Wordle Solver Project Report

Project name: Wordle Solver in C

Course: Algorithms & Data Structures in C

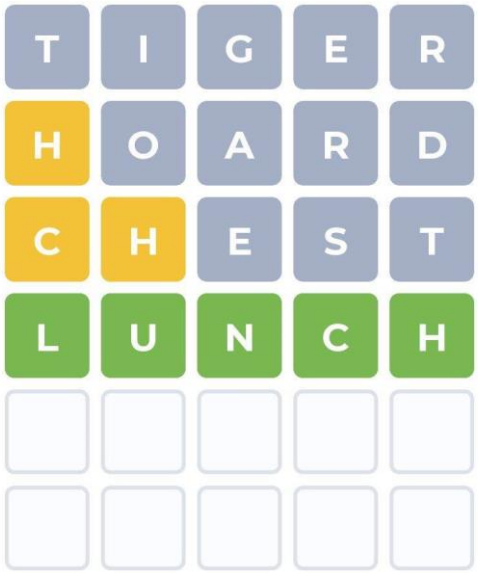
Author: Nourelhouda Bouhadja

Ikram Aktouf

Fatma Zahra Mekki

Class: Isil C g4

Date: December 2025



Introduction

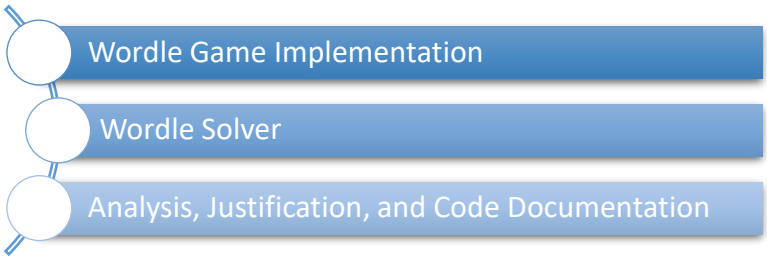
Wordle is a word-guessing game where the player has six attempts to find a secret 5-letter word.

After each guess, the game gives feedback using three symbols:

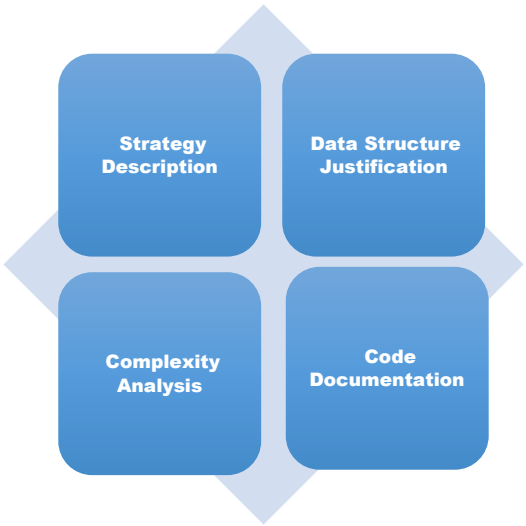
- Green: the letter is correct and in the correct position.
- Yellow: the letter is in the word but in the wrong position.
- Gray: the letter is not in the word at all.

The objective of this mini project is to implement Wordle in C and also create a solver that can play the game automatically. The solver uses guesses and feedback to reduce the list of possible words until it finds the correct answer.

So, this mini-project is organized into three main parts:



Our report will explain :



Strategy Description

Word Selection Strategy

The Wordle solver uses a progressive elimination strategy.

It starts with a fixed initial word, “arise”, which contains common English letters. This first guess helps collect maximum information about the target word.

After each attempt, the solver selects the next guess from the remaining valid candidates. The current implementation chooses the first word that satisfies all known constraints.

Use of Feedback to Eliminate Possibilities

After each guess, feedback is analyzed as follows:

~ GREEN letters indicate correct letters in correct positions and are fixed in all remaining candidates.

~ YELLOW letters indicate letters that exist in the word but must appear in different positions.

~ GRAY letters indicate letters that must not appear in any remaining candidate

Based on this feedback, the solver filters the dictionary and removes all words that do not satisfy these conditions.

Effectiveness of the Approach

This approach is effective because:

° It reduces the number of possible words after every guess.

° Each new guess is consistent with all previous feedback.

° The algorithm is simple, reliable, and converges quickly.

° The solver generally finds the correct word within the allowed six attempts.

Data Structure Justification

Data Structures Used

The main data structure used is a static array of strings:

```
char words[MAX_WORDS][WORD_LEN+1];
```

This array stores both the dictionary and the remaining candidate words.

Justification

This choice was made because:

- Arrays allow fast and direct access to elements.
- They are easy to traverse and filter.
- The maximum dictionary size is known in advance.
- Memory management is simple and safe.

