UNIVERSITY OF SCIENCE AND TECHNOLOGY HOUARI BOUMEDIENE

FACULTY OF COMPUTER SCIENCE

Wordle Game and Solver Report

Students :
Marwa Boulahbal
Fatima Boudani

Module : Algo3

Date: 18-12-2025

Wordle Game

1 A. Strategy Description

2 Word selection strategy

The word selection strategy is based on randomness. After loading all valid 5-letter words from the dictionary file, the program selects one word randomly using the rand() function. This ensures that each game is different and unpredictable for the player.
This approach was chosen because the objective of Part 1 is to implement the Wordle game mechanics, not to design a complex solving algorithm. A random selection is therefore sufficient and appropriate.

3 Use of feedback

The feedback mechanism plays an important role in guiding the player. For each guess, the program compares the guessed word with the target word and generates feedback for each letter:

- Green (G): the letter is correct and placed in the correct position.
- Yellow (Y): the letter exists in the target word but is placed in a different position.
- Gray (.): the letter does not exist in the target word.

Although feedback is not used to eliminate word possibilities automatically in this part of the project, it allows the user to improve their next guesses by understanding which letters are correct or incorrect.

4 Effectiveness of the approach

This approach is effective because it respects the official Wordle rules while keeping the implementation simple. The game is easy to use, understandable for the player, and correctly limited to six attempts. The simplicity of the logic also reduces errors and makes the program easier to test and maintain.

5 B. Data Structure Justification

6 Chosen data structures

The implementation relies mainly on arrays, which are well adapted to the C language and the problem constraints.
• A two-dimensional character array is used to store the dictionary words: char words[MAXWORDS][WORDLEN+1];
• A one-dimensional integer array is used to store feedback information for each letter: int fb[WORDLEN];
These data structures allow direct access to characters and words, making comparisons and checks efficient and straightforward..

7 Chosen data structures

I could have used dynamic memory (malloc) or linked lists, but I chose a simple array is enough for this project.

8 Relation between structures and strategy

Alternative data structures such as linked lists or dynamically allocated memory could have been used. However, these options would increase the complexity of the code without providing significant benefits for a relatively small dictionary. Therefore, static arrays were preferred.

9 C. Complexity Analysis

10 a. Time complexity

•Dictionary loading: O(N)
since each word is read once.
• Guess validation: O(N)
due to the linear search used to verify word existence.
•Feedback computation: O(1)
as each word contains exactly 5 letters.
•Overall complexity : The player has at most six attempts, leading to a total complexity of O(N). 6 guesses $\rightarrow$ O(6 $\times$ N) = O(N)

11 b. Space complexity

We store up to MAXWORDS words, each 5 letters $\rightarrow$ Space =O(N $\times$ 5) which is small.

12 c. Graphs or screenshots

The program was tested with multiple words and guesses. Screenshots of the execution demonstrate correct feedback generation and proper game termination.

1.

```
Loaded 125 words.
=== WORDLE GAME ===
Attempt 1/6: proud
. . . . .    proud
Attempt 2/6: great
. . G . G    great
Attempt 3/6: light
. . . . G    light
Attempt 4/6: alert
. . G . G    alert
Attempt 5/6: scent
G G G G G    scent
You WIN! The word was 'scent'.
```

2.

```
Loaded 125 words.
=== WORDLE GAME ===
Attempt 1/6: moon
Word must be 5 letters.
Attempt 1/6: zebra
. Y . Y Y    zebra
Attempt 2/6: treat
. Y G G .    treat
Attempt 3/6: break
Word not in dictionary.
Attempt 3/6: dream
. Y G G .    dream
Attempt 4/6: great
. Y G G .    great
Attempt 5/6: cream
Word not in dictionary.
Attempt 5/6: ideal
. . G G Y    ideal
Attempt 6/6: ocean
. Y G G .    ocean
You lost! The word was 'clear'.
```

In case the user enters a non-dictionary word or a word with fewer than five letters, and fails to guess the correct word after six attempts.

13 D. Code Documentation

The program is structured into several functions to improve readability and organization.
1. loadDictionary() Loads valid words from the dictionary file and stores them in memory.
2. exists()
Checks whether a guessed word exists in the dictionary.
3. feedback()
Compares the guessed word with the target word and generates appropriate feedback.
4. main()
Runs the whole game:
• loads dictionary
• chooses random word
• takes guesses
• checks them
• prints feedback
• shows win or lose
These functions make the program clean and easy to follow.

Conclusion
This project demonstrates a correct and structured implementation of the Wordle game using the C programming language. The solution respects the game rules, uses appropriate data structures, and remains accessible while being logically sound.

