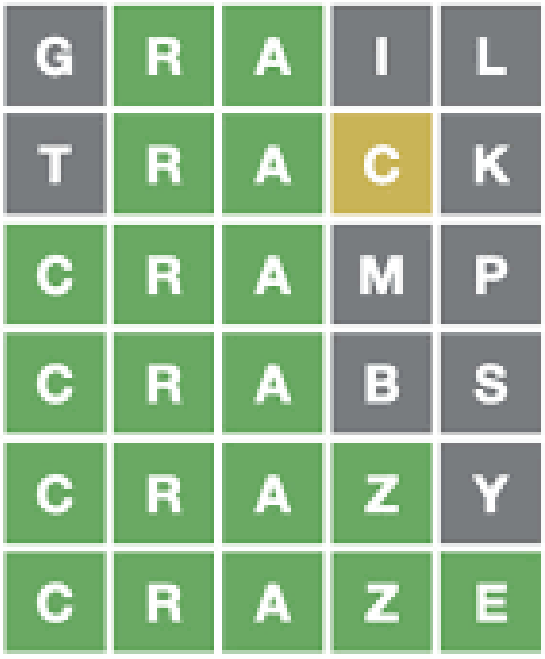
WORDLE SOLVER PROJECT

Algorithmique & Langage C



Réalisé par :

Bousbaa Tadj El Baha Lyna
Boutine Ikram
Gharbi Aicha

Classe : **L2 ISIL C G1**

Année universitaire : **2025 – 2026**

1 Introduction

Le mini-projet Wordle en langage C a pour objectif d'appliquer des concepts algorithmiques à travers un contexte pratique, en réalisant à la fois une version jouable du jeu et un solveur intelligent. Le solveur doit deviner un mot secret de cinq lettres en utilisant uniquement le retour fourni après chaque tentative, ce qui nécessite une stratégie efficace et des structures de données adaptées.

Au-delà de l'implémentation, le projet met également l'accent sur l'analyse :

- comprendre comment le solveur réduit le dictionnaire de mots possibles, comment cela influence les performances
- ainsi que le comportement global de l'algorithme en termes de complexité temporelle et spatiale.

Ce rapport résume les idées essentielles de notre approche, explique les choix de conception et illustre l'implémentation à travers des exemples de code documentés.

