre fonction du premier coup, nous avons donc décidé de nous focaliser ensuite sur les IA plutôt que de simplifier le code en découpant la fonction en plusieurs sous-fonctions.

2.3 Améliorations possibles

Pour améliorer cette affichage, nous aurions souhaité afficher les lettres en couleur comme dans le jeu officiel. Cependant, nous avons du faire face à un manque de temps et il nous semblait plus pertinent de nous concentrer sur l'optimisation de notre programme (et en particulier de nos IA) que d'améliorer l'affichage graphique. En effet, avec le README, le jeu est facile à comprendre et à prendre en main tel qu'il est actuellement donc cet aspect n'était pas notre priorité.

En terme de complexité temporelle l'affichage est en $O(\text{LONGUEUR}^2)$, ce qui nous semble être la meilleure complexité possible (comment faire mieux étant donné qu'il faut parcourir les deux mots ?).

3 IA

Nous avons chacun fait une méthode, n'utilisant pas de pré-requis de la théorie de l'information. En effet, nous avons préféré implémenté des IAs associées à nos logiques de jeu. Sans aucun doute, ces IAs sont améliorables mais sont une bonne base de réflexion concernant ces questions.

La meilleure combinaison pour débuter est déjà connue : TARIE (cf probabilité de présence des lettres en français)
Nos IA commenceront donc par ce mot pour pouvoir aboutir au plus vite et dans un maximum de cas à la solution.

3.1 IA n°1

Cette première version d'intelligence artificielle se base en partie sur la même programmation que l'affichage, l'idée est la suivante :

1. Parcourir simultanément le dernier mot proposé par l'IA et celui à deviner pour trouver les lettres bien placées (x) ou celles présentes mais mal placées (o)
2. Compléter le mot par des lettres aléatoires là ou les lettres n'étaient pas les bonnes (-) en testant à chaque itération si le mot appartient au dictionnaire

Cette IA est efficace en terme de proportion de succès mais cependant elle est présente un net manque de rapidité notamment lorsque le mot à deviner ne présente aucune lettre commune au mot TARIE. En effet, si aucune lettre n'est retenue, le programme doit tourner jusqu'à trouver un autre mot du dictionnaire en ne testant que des lettres aléatoires. (par exemple, si aucune lettre du mot TARIE n'est présente dans le mot à deviner, il se peut que l'ordinateur surchauffe un peu jusqu'à trouver une combinaison de 5 lettres aléatoires formant un mot du dictionnaire).
Pour ne pas avoir à faire autant d'itération, nous avons pensé à une nouvelle solution : chercher les mots directement dans le dictionnaire. Cette nouvelle technique vous est présentée dans l'IA n°2.

3.2 IA n°2

Cette deuxième IA cherche à exploiter au maximum les informations que l'on peut acquérir lorsque l'on teste un mot, à savoir :

- Quelles sont les lettres bien placées et leurs positions dans le mot ?
- Quelles sont les lettres mal-placées ?
- Quelles sont les lettres impossibles ?

A chaque fois que l'on teste un mot, on garde en mémoire ces informations, puis on vient ré écrire dans un nouveau dictionnaire que les mots qui vérifient les informations que l'on a récupérer. En d'autre terme, à chaque itération, on enlève de la liste des mots possibles les mots qui possèdent des lettres impossibles, puis les mots qui possèdent d'autres lettres que la lettre sure à la position de cette lettre bien placée, puis les mots qui ne possèdent pas toutes les lettres mal placées. Enfin, on tire au hasard un mot parmi tous ceux qui respectent les conditions ci-dessus.

Cette IA implique donc de lire et d'écrire dans des fichiers textes différents, ce qui peut être lourd en terme de mémoire. Cependant, en terme de temps d'exécution du programme, ce dernier est assez rapide.

Cependant, cette IA ne fonctionne pas à tous les coups. A plusieurs occasions, lors de la dernière itération, il n'y avait que quelques mots possibles (par exemple RIRES et RIDES). Or comme on tire le mot de manière aléatoire dans le fichier des mots possibles, on a donc une chance sur 2 de tirer le bon. Par ailleurs, il y a aussi le cas ou les informations recueillis lors d'un essai ne sont pas assez discriminante. Par exemple, l'information "il n'y a pas de **Y** dans le mot" est moins discriminante que l'information "il n'y a pas de **A** dans le mot".

3.3 Comparaison et améliorations possibles

La seconde IA est donc beaucoup plus rapide que la première (par construction). Cependant, elle nécessite beaucoup de ressource mémoire car à chaque essai, il y a la réécriture de 3 dictionnaires.
Pour diminuer la complexité spatiale, il faudrait modifier directement le dictionnaire original en supprimant les mots indésirables (cf description de l'IA n°2).

Il y a plusieurs autres pistes d'amélioration de la deuxième IA, que nous n'avons malheureusement pas eu le temps d'implémenter:

> Enlever les mots qui possèdent à une position donnée une lettre mal-placée dont on sait qu'elle n'est pas la.
>
> Modifier la fonction de tirage aléatoire du mot pour prendre un mot qui peut nous apporter le plus d'informations nouvelles, en utilisant notamment la fréquence d'apparition des lettres dans la langue française.

Conclusion

Ce projet d'informatique en groupe nous a permis de nous confronter à différents enjeux, allant de la difficulté de programmer en C et de la compréhension de ce langage proche du langage de l'ordinateur, jusqu'au travail de groupe et en particulier l'utilisation d'outil tel que *Github* pour faciliter la collaboration sur un projet informatique.

Nous avons pu nous rendre compte de l'utilité de pouvoir stocker les codes sur un site tel que *Github*, permettant de récupérer assez facilement le travail des collaborateurs. Cependant, dans notre cas, l'utilisation de cet outil a été pour

nous la source de grande difficulté, que ce soit pour la création du projet (nous avons du nous y reprendre à 3 fois), que pour l'échange d'information. En effet, nous avons eu à de nombreuses reprises eu des problèmes de merge et de fichier corrompu en essayant de "push" le projet.

De plus, ce projet nous a permis de comprendre certains aspects plus spécifiques de la programmation en C :

- Les fichiers header : ce type de fichier était présent dans nos précédentes matières d'informatique mais nous n'avions jamais eu à les implémenter. Nous avons donc appris à nous en servir. Leur utilité est d'autant plus grande que le projet présente un grand nombre de fonctions et de documents. En particulier, nous avons compris l'utilité de mettre une condition de non définition pour que le fichier header ne compile qu'une seule fois.
- La fonction *rand* est une fonction déterministe. Il faut donc utiliser la librairie time.h pour obtenir un vrai nombre aléatoire.
- L'allocation mémoire des mots : il faut manipuler des chaînes de caractères de longueur LONGUEUR+1 pour avoir une case mémoire pour le "\0", sinon les mots se suivent de manière continue car il n'y a plus la séquence d'arrêt.

Si nous avions pu accorder plus de temps à ce projet, nous aurions aimé pouvoir (en plus de ce qui a déjà été suggéré):

- Développer la fonction permettant de trouver le mot idéal à tester en premier
- Supprimer les mots non usuels du dictionnaire pour rendre le jeu adapté au langage courant
- Développer d'autre IAs
- Mesurer quantitativement l'occupation de la mémoire afin de comparer précisément les IAs.