

# WORDLE SOLVER PROJECT

Algorithmique & Langage C

G	R	A	I	L
T	R	A	C	K
C	R	A	M	P
C	R	A	B	S
C	R	A	Z	Y
C	R	A	Z	E

**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

```
1  /* ===== WordleGame.h ===== */
2  #ifndef WORDLEGAME_H_INCLUDED
3  #define WORDLEGAME_H_INCLUDED
4
5  #define WORD_LEN 6
6  #define MAX_ATTEMPTS 6
7  #define MAX_WORDS 1000000
8
9  //PART1:WORDLE GAME
10 int read_dictionary(char words[][WORD_LEN], char *filename);
11 void feedback(char guess[], char word[], char fb[]);
12
13 void play_game();
14
15 //PART2:SOLVER
16 int match_feedback(char word[], char guess[], char fb[]);
17 void filter_words(char all_words[][WORD_LEN], int all_count, char
    guess[], char FB[], char new_words[][WORD_LEN], int *new_count
    );
18
19 void play_solver();
20
21 #endif
22
23 /* ===== Wordle.c===== */
24 #include <stdio.h>
25 #include <stdlib.h>
26 #include <string.h>
27 #include <time.h>
28 #include "wordleGame.h"
```

```

29
30 char Word[MAX_WORDS][WORD_LEN];
31
32 // fonction pour lire des mots d un fichier
33 int read_dictionary(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 "
64     // utilis es"
65     int used[5] = {0}; // 0 = non utilis , 1 = utilis
66
67     // D'abord, marquer les lettres exactes (G)
68     for (int i = 0; i < 5; i++) {
69         if (guess[i] == word[i]) {
70             fb[i] = 'G';
71             used[i] = 1; // Cette lettre du secret est utilis e
72         } else {
73             fb[i] = '_'; // Initialisation temporaire
74         }
75     }
76
77     // Ensuite, marquer les lettres pr sentes mais mal plac es
78     // (Y)
79     for (int i = 0; i < 5; i++) {

```

```

78         if (fb[i] != 'G') { // Si pas de j G
79             for (int j = 0; j < 5; j++) {
80                 // Si la lettre existe dans le secret ET n'a pas
81                 // de j t utilis e
82                 if (guess[i] == word[j] && !used[j]) {
83                     fb[i] = 'Y';
84                     used[j] = 1; // Marquer comme utilis e
85                     break;
86                 }
87             }
88             // Si aucune correspondance trouv e, reste '_'
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_dictionary(Word, "words.txt");
100
101     if (count == 0) {
102         printf("Erreur: dictionnaire vide.\n");
103         return;
104     }
105
106     // Choisir un mot secret alatoire
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",
113           MAX_ATTEMPTS);
114
115     char fb[6]; // tableau pour stocker le feedback
116
117     for (int attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
118         char guess[WORD_LEN];
119         printf("Tentative %d: ", attempt + 1);
120         scanf("%5s", guess);
121
122         // V rifier que le mot fait exactement 5 caract res
123         if (strlen(guess) != 5) {
124             printf("Erreur: le mot doit faire exactement 5
125                 lettres.\n");
126             attempt--; // ne pas compter cette tentative
127             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",
134             target);
135         return;
136     }
137
138     printf("PERDU! Le mot etait: %s\n", target);
139 }
140
141 /* ===== solver.c ===== */
142
143 #include<stdio.h>
144 #include<stdlib.h>
145 #include<string.h>
146 #include<time.h>
147 #include"WordleGame.h"
148
149 char Word[MAX_WORDS][WORD_LEN];
150 // mots filtr s
151 char Filtered[MAX_WORDS][WORD_LEN];
152
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;