2 Code du programme

```c
/* ====================== WordleGame.h ======================== */
#ifndef WORDLEGAME_H_INCLUDED
#define WORDLEGAME_H_INCLUDED

#define WORD_LEN 6
#define MAX_ATTEMPTS 6
#define MAX_WORDS 1000000

//PART1:WORDLE GAME
int read_dictoinary(char words[][WORD_LEN], char *filename);
void feedback(char guess[], char word[], char fb[]);

void play_game();

//PART2:SOLVER
int match_feedback(char word[], char guess[], char fb[]);
void filter_words(char all_words[][WORD_LEN], int all_count, char
    guess[], char FB[], char new_words[][WORD_LEN], int *new_count
   );

void play_solver();

#endif

/* ========================= Wordle.c========================== */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "wordleGame.h"
```

```
29
30  char Word[MAX_WORDS][WORD_LEN];
31
32 // fonction pour lire des mots d un fichier
33 int read_dictoinary(char Words[][WORD_LEN], char *filename){
34     FILE *f = fopen(filename , "r");
35     if( f == NULL){
36         printf("erreur!!");
37         return 0;
38     }
39
40     int count = 0;
41
42     while(fscanf(f , "%5s", Words[count]) != EOF){
43
44         // convertir en minuscules
45         for(char *p = Words[count]; *p; p++){
46             if(*p >= 'A' && *p <= 'Z'){
47                 *p += 32;
48             }
49         }
50
51         count++;
52         if(count >= MAX_WORDS){
53             break;
54         }
55     }
56
57     fclose(f);
58     return count;
59 }
60
61 // Version avec affichage
62 void feedback(char guess[], char word[], char fb[]) {
63     // Tableau pour marquer les lettres du secret d  j   "
       utilis es"
64     int used[5] = {0};  // 0 = non utilis , 1 = utilis
65
66     // D'abord, marquer les lettres exactes (G)
67     for (int i = 0; i < 5; i++) {
68         if (guess[i] == word[i]) {
69             fb[i] = 'G';
70             used[i] = 1;  // Cette lettre du secret est utilis e
71         } else {
72             fb[i] = '_';  // Initialisation temporaire
73         }
74     }
75
76     // Ensuite, marquer les lettres pr sentes mais mal plac es
       (Y)
77     for (int i = 0; i < 5; i++) {
```

```
78          if (fb[i] != 'G') {  // Si pas d j  G
79              for (int j = 0; j < 5; j++) {
80                  // Si la lettre existe dans le secret ET n'a pas
                       d j    t   utilis e
81                  if (guess[i] == word[j] && !used[j]) {
82                      fb[i] = 'Y';
83                      used[j] = 1;  // Marquer comme utilis e
84                      break;
85                  }
86              }
87              // Si aucune correspondance trouv e , reste '_'
88          }
89      }
90
91      fb[5] = '\0';    // terminer la cha ne
92      printf("Feedback: %s\n", fb);
93  }
94
95  void play_game() {
96      int count;
97
98      // Charger le dictionnaire
99      count = read_dictoinary(Word, "words.txt");
100
101     if (count == 0) {
102         printf("Erreur : dictionnaire vide.\n");
103         return;
104     }
105
106     // Choisir un mot secret al atoire
107     srand(time(NULL));
108     char target[WORD_LEN];
109     strcpy(target, Word[rand() % count]);
110
111     printf("====== WORDLE GAME ======\n");
112     printf("Vous avez %d tentatives pour deviner le mot secret.\n
            ", MAX_ATTEMPTS);
113
114     char fb[6];  // tableau pour stocker le feedback
115
116     for (int attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
117         char guess[WORD_LEN];
118         printf("Tentative %d : ", attempt + 1);
119         scanf("%5s", guess);
120
121         // V rifier que le mot fait exactement 5 caract res
122         if (strlen(guess) != 5) {
123             printf("Erreur : le mot doit faire exactement 5
                       lettres.\n");
124             attempt--;  // ne pas compter cette tentative
125             continue;
```

```
126         }
127
128         // Calcul et affichage du feedback
129         feedback(guess, target, fb);
130
131         // V rification du mot
132         if (strcmp(target, guess) == 0) {
