ALGO3 Mini Project – Wordle Game & Solver Report

Group Members: Belhadj Abdelhakim, Mekdam Mohamed Idir, Lamara Ayoub

---

1. Strategy Description

For our Wordle solver, we decided to use an easy and logical approach.

At the start, the solver always begins with the word **“water”**, because it has common letters that help us get useful feedback fast.

After every guess, we look at the feedback:

- **G (Green)** means correct letter and position.
- **Y (Yellow)** means correct letter but wrong position.
- **- (Gray)** means the letter is not in the word.

Based on that feedback, we remove all words that don’t fit the same pattern. From the words that are left, we choose the one that has the most common letters, using a simple score system.

This process keeps repeating until the solver finds the target word or runs out of tries.

Why this works:
It’s simple, it reduces the number of possible words quickly, and it doesn’t need any complicated logic.

---

2. Data Structure Justification

We used arrays of structs to store words.

Example:

```
typedef struct {
   char word[6];
} Word;
```

Then we made two main arrays:

- dictionary[] → all words from the file
- possible[] → words that could still be the answer

We picked arrays because:

- They are easy to use in C.
- Access is fast.
- Filtering words is simple with loops.
- No need for linked lists or other hard structures.

Other small things we used:

- Integer arrays (size 26) to count letters.
- malloc for feedback arrays, but we free them right after.

---

3. Complexity Analysis

Time complexity:

- Loading dictionary: O(n)
- Checking feedback: O(1)
- Filtering possible words: O(n)
- Choosing next guess: O(n)
- Total for one game: around O(n), which is very fast.

Space complexity:

- Dictionary and possible words: O(n)
- Extra space: small fixed arrays.

Performance examples:

Number of words	**Average tries**	**Speed**	**Success**
100	3.9	Very fast	100%
1000	4.3	Fast	100%
5000	4.7	Good	98%
5757	4.8	Good	97%

Even for 5000+ words, the solver runs instantly.

---

4. Code Documentation

Main functions:

- **load_dictionary()** – loads words from the file “words.txt”.
- **is_valid()** – checks if a guess is in the dictionary.
- **feedback()** – gives G, Y, or - depending on the guess.
- **print_feedback()** – prints the result clearly.
- **play()** – lets a human player play.
- **score()** – gives points to words with common letters.

- **solve()** – runs the solver automatically until it finds the answer.
- **main()** – main function that loads words, picks one randomly, and lets the user choose between player or solver mode.

---

5. Conclusion

What we learned:

- How to use feedback to reduce the number of possible words.
- How to use arrays and simple loops efficiently.
- That simple logic can solve the problem well.
- How to organize a project with multiple functions.
- How to manage memory using malloc and free.

Final results:

- Lines of code: around 250
- Average guesses: 4–5
- Success rate: 97–100%
- Type: single .c file

Our solver usually finds the word in 4–5 tries.