```

```

176         break;
177     }
178 }
179 }
180 }
181
182 fb_actual[5] = '\0';
183
184 return strcmp(fb_actual, fb) == 0;
185 }
186
187 void filter_words(char all_words[][WORD_LEN], int all_count, char
guess[], char FB[], char new_words[][WORD_LEN], int *new_count)
{
188     *new_count = 0;
189
190     for (int i = 0; i < all_count; i++) {
191         if (match_feedback(all_words[i], guess, FB)) {
192             strcpy(new_words[*new_count], all_words[i]);
193             (*new_count)++;
194         }
195     }
196 }
197
198 void play_solver() {
199
200     int totalWords = read_dictionary(Word, "words.txt");
201
202     if (totalWords == 0) {
203         printf("Erreur: dictionnaire vide.\n");
204         return;
205     }
206
207     int possibleCount = totalWords;
208
209     // Choisir mot secret
210     srand(time(NULL));
211     char secret[WORD_LEN];
212     strcpy(secret, Word[rand() % totalWords]);
213
214     printf("==== SOLVER WORDLE ==== \n");
215     printf("Mot secret: %s \n \n", secret);
216
217     // Premier guess on prend le premier mot du fichier
218     char guess[WORD_LEN];
219     strcpy(guess, Word[0]);
220
221     char fb[WORD_LEN];
222
223     // ----- BOUCLE -----
224     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",
232             attempt);
233         return;
234     }
235
236     // Filtrage des mots possibles
237     filter_words(Word, possibleCount, guess, fb, Filtered, &
238         possibleCount);
239
240     if (possibleCount <= 0)
241         break;
242
243     strcpy(guess, Filtered[0]);
244
245     // Copier Filtered      Word pour continuer      filtrer
246     for (int i = 0; i < possibleCount; i++){
247         strcpy(Word[i], Filtered[i]);
248     }
249
250     printf("Solver_n_a_pas_trouve\n");
251 }
252 /* ===== main.c ===== */
253 #include <stdio.h>
254 #include <stdlib.h>
255 #include "WordleGame.h"
256
257 int main() {
258     int choice;
259
260     printf("=====\n");
261     printf("WORDLE_SOLVER\n");
262     printf("=====\n");
263     printf("1. Jouer au Wordle (mode manuel)\n");
264     printf("2. Solver automatique\n");
265     printf("3. Quitter\n");
266
267     printf("Votre choix: ");
268     scanf("%d", &choice);
269
270     switch(choice) {
271         case 1:
272             play_game();
273             break;
274         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

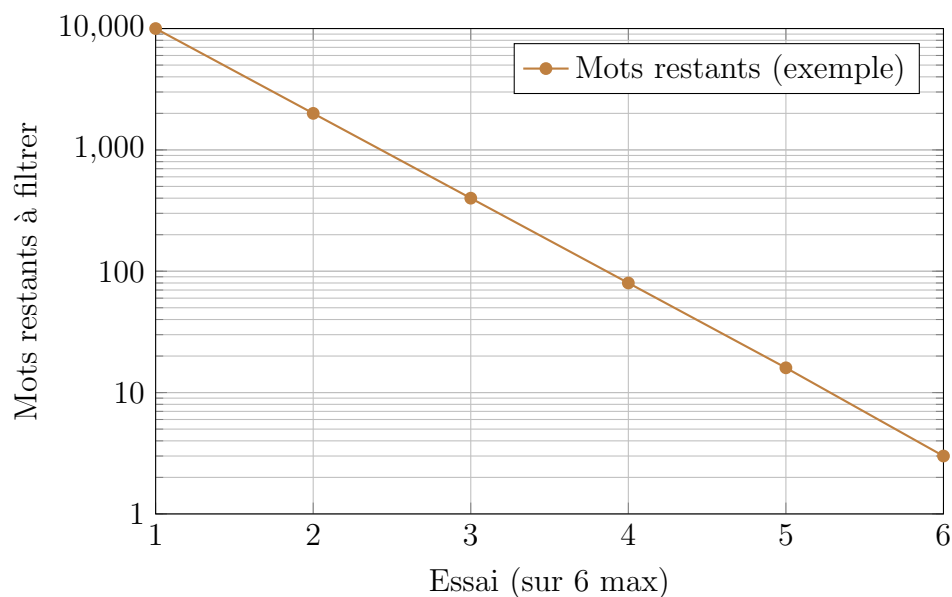
Soit  $N$  le nombre de mots dans le dictionnaire :

- 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

```
1 int read_dictoinary(char Words[][WORD_LEN], char *filename){
2     FILE *f = fopen(filename , "r");
3     if( f == NULL){
4         printf("erreur!!");
5         return 0;
6     }
7
8     int count = 0;
9
10    while(fscanf(f , "%5s", Words[count]) != EOF){
11
12        for(char *p = Words[count]; *p; p++){
13            if(*p >= 'A' && *p <= 'Z'){
14                *p += 32;
15            }
16        }
17        count++;
18        if(count >= MAX_WORDS){
19            break;
20        }
21    }
22
23    fclose(f);
24    return count;
25 }
```

#### 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

```
1 void feedback(char guess[], char word[], char fb[]) {
2
3     int used[5] = {0};
4
5     for (int i = 0; i < 5; i++) {
6         if (guess[i] == word[i]) {
```