133             printf("BRAVO␣!␣Vous␣avez␣trouv ␣le␣mot␣'%s'\n",
                    target);
134             return;
135         }
136     }
137
138     printf("PERDU␣!␣Le␣mot␣etait␣:␣%s\n", target);
139 }
140
141 /* ========================= solver.c========================== */
142
143
144 #include<stdio.h>
145 #include<stdlib.h>
146 #include<string.h>
147 #include<time.h>
148 #include"WordleGame.h"
149
150 char Word[MAX_WORDS][WORD_LEN];
151 // mots filtr s
152 char Filtered[MAX_WORDS][WORD_LEN];
153
154
155 int match_feedback(char word[], char guess[], char fb[]) {
156     char fb_actual[6];
157     int used[5] = {0};
158
159     // G
160     for (int i = 0; i < 5; i++) {
161         if (guess[i] == word[i]) {
162             fb_actual[i] = 'G';
163             used[i] = 1;
164         } else {
165             fb_actual[i] = '_';
166         }
167     }
168
169     // Y
170     for (int i = 0; i < 5; i++) {
171         if (fb_actual[i] != 'G') {
172             for (int j = 0; j < 5; j++) {
173                 if (guess[i] == word[j] && !used[j]) {
174                     fb_actual[i] = 'Y';
175                     used[j] = 1;
```

```
                    break;
                }
            }
        }
    }

    fb_actual[5] = '\0';

    return strcmp(fb_actual, fb) == 0;
}

void filter_words(char all_words[][WORD_LEN], int all_count,char
   guess[], char FB[], char new_words[][WORD_LEN], int *new_count)
   {
    *new_count = 0;

    for (int i = 0; i < all_count; i++) {
        if (match_feedback(all_words[i], guess, FB)) {
            strcpy(new_words[*new_count], all_words[i]);
            (*new_count)++;
        }
    }
}

void play_solver() {

    int totalWords = read_dictoinary(Word, "words.txt");

    if (totalWords == 0) {
        printf("Erreur : dictionnaire vide.\n");
        return;
    }

    int possibleCount = totalWords;

    // Choisir mot secret
    srand(time(NULL));
    char secret[WORD_LEN];
    strcpy(secret, Word[rand() % totalWords]);

    printf("===== SOLVER WORDLE =====\n");
    printf("Mot secret : %s\n\n", secret);

    // Premier guess on prend le premier mot du fichier
    char guess[WORD_LEN];
    strcpy(guess, Word[0]);

    char fb[WORD_LEN];

    // ----- BOUCLE -----
    for (int attempt = 1; attempt <= MAX_ATTEMPTS; attempt++) {
```

```
225
226         printf("Tentative␣%d␣:␣%s\n", attempt, guess);
227
228         feedback(guess, secret, fb);
229
230         if (strcmp(guess, secret) == 0) {
231             printf("Solver␣a␣trouve␣en␣%d␣tentatives␣!\n",
                    attempt);
232             return;
233         }
234
235         // Filtrage des mots possibles
236         filter_words(Word, possibleCount, guess, fb, Filtered, &
                possibleCount);
237
238         if (possibleCount <= 0)
239             break;
240
241         strcpy(guess, Filtered[0]);
242
243         // Copier Filtered     Word pour continuer    filtrer
244         for (int i = 0; i < possibleCount; i++){
245             strcpy(Word[i], Filtered[i]);
246     }
247     }
248
249     printf("Solver␣n␣a␣pas␣trouve\n");
250 }
251 /* ========================= main.c========================== */
252 #include <stdio.h>
253 #include <stdlib.h>
254 #include"WordleGame.h"
255
256 int main() {
257     int choice;
258
259     printf("==================================\n");
260     printf("␣␣␣␣␣␣␣␣␣WORDLE␣SOLVER␣␣␣␣␣␣␣\n");
261     printf("==================================\n");
262     printf("1.␣Jouer␣au␣Wordle␣(mode␣manuel)\n");
263     printf("2.␣Solver␣automatique\n");
264     printf("3.␣Quitter\n");
265
266     printf("Votre␣choix:␣");
267     scanf("%d", &choice);
268
269     switch(choice) {
270         case 1:
271             play_game();
272             break;
273         case 2:
```

```
274             play_solver();
275             break;
276         case 3:
277             printf("Au revoir!\n");
278             break;
279         default:
280               printf("Choix invalide!\n");
281     }
282 
283     return 0;
284 }
```