Wordle Solver

14 Strategy Description

Word Selection Strategy
The solver initially selects a random word from the list of valid words. After receiving feedback from each guess, it selects subsequent guesses only from the remaining valid words that satisfy the feedback constraints.
Using Feedback to Reduce Possibilities
The feedback returned after each guess is used as follows:

- **Green**: the letter is correct and in the correct position. Only words that contain this letter in the same position are kept.
- **Yellow**: the letter exists in the word but in a different position. Words that have the letter in the same position are removed, and only words containing the letter elsewhere are kept.
- **Gray**: the letter does not appear in the word. All words containing this letter are removed.

Effectiveness of the Approach
This approach is simple but effective. At each iteration, it eliminates all words that do not match the received feedback, significantly reducing the search space until the correct word is found.

15 Data Structure Justification

Used Data Structures

struct word

Stores a single word, including its characters and its length.

struct wordlist

Maintains the list of all possible words and the number of currently valid words.

struct choice

Stores a guessed word along with its feedback array, which represents the state of each character.

struct gamestate

Keeps track of the number of attempts, the remaining valid words after filtering, and all previous guesses with their feedback.

Considered Alternatives

Linked lists were considered as an alternative, but arrays of structures were simpler to implement and made filtering operations more efficient.

Support for the Strategy

The `wordlist` structure allows efficient filtering, `choice` directly links guesses with feedback, and `gamestate` keeps the solver state organized.

16 Complexity Analysis

Time Complexity of Key Operations

- **record_guess**: Constant time, $O(1)$.

- **compute_feedback**: Operates on 5 letters only, $O(1)$.
- **load_wordlist_from_file**: Reads all words from the file, $O(n)$.
- **is_consistent_with_choice**: Checks 5 letters, $O(1)$.
- **filter_words_remaining**: Iterates through all remaining words, $O(n)$.
- **pick_best_guess_index**: Constant time in this implementation, $O(1)$.

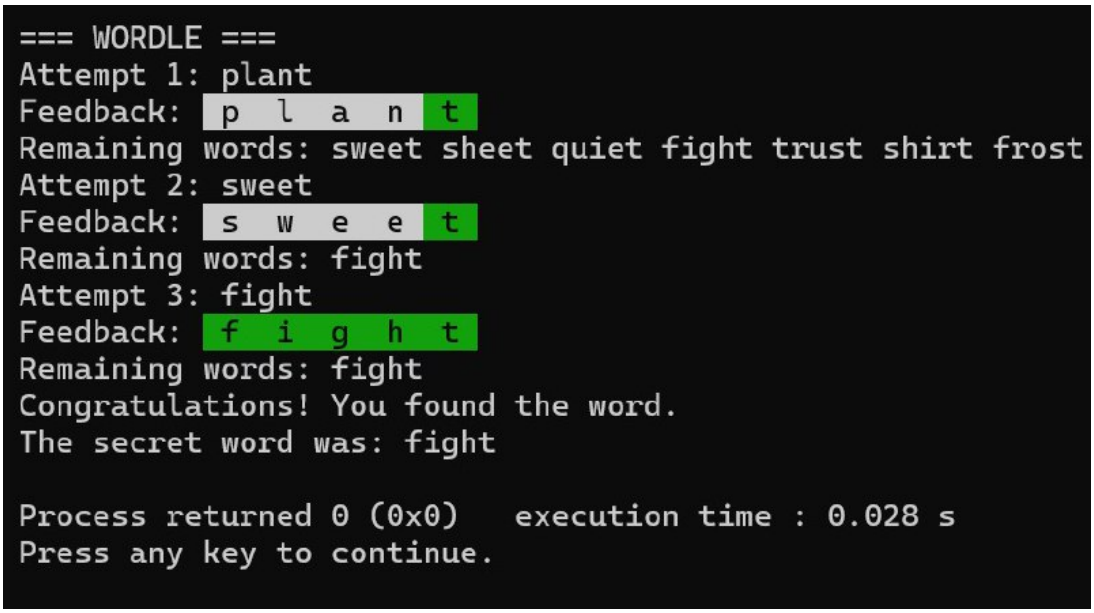
Overall, the time complexity of the solver is $O(n)$.