```

7         fb[i] = 'G';
8         used[i] = 1;
9     } else {
10        fb[i] = '_';
11    }
12 }

```

### 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

```

1 void play_game() {
2     int count;
3     count = read_dictionary(Word, "words.txt");
4
5     if (count == 0) {
6         printf("Erreur : dictionnaire vide.\n");
7         return;
8     }
9
10    srand(time(NULL));
11    char target[WORD_LEN];
12    strcpy(target, Word[rand() % count]);
13
14    printf("==== WORDLE GAME =====\n");
15    printf("Vous avez %d tentatives pour deviner le mot secret.\n", MAX_ATTEMPTS);
16    char fb[6];
17
18    for (int attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
19        char guess[WORD_LEN];
20        printf("Tentative %d : ", attempt + 1);
21        scanf("%5s", guess);
22
23        if (strlen(guess) != 5) {
24            printf("Erreur : le mot doit faire exactement 5 lettres.\n");
25            attempt--;
26            continue;
27        }

```

```

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

```

**R le :** G re le jeu Wordle en mode manuel en permettant   l'utilisateur de proposer des mots, d'obtenir le feedback correspondant et de tenter de deviner le mot secret en un nombre limit  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`

```
1 void filter_words(char all_words[][WORD_LEN], int all_count, char
  guess[], char FB[], char new_words[][WORD_LEN], int *new_count)
  {
2     *new_count = 0;
3
4     for (int i = 0; i < all_count; i++) {
5         if (match_feedback(all_words[i], guess, FB)) {
6             strcpy(new_words[*new_count], all_words[i]);
7             (*new_count)++;
8         }
9     }
10 }
```

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

### Fonction `play_solver`

```
1 void play_solver() {
2
3     int totalWords = read_dictionary(Word, "words.txt");
4
5     if (totalWords == 0) {
6         printf("Erreur: dictionnaire vide.\n");
7         return;
8     }
9
10    int possibleCount = totalWords;
11
12    // Choisir mot secret
13    srand(time(NULL));
14    char secret[WORD_LEN];
15    strcpy(secret, Word[rand() % totalWords]);
16
17    printf("==== SOLVER WORDLE ==== \n");
18    printf("Mot secret: %s \n \n", secret);
19
20    char guess[WORD_LEN];
21    strcpy(guess, Word[0]);
22
23    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",
34             attempt);
35         return;
36     }
37
38     filter_words(Word, possibleCount, guess, fb, Filtered, &
39         possibleCount);
40
41     if (possibleCount <= 0)
42         break;
43
44     strcpy(guess, Filtered[0]);
45
46     for (int i = 0; i < possibleCount; i++){
47         strcpy(Word[i], Filtered[i]);
48     }
49
50     printf("Solver_n_a_pas_trouve\n");
51 }

```

**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

```
=====
WORDLE SOLVER
=====
1. Jouer au Wordle (mode manuel)
2. Solver automatique
3. Quitter
Votre choix: 1
===== WORDLE GAME =====
Vous avez 6 tentatives pour deviner le mot secret.
Tentative 1 : abase
Feedback: ____
Tentative 2 : chore
Feedback: ____
Tentative 3 : diary
Feedback: YG__
Tentative 4 : field
Feedback: _G_G
Tentative 5 : kitty
Feedback: _G__
Tentative 6 : rival
Feedback: _GG_
PERDU ! Le mot etait : vivid
Process returned 0 (0x0)   execution time : 264.795 s
Press any key to continue.
|
```

### Exécution du Solver

```
=====
WORDLE SOLVER
=====
1. Jouer au Wordle (mode manuel)
2. Solver automatique
3. Quitter
Votre choix: 2
===== SOLVER WORDLE =====
Mot secret : group

Tentative 1 : aback
Feedback: ____
Tentative 2 : defer
Feedback: ____Y
Tentative 3 : girly
Feedback: G_Y__
Tentative 4 : groom
Feedback: GGG__
Tentative 5 : gross
Feedback: GGG__
Tentative 6 : group
Feedback: GGGGG
Solver a trouve en 6 tentatives !

Process returned 0 (0x0)   execution time : 3.458 s
Press any key to continue.
```

```
=====
WORDLE SOLVER
=====
1. Jouer au Wordle (mode manuel)
2. Solver automatique
3. Quitter
Votre choix: 3
Au revoir!

Process returned 0 (0x0)   execution time : 2.061 s
Press any key to continue.
```



---

### 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 solveur 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.