3 Étude du Solver Wordle

3.1 Description de la stratégie

3.1.1 Stratégie de sélection des mots

Le solveur repose sur une stratégie d'élimination progressive par contraintes. Initialement, tous les mots du dictionnaire sont considérés comme des solutions possibles. Après chaque tentative, le feedback obtenu est utilisé pour éliminer les mots incompatibles.

Le prochain mot proposé est toujours choisi parmi les mots restant possibles, garantissant la cohérence avec les informations déjà acquises.

3.1.2 Utilisation du feedback

Le feedback est représenté par trois symboles :

- **G** : lettre correcte et bien positionnée,
- **Y** : lettre présente mais mal positionnée,
- **-** : lettre absente du mot.

Ces symboles sont utilisés comme contraintes logiques pour réduire progressivement l'espace de recherche.

3.1.3 Efficacité de l'approche

Cette méthode est efficace car chaque feedback réduit significativement le nombre de candidats possibles. En pratique, le solveur converge vers la solution en un nombre réduit de tentatives, généralement inférieur ou égal à six.

3.2 Justification des structures de données

3.2.1 Structures utilisées

Le programme utilise principalement :

- des tableaux statiques de chaînes de caractères pour stocker les mots,
- des variables entières pour suivre le nombre de mots valides.

Ces structures sont adaptées à la manipulation de mots de longueur fixe.

3.2.2 Alternatives envisagées

Les listes chaînées ou structures dynamiques ont été envisagées, mais leur gestion aurait introduit une complexité inutile dans ce contexte.

3.2.3 Comment les choix effectués soutiennent la stratégie

L'utilisation de tableaux permet :

- un parcours simple et rapide des mots.
- une mise à jour directe de la liste des mots possibles.
- une implémentation claire du filtrage logique.

Ces choix renforcent l'efficacité de la stratégie de réduction progressive de l'espace de recherche.

3.3 Analyse de la complexité

3.3.1 Complexité temporelle

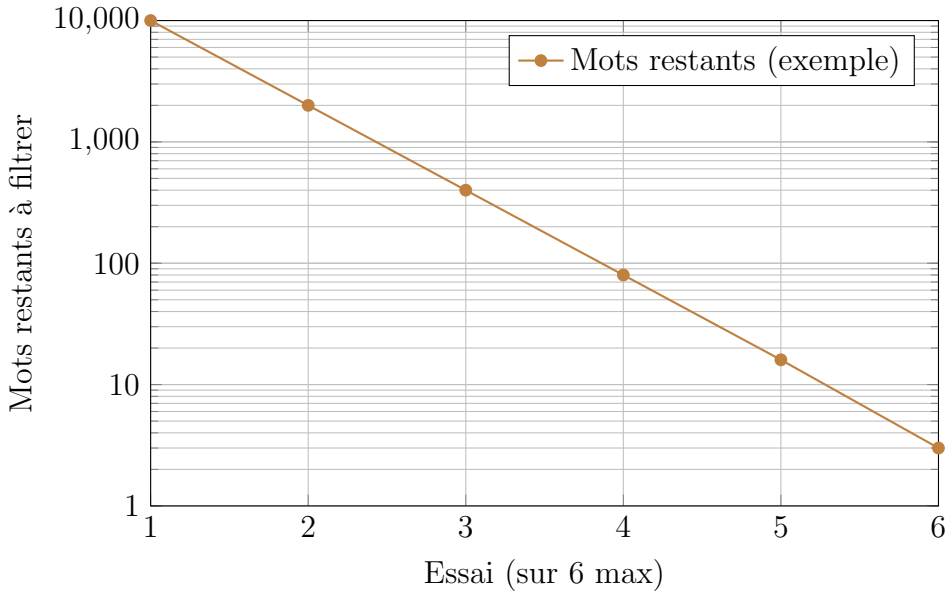
Soit N le nombre de mots dans le dictionnaire :

- Génération du feedback : $O(1)$,
- Filtrage des mots : $O(N)$ par tentative,
- Complexité totale : $O(N)$ (6 tentatives maximum).

3.3.2 Complexité spatiale

La complexité mémoire est en $O(N)$, due au stockage du dictionnaire et des mots filtrés.

3.3.3 Analyse des performances selon la taille du dictionnaire



3.4 Documentation du code

code du wordle

Fonction read_dictoinary

```c
int read_dictoinary(char Words[][WORD_LEN], char *filename){
    FILE *f = fopen(filename , "r");
    if( f == NULL){
        printf("erreur!!");
        return 0;
    }

    int count = 0;

    while(fscanf(f , "%5s", Words[count]) != EOF){

        for(char *p = Words[count]; *p; p++){
            if(*p >= 'A' && *p <= 'Z'){
                *p += 32;
            }
        }
        count++;
        if(count >= MAX_WORDS){
            break;
        }
    }

    fclose(f);
    return count;
}
```

Rôle :
Cette fonction charge le dictionnaire des mots à partir d'un fichier texte. Elle constitue la base de données du solver, car toutes les tentatives et filtrages reposent sur les mots chargés par cette fonction.