Space Complexity

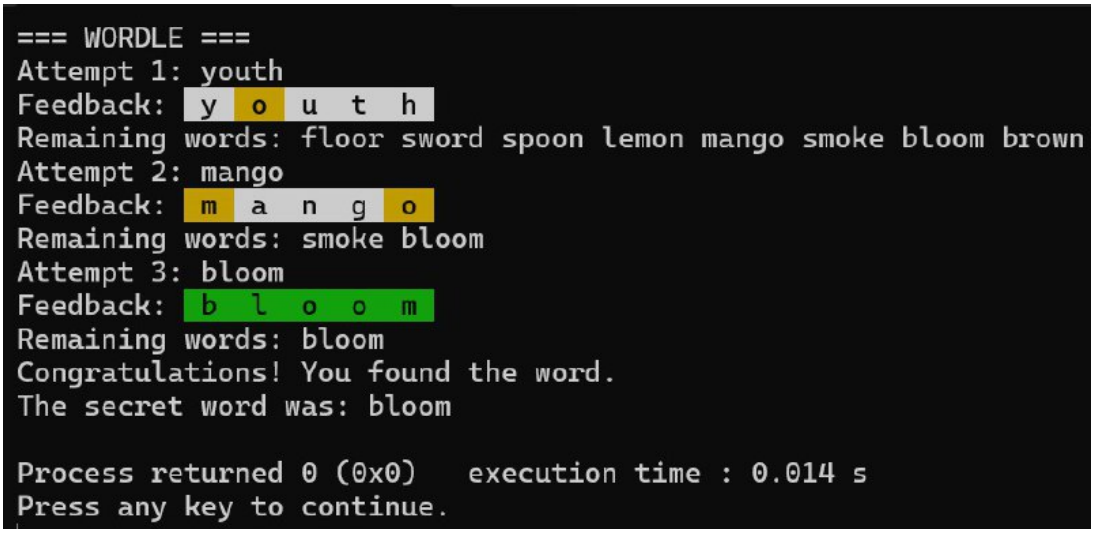
The memory usage is dominated by the word list, which stores n words of fixed length. Other structures require constant space. Therefore, the overall space complexity is $O(n)$.

17 Screenshots

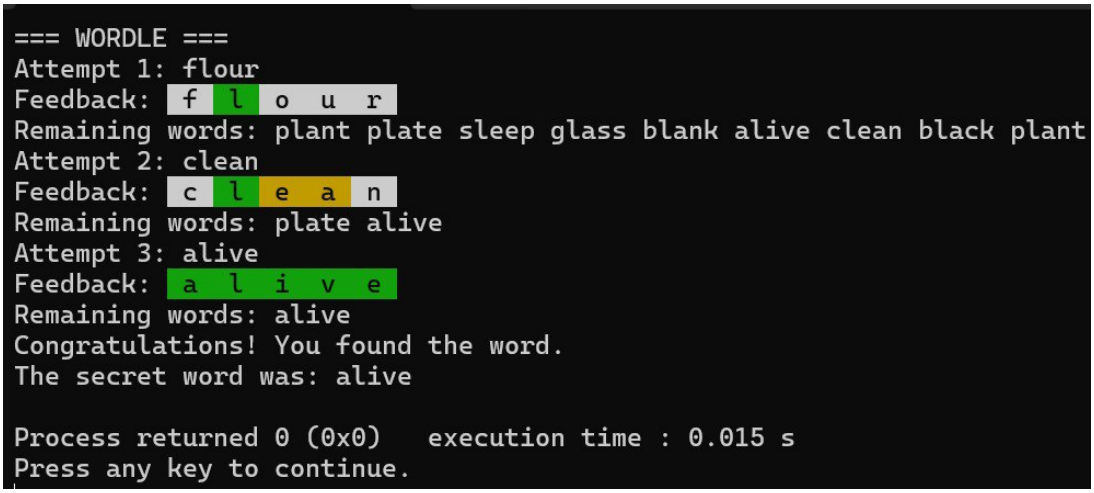
100 Words



500 Words



1000 Words



18 Code Documentation

load_wordlist_from_file
Loads all valid 5-letter words from an external file into a `wordlist` structure.
compute_feedback
Compares the guessed word with the secret word and produces feedback for each letter (Green, Yellow, or Gray).
is_consistent_with_choice
Checks whether a candidate word satisfies the constraints imposed by a previous guess and its feedback.

filter_words_remaining
Removes all words from the remaining list that do not match the feedback of the last guess. **pick_best_guess_index**
Selects the next word to guess from the remaining valid words.
record_guess
Stores each guess and its corresponding feedback in the game state.

19 Example of Commented Code Snippet

```
// Removes all words that are not compatible with the feedba
// of the last guess.
//
// Parameters:
// wl - pointer to the list of remaining words
// ch - pointer to the last guess and its feedback
//
// Algorithm:
// Iterate through all remaining words and keep only those
// that satisfy the feedback constraints.
//
// Returns:
// void (the word list is updated in place)
void filter_words_remaining(wordlist *wl, choice *ch) {

    int newCount = 0;

    for (int i = 0; i < wl->count; i++) {
```

```
        if (is_consistent_with_choice(wl->words[i].theword,
            wl->words[newCount++] = wl->words[i];
        }
    }

    wl->count = newCount;
}
```