Rapport IN104

Laura DUCKI
Valentin HALBERT

May 17, 2022

1 Introduction

Le projet d'IN104 que nous avons choisi est la création du jeu Wordle. Ce qui nous intéressait était que nous pouvions coder un jeu auquel nous étions susceptibles de jouer, ce qui rend la tâche plus ludique.
Pour ce faire, le projet se déroulait en deux étapes : tout d'abord coder le jeu, puis ensuite demander à un ordinateur de le résoudre de la meilleure manière.
Dans ce rapport nous expliquons grossièrement les différentes étapes de nos codes, et rapidement les problèmes que nous avons rencontré. Nous complèterons ces remarques lors de la soutenance orale.

2 Etape 1 : Création du jeu

a) Trouver un dictionnaire : **dico.txt**

Nous avons commencé par importer un dictionnaire contenant tous les mots de 5 lettres de la langue française. Pour cela, nous avons mis tous les mots dans un fichier .txt.
Le dictionnaire que nous avonc choisi ne contient pas de mots "compliqués" comme ceux de noms de plantes ou de choses comme ça, il est somme toute assez basique. Nous aurions voulu trouver un dictionnaire plus complet mais nous n'avons pas eu le temps de l'implémenter.

b) Demander à l'utilisateur de rentrer un mot de 5 lettres : **mot.c**

On demande à l'utilisateur de rentrer le mot qu'il veut tenter à l'aide de "scanf". Dans ce code on vérifie directement que le mot fait bien 5 lettres, avec la fonction **veriftaillemot.c**. Ce programme renvoie un message d'erreur si le mot ne fait pas la bonne taille.

c) Vérifier que le mot est dans le dictionnaire : **recherchedico.c**.

Tout d'abord, nous allons "charger" le dictionnaire dans un tableau. Pour cela, nous allons parcourir une première fois le dictionnaire, de manière à compter le nombre de mot qui s'y trouve. On parcourra ensuite le dictionnaire une deuxième fois, et on mettra chacun des mots dans une des cases d'un tableau.
Une fois le dictionnaire chargé dans le tableau, il suffira de parcourir ce dernier item par item pour vérifier que le mot saisi appartient bien au dictionnaire.

d) Vérifier pour chaque lettre si elle fait partie du mot et si elle est à la bonne place : **verifilettre.c**.

L'idée du programme est de remplacer chaque lettre par un -1 si elle est à la bonne place, par un -2 si elle se trouve dans le mot mais pas à la bonne place, on encore de laisser le ASCII. Ensuite, selon le numéro qui leur est attribué, les lettres seront affichées en vert si à la bonne place, en jaune si dans le mot à la mauvaise place, ou en gris si elle n'appartiennent pas au mot recherché.
La difficulté que nous avons rencontré dans cette étape est qu'au départ, le programme ne prenait pas en compte le nombre de fois où la lettre apparaissait dans le mot. Par exemple, si le mot à deviner est

le mot "monde" et que je tente le mot "echec", les 2 "e" ressortaient en jaune alors qu'il n'y a qu'un "e" dans le mot "monde".

e) Enfin, il fallait faire en sorte que la séquence se répète jusqu'à ce que le joueur trouve le bon mot ou arrive au bout de ses 6 essais et que si ce n'était pas le cas, s'il ne trouvait pas le mot au bout des 6 essais, celui-ci s'affiche avec un message de fin de partie. Pour cela nous avons utilisé des **break**, qui nous faisaient sortir de la boucle lorsqu'il le fallait (avec un message "bravo" si le mot était trouvé, ou avec un message "dommage" si ce n'était pas le cas).

3 Etape 2 : Programmation de l'ordinateur

Dans cette étape, nous souhaitons faire jouer un ordinateur de la meilleure manière possible. Pour cela, nous allons d'abord entrer un mot suscpeptible d'avoir des letters qui reviennent souvent (tarie), puis ensuite affiner le dictionnaire.
Notre résonnement afin d'affiner le dictionnaire est le suivant :

Tout d'abord, nous pouvions regarder quelles lettres étaient vertes, et supprimer les mots dans le dictionnaire copié qui ne possèdent pas ces lettres à cet emplacement. Puis, la deuxième idée était de cette fois se concentrer sur les lettres grises, et d'éliminer tous les mots qui possèdent ces lettres.

Nous avons donc implémenter les deux méthodes afin de garder celle des deux qui serait la plus rapide/la moins coûteuse, à quel point on peut réduire notre dictionnaire. Il s'avère que la plus efficace est celle où on travaille à partir des lettres grises (celles qui n'y sont pas). Mais de toute évidence, la méthode la plus performante reste celle où on allie les deux.

Explication globale du code de résolution :

L'idée de ce programme est de garder le dictionnaire (la liste) et de remplacer les mots qui s'y trouvent par des true et des false. De cette manière on ne doit pas créer de nouvelle liste à chaque fois avec les mots possibles et on gagne du temps et de la simplicité. C'est ce que fait le programme **maj**.

Le programme **ordi** lui se contente de tirer au sort l'un des mots possibles restants. C'est pourquoi il doit tirer un nombre aléatoire i entre 0 et npossibles (avec la fonction rand). Mais au vue de la manière dont nous avons procédé, il ne suffira pas par la suite de choisir comme tentative le ième mot du dico. Il faudra en effet, prendre le ième "True".
Un fois que ceci est pris en compte, le programme choisi donc bien un mot dans la liste des mots possibles et le propose.

4 Conclusion

Ainsi, nous pouvons affirmer que ce projet nous a apporté de nouvelles compétences. En effet, il nous a permis de remettre en oeuvre des méthodes sollicitées dans les cours d'informatique précédents (usage des malloc, des pointeurs). Aussi, c'était la première fois que nous devions coder la réponse intelligente d'un ordinateur, et bien que nous n'ayons pas eu le temps d'approfondir plus la théorie de l'information, c'était très intéressant pour nous d'en prendre connaissance et de connaitre "les dessous" de ces fonctionnements.