ALGO3 Mini Project Assignment

Wordle solver

Author: *ABADLI Badreddine*
Course: *Algorithms & Data Structures in C*

Assigned: **01/11/2025** Due: **08/12/2025**

Introduction

Wordle is a word-guessing game where a player has six attempts to discover a secret 5-letter word. After each guess the game returns feedback:

- **Green:** letter correct and in the correct position.
- **Yellow:** letter appears in the word but in a different position.
- **Gray:** letter does not appear in the word.

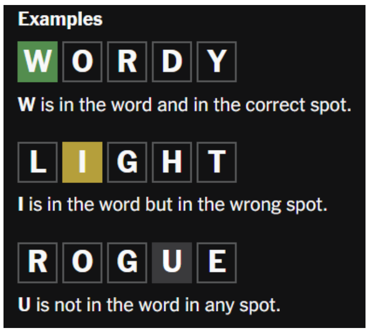
Examples
W O R D Y
W is in the word and in the correct spot.
L I G H T
I is in the word but in the wrong spot.
R O G U E
U is not in the word in any spot.

Figure 1: Wordle Game Interface

If you want to try Wordle in your browser, see the official game: NYTimes Wordle.

Project Objective

You will implement two components in `C`:

1. **Wordle Game:** selects a random 5-letter target word and accepts guesses (from a human or automated solver), returning feedback after each guess.
2. **Wordle Solver:** an automated program that plays Wordle and attempts to guess the target word in **as few attempts as possible**.

A GUI is not required; command-line interaction is sufficient.

Requirements

Part 1 – Wordle Game Implementation (30%)
Implement a Wordle game with the following features:

- Read a dictionary of valid 5-letter words from a `.txt` file (one word per line).
- Select a random target word from the dictionary.
- Accept guesses from the user or solver.
- Validate that a guess is a 5-letter word present in the dictionary.
- Provide feedback for each guess (correct position, wrong position, not in word).
- Track attempts and declare win/loss after 6 guesses.

Part 2 – Wordle Solver (50%)
Implement an automated solver that plays Wordle. The solver requires algorithmic design:

- Design a strategy to select guesses that efficiently narrow down possibilities.
- Choose suitable data structures to implement the strategy.
- Implement the solver to consistently solve Wordle puzzles.

The solver must:

- Start with no prior information about the target word.
- Make guesses and use feedback to eliminate impossible words.
- Converge on the correct answer in **as few attempts as possible**.

Part 3 – Analysis, Justification, and Code Documentation (20%)

Provide a written analysis (submit as a separate PDF) that includes the following sections:

a. **Strategy Description**
    - What is your word selection strategy?
    - How is feedback used to eliminate possibilities?
    - Why is your approach effective?

b. **Data Structure Justification**
    - Which data structures did you use and why?
    - What alternatives did you consider?
    - How do your choices support your strategy?

c. **Complexity Analysis**
    - Analyze the time complexity of key operations (e.g., filtering, next-guess selection).
    - Analyze the space complexity (memory usage).
    - Provide graphs or screenshots on various scenarios (100 words, 1000 words ...).

d. **Code Documentation**
    - For each major function or module, briefly describe its purpose and role in the solver.
    - Include an example snippet of commented code in the report (e.g., how you describe parameters, return values, and algorithm logic).

Deliverables

Each **group of 3 students** must host their project on a public **GitHub repository**. The repository must contain all required source files, documentation, and the written report.

Repository Structure

Your GitHub repository should include at least the following:

- **Source Code:** All `.c` and `.h` files required to build and run the project.
- **Dictionary File:** A `words.txt` file containing valid 5-letter words (one per line).
- **README File:** A `README.md` written in Markdown, containing:
  - A short description of the project and its objective.
  - Clear instructions on how to download (clone) and run the program.
  - Example usage and sample output.
  - At least one or two screenshots of the program running in the terminal.
- **Report:** A `.pdf` file containing your written analysis, justification, and code documentation (Part 3).

Submission

- Submit the link to your GitHub repository via the course submission platform.
- Ensure that the repository is accessible (public).
- The final commit date on GitHub will be considered the official submission date.

Evaluation Criteria

Grading is split as follows:

- **Correctness (20%)**: Does the game and solver function correctly?
- **Solver Efficiency (20%)**: Average number of guesses needed by the solver (lower is better).
- **Code Quality (20%)**: Readability, documentation, and memory management.
- **Data Structure Usage (20%)**: Appropriate and effective use of data structures.
- **Analysis (20%)**: Depth and clarity of the written justification and complexity analysis.

Optional Extensions (Extra credit)

- Implement alternative solver strategies and compare their performance (report results in the analysis).

Good luck!
Please contact us with any questions or clarifications.