Alternatives Considered

Other structures such as linked lists or trees were considered but not used because:

- They add unnecessary complexity.
- They require dynamic memory management.
- They do not significantly improve performance for this problem.

The array-based solution efficiently supports the filtering strategy used by the solver.

Complexity Analysis

Time Complexity

- **Dictionary loading**:

  Each word is read once from the file.

  Time complexity: O(N)

- **Filtering candidates:**

  Each remaining word is checked letter by letter (5 letters).

  Time complexity per attempt: O(N)

- **Next-guess selection:**

  The first valid candidate is selected.

  Time complexity: O(1)

Since the solver runs for at most 6 attempts, the overall time complexity is:

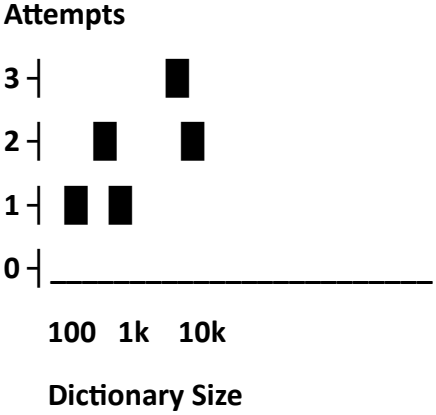
Total time complexity: O(N)

Space Complexity

- The dictionary and candidate list are stored in a fixed-size array.
- No additional large data structures are used.

Space complexity: O(N)

Experimental Observations



X-axis: Dictionary size (100, 1,000, 10,000)

Y-axis: Average number of attempts

Bars show how the solver slightly increases in attempts as dictionary grows.

Code Documentation

Function and Module Descriptions

• load_dictionary()

Purpose: Loads all valid 5-letter words from a text file into an array.

Role in solver: Provides the initial list of possible candidate words for the game.

• print_colored_feedback() / give_feedback()

Purpose: Displays feedback for a guessed word using colored boxes (green, yellow, gray).

Role in solver: Helps visualize which letters are correct, misplaced, or absent in the target word.

• filter_candidates()

Purpose: Removes words from the candidate list that do not satisfy the feedback rules.

Role in solver: Reduces the search space by eliminating impossible words after each guess.

Example snippet**:**

```c
// Filters candidate words based on Wordle feedback
// Parameters:
//   words  : array of current candidate words
//   count  : number of candidate words
//   guess  : the guessed word
//   target : the secret word
// Return value:
//   Number of valid candidate words remaining
int filter_candidates(char words[][WORD_LEN+1], int count,
                      const char *guess, const char *target) {
    int new_count = 0;
    for (int i = 0; i < count; i++) {
        int valid = 1;
        for (int j = 0; j < WORD_LEN; j++) {
            if (guess[j] == target[j] && words[i][j] != guess[j])
                valid = 0;
            if (strchr(target, guess[j]) && words[i][j] == guess[j]
                && guess[j] != target[j])
                valid = 0;
            if (!strchr(target, guess[j]) && strchr(words[i], guess[j]))
                valid = 0;
        }
        if (valid) {
            strcpy(words[new_count], words[i]);
            new_count++;
        }
    }
    return new_count;
}
```

This snippet demonstrates how the solver applies feedback to remove invalid words. Parameters, return value, and logic are explained in the comments

• select_next_guess()

Purpose: Chooses the next word to guess from the remaining candidates.

Role in solver: Ensures that each new guess is consistent with all previous feedback.

• **solver()**

Purpose: Controls the Wordle-solving process by iterating guesses and updating candidates.

Role in solver: Implements the main algorithm loop until the target word is found or maximum attempts are reached.

Example of Commented Code

```
// Check each letter of the candidate word
for (int j = 0; j < WORD_LEN; j++) {

    // Green case: letter is correct and in correct position
    if (guess[j] == target[j] && words[i][j] != guess[j])
        valid = 0;

    // Yellow case: letter exists but in wrong position
    if (strchr(target, guess[j]) && words[i][j] == guess[j]
        && guess[j] != target[j])
        valid = 0;

    // Gray case: letter does not exist in the word
    if (!strchr(target, guess[j]) && strchr(words[i], guess[j]))
        valid = 0;
}
```

This snippet shows how the solver checks each letter of a candidate word against the target and applies green, yellow, and gray rules. Parameters and return values are explained in comments.

The Wordle solver project was successfully implemented in C. All functions work together to guess the target word efficiently, and the program demonstrates how feedback can be used to narrow down possibilities. This project illustrates the application of algorithms, data structures, and modular programming in a practical problem.

{In the end, we sincerely thank our teacher for their guidance and for providing us with this learning opportunity}