Technical Report

Wordle Game and Solver in C

1. Introduction

This project implements a simplified version of the Wordle game in the C programming language. The application is structured into multiple source and header files in order to ensure modularity, separation of concerns, and code maintainability. The project is divided into two main components: the game logic, which handles user interaction and rule enforcement, and an automated solver, which applies an algorithmic strategy to deduce the correct word.

The program operates using a predefined dictionary loaded from a text file and supports both manual gameplay and automated solving.

2. Objective of the Project

The primary objective of this project is to design and implement a modular C application that:

- Simulates the Wordle game mechanics
- Validates player guesses against a target word
- Provides structured feedback for each guess
- Implements an algorithm capable of solving the game automatically
- Demonstrates proper use of data structures, file handling, and dynamic memory management

3. Problem Being Solved

The problem addressed by this project is the implementation of a constrained word-guessing game where:

- The target word is randomly selected from a dictionary
- The player or solver must guess the word within a limited number of attempts
- Each guess produces positional feedback (correct position, present elsewhere, or absent)
- The solver must exploit previous feedback to reduce the search space efficiently

4. Project Architecture

The project follows a **modular architecture**, separating responsibilities across different compilation units.

Directory Structure

- src/
  Contains implementation files (.c)
- include/
  Contains interface definitions (.h)
- data/
  Contains the word dictionary
- main.c
  Entry point and mode selection

Main Modules

- Game module (wordle_game.c)
- Solver module (wordle_solver.c)
- Program entry (main.c)

5. Source File Descriptions

5.1 main.c

This file serves as the program entry point. Its responsibilities include:

- Displaying a menu to the user

- Allowing selection between manual gameplay and solver mode
- Delegating execution to the appropriate module

The file does not contain any game or solver logic; it strictly controls program flow.

5.2 wordle_game.c

This file implements the core Wordle game mechanics. Its responsibilities include:

- Loading words from the dictionary file
- Randomly selecting a target word
- Handling player guesses
- Computing feedback for each guessed letter
- Enforcing attempt limits

The feedback logic compares each guessed letter against the target word and assigns a color code representing correctness.

5.3 wordle_solver.c

This file implements an automated solver capable of playing the game without user input. Its responsibilities include:

- Maintaining a list of candidate words
- Evaluating feedback constraints
- Scoring possible guesses
- Selecting the optimal next guess
- Iteratively narrowing the search space until the target word is found

The solver is independent of the user interface and interacts with the game logic only through shared data structures and rules.

6. Header File Descriptions

6.1 wordle_game.h

This header exposes the public interface of the game module. It defines:

- Constants related to word length and attempt limits
- Function prototypes for:
  - Loading the dictionary
  - Starting the game
  - Accessing word count information

It hides all internal game state from other modules.

6.2 wordle_solver.h

This header defines the solver interface and its data structures. It includes:

- The LetterFeedback structure, which stores:
  - A character
  - Its associated color feedback
- The prototype of the solve() function

This file ensures that solver logic remains decoupled from the game implementation.

7. Module Interaction

- main.c initializes the program and selects the execution mode
- The game module handles dictionary loading and feedback generation
- The solver module uses the same dictionary and feedback rules to simulate gameplay
- Shared behavior is coordinated through header files without exposing internal state

8. Game Logic

8.1 Data Structures

- Fixed-size character arrays for words
- Arrays of strings to store the dictionary
- Arrays of integers to represent feedback colors

These structures are chosen for simplicity and predictable memory usage.

8.2 Core Algorithm and Flow

1. Load the dictionary from words.txt
2. Randomly select a target word
3. For each attempt:
    a. Read the guess
    b. Compare it with the target word
    c. Generate per-letter feedback
4. Stop when:
    a. The word is guessed correctly, or
    b. The maximum number of attempts is reached

8.3 Important Functions

- Word loading function: reads and stores dictionary words
- Feedback generation function: compares guess and target
- Game loop function: controls attempts and termination conditions

Each function has a single, well-defined responsibility.

9. Solver Logic

9.1 Algorithm Used

The solver uses a **constraint-based elimination strategy**:

1. Initialize the candidate list with all dictionary words
2. Make an initial guess
3. Receive simulated feedback
4. Remove all words that violate feedback constraints
5. Score remaining candidates
6. Select the highest-scoring word
7. Repeat until the solution is found

9.2 Strategy and Decision Process

- Letters confirmed in correct positions are enforced strictly
- Letters marked as present but misplaced must appear elsewhere
- Letters marked absent are excluded entirely
- Word scoring favors guesses that maximize information gain

9.3 Justification of the Approach

This approach was chosen because:

- It systematically reduces the search space
- It is deterministic and reproducible
- It balances performance and implementation complexity
- It closely models human Wordle-solving strategies

10. Algorithmic Complexity

10.1 Time Complexity

- Dictionary loading: **O(N)**
- Feedback evaluation per guess: **O(L)**, where L is word length
- Solver filtering per iteration: **O(N × L)**

Best Case

- Target word guessed early: **O(N)**

Worst Case

- All iterations required: **O(K × N × L)**, where K is number of attempts

10.2 Space Complexity

- Dictionary storage: **O(N × L)**
- Solver candidate lists: **O(N × L)**
- Auxiliary scoring arrays: **O(N)**

Overall space complexity remains linear with respect to dictionary size.

11. Memory Management

11.1 Dynamic Allocations

- Dynamic arrays are allocated for:
    - Word lists
    - Score tracking
    - Candidate storage

11.2 Deallocation Strategy

- Every malloc has a corresponding free
- Temporary buffers are released after use
- Solver arrays are freed at the end of execution

11.3 Memory Leak Prevention

- Ownership of allocated memory is clearly defined
- No dangling pointers are used
- Memory is released before program termination

12. Summary

This project successfully implements a modular Wordle game and an automated solver in C. The design emphasizes separation of concerns, algorithmic clarity, and responsible memory management.