Paramètres :

- `Words` : tableau de stockage,
- `filename` : nom du fichier des mots .

Retour : Nombre de mots lus ou `0` en cas d'erreur.

Fonction feedback

```c
void feedback(char guess[], char word[], char fb[]) {

    int used[5] = {0};

    for (int i = 0; i < 5; i++) {
        if (guess[i] == word[i]) {
```

```
            fb[i] = 'G';
            used[i] = 1;
        } else {
            fb[i] = '_';
        }
    }
```

Rôle :

La fonction feedback compare un mot proposé avec le mot secret et génère un code de retour (G, Y, -). Elle est essentielle au solver, car le filtrage des mots possibles dépend directement du feedback produit.

Paramètres :

- `guess` : mot proposé.
- `used` : tableau indiquant les lettres du mot secret déjà utilisée.
- `word` : mot secret.
- `fb` : feedback généré.

Fonction play_game

```
void play_game() {
    int count;
    count = read_dictoinary(Word, "words.txt");

    if (count == 0) {
        printf("Erreur : dictionnaire vide.\n");
        return;
    }

    srand(time(NULL));
    char target[WORD_LEN];
    strcpy(target, Word[rand() % count]);

    printf("====== WORDLE GAME ======\n");
    printf("Vous avez %d tentatives pour deviner le mot secret.\n
       ", MAX_ATTEMPTS);
    char fb[6];

    for (int attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
        char guess[WORD_LEN];
        printf("Tentative %d : ", attempt + 1);
        scanf("%5s", guess);

        if (strlen(guess) != 5) {
            printf("Erreur : le mot doit faire exactement 5
               lettres.\n");
            attempt--;
            continue;
        }
```

```
28
29          feedback(guess, target, fb);
30
31          if (strcmp(target, guess) == 0) {
32              printf("BRAVO ! Vous avez trouv le mot '%s'\n",
                    target);
33              return;
34          }
35      }
36      printf("PERDU ! Le mot etait : %s\n", target);
37  }
```

Rôle : Gère le jeu Wordle en mode manuel en permettant à l'utilisateur de proposer des mots, d'obtenir le feedback correspondant et de tenter de deviner le mot secret en un nombre limité de tentatives.

code du solver

Fonction match_feedback

```
1   int match_feedback(char word[], char guess[], char fb[]) {
2       char fb_actual[6];
3       int used[5] = {0};
4
5       // G
6       for (int i = 0; i < 5; i++) {
7           if (guess[i] == word[i]) {
8               fb_actual[i] = 'G';
9               used[i] = 1;
10          } else {
11              fb_actual[i] = '_';
12          }
13      }
14      // Y
15      for (int i = 0; i < 5; i++) {
16          if (fb_actual[i] != 'G') {
17              for (int j = 0; j < 5; j++) {
18                  if (guess[i] == word[j] && !used[j]) {
19                      fb_actual[i] = 'Y';
20                      used[j] = 1;
21                      break;
22                  }
23              }
24          }
25      }
26
27      fb_actual[5] = '\0';
28
29      return strcmp(fb_actual, fb) == 0;
30  }
```

Rôle : Cette fonction vérifie si un mot candidat est compatible avec un feedback donné. Elle permet au solver de tester si un mot peut encore être une solution possible.

Retour :

- `1` : compatible,
- `0` : non compatible.

Fonction filter_words

```
void filter_words(char all_words[][WORD_LEN], int all_count,char
    guess[], char FB[], char new_words[][WORD_LEN], int *new_count)
    {
     *new_count = 0;

    for (int i = 0; i < all_count; i++) {
        if (match_feedback(all_words[i], guess, FB)) {
            strcpy(new_words[*new_count], all_words[i]);
            (*new_count)++;
        }
    }
}
```

Rôle : Cette fonction élimine les mots impossibles après chaque tentative. Elle réduit progressivement l'espace de recherche du solver.

Fonction play_solver

```
void play_solver() {

    int totalWords = read_dictoinary(Word, "words.txt");

    if (totalWords == 0) {
        printf("Erreur : dictionnaire vide.\n");
        return;
    }

    int possibleCount = totalWords;

    // Choisir mot secret
    srand(time(NULL));
    char secret[WORD_LEN];
    strcpy(secret, Word[rand() % totalWords]);

    printf("===== SOLVER WORDLE =====\n");
    printf("Mot secret : %s\n\n", secret);

    char guess[WORD_LEN];
    strcpy(guess, Word[0]);

    char fb[WORD_LEN];
```

```
24
25      // ----- BOUCLE -----
26      for (int attempt = 1; attempt <= MAX_ATTEMPTS; attempt++) {
27
28          printf("Tentative␣%d␣:␣%s\n", attempt, guess);
29
30          feedback(guess, secret, fb);
31
32          if (strcmp(guess, secret) == 0) {
33              printf("Solver␣a␣trouve␣en␣%d␣tentatives␣!\n",
                    attempt);
34              return;
35          }
36
37          filter_words(Word, possibleCount, guess, fb, Filtered, &
                possibleCount);
38
39          if (possibleCount <= 0)
40              break;
41
42          strcpy(guess, Filtered[0]);
43
44          for (int i = 0; i < possibleCount; i++){
45              strcpy(Word[i], Filtered[i]);
46      }
47      }
48
49      printf("Solver␣n␣a␣pas␣trouve\n");
50  }
```

Rôle : Cette fonction coordonne toutes les autres fonctions pour résoudre automatiquement le jeu Wordle. Elle constitue le cœur du solver(basé sur le filtrage logique.)

Fonction main

```
1   int main() {
2       int choice;
3
4       printf("==================================\n");
5       printf("␣␣␣␣␣␣␣␣␣WORDLE␣SOLVER␣␣␣␣␣␣␣\n");
6       printf("==================================\n");
7       printf("1.␣Jouer␣au␣Wordle␣(mode␣manuel)\n");
8       printf("2.␣Solver␣automatique\n");
9       printf("3.␣Quitter\n");
10
11      printf("Votre␣choix:␣");
12      scanf("%d", &choice);
13
14      switch(choice) {
15          case 1:
16              play_game();
```

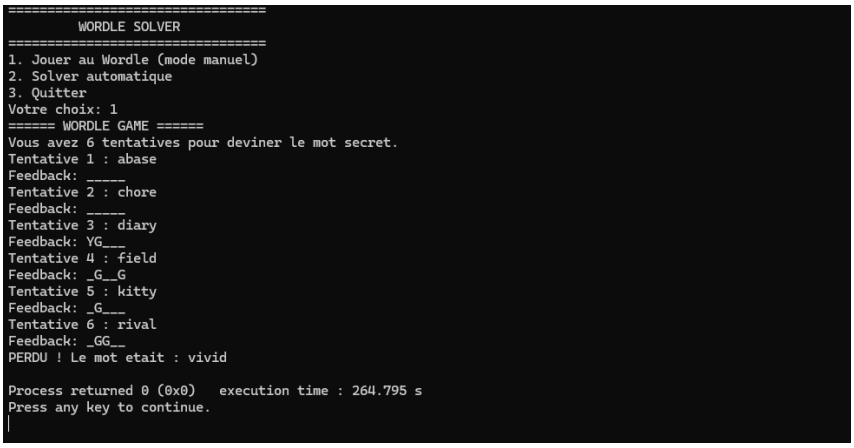
```
17              break;
18          case 2:
19              play_solver();
20              break;
21          case 3:
22              printf("Au revoir!\n");
23              break;
24          default:
25              printf("Choix invalide!\n");
26      }
27      return 0;
28  }
```

Rôle : Affiche le menu principal et redirige l'utilisateur vers le mode de jeu choisi en utilisant la boucle 'switch'.

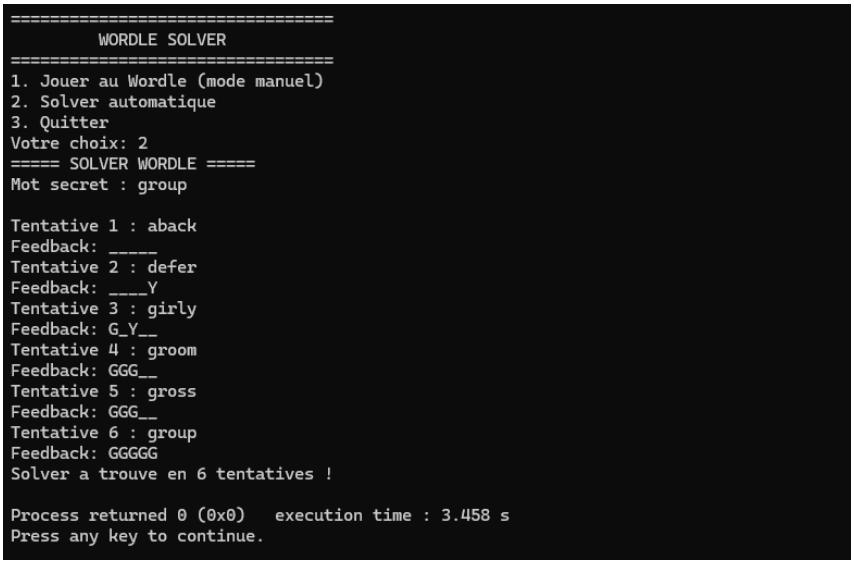
Valeur de retour : `0` en cas de fin normale du programme.

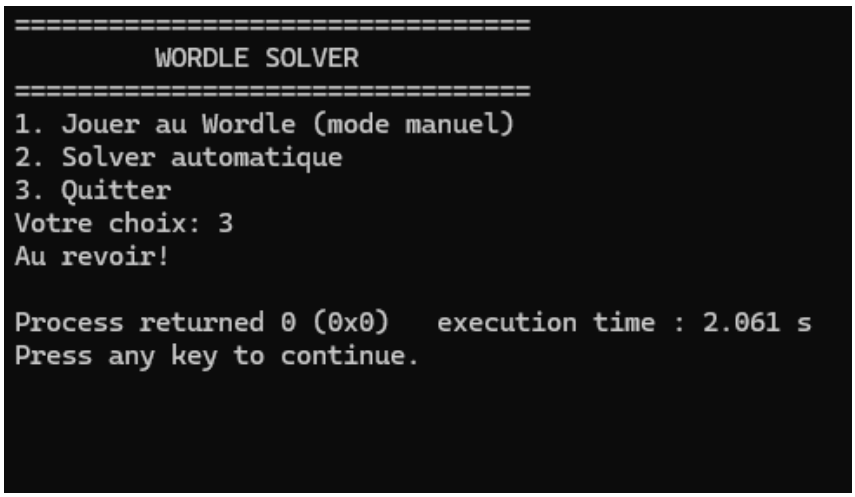
3.5 Exemples d'exécution

Exécution du Wordle



Exécution du Solver





3.6 Conclusion

Ce projet a permis de mettre en pratique les concepts d'algorithmique et de programmation en langage C à travers l'implémentation du jeu Wordle et d'un solver automatique. La stratégie de filtrage progressif s'est révélée efficace pour réduire l'espace de recherche et trouver le mot secret en un nombre limité de tentatives. Ce travail illustre l'importance du choix des structures de données et de l'analyse de complexité dans la conception d'algorithmes performants.