a. Strategy Description

- What is your word selection strategy?

My strategy is simple and logical. I call it the "Filter and Pick First" method.

  - The solver looks at the list of all possible words.
  - It picks the very first word in the list as its guess.
  - It does not guess randomly; it always chooses the next available option.
- How is feedback used to eliminate possibilities?

After making a guess, the game gives feedback (Green, Yellow, or Gray letters). My program uses this information to "filter" the list. It looks at every remaining word and asks: "Does this word fit the clues I just got?" If a word does not fit the clues, it is deleted from the list.

- Why is your approach effective?

It is effective because it is guaranteed to find the answer. Every time the solver guesses, it learns new information. This shrinks the list of possible words smaller and smaller until only the correct answer is left.

b. Data Structure Justification

- Which data structures did you use and why?

I used a 2D Character Array (a simple list of strings).

  - **Reason:** This is the simplest and most efficient way to store a fixed list of words in the C language. Since we know the words are always 5 letters long and there are at most 6000 words, a simple array is perfect.
- What alternatives did you consider?

I considered using a Linked List. However, Linked Lists are more complex to write and can be slower when you need to read the whole list repeatedly.

- How do your choices support your strategy?

Since my strategy is to check every single word in the list one by one, an Array is the fastest choice. It allows the computer to read the data very quickly.

c. Complexity Analysis

- Time Complexity (Speed)

The speed of the program depends on the number of words in the dictionary N.

  - **Filtering:** To filter the words, the program checks every word in the list. This is called **Linear Time** or **O(N)**
  - **Result:** If we double the number of words, the program takes roughly twice as long to run.
- Space Complexity (Memory)

The program uses very little memory. It only needs enough space to store the characters of the words. For 6000 words, this is very small (about 36 Kilobytes).

- **Performance Scenario**
  - **100 words:** The program is instant.
  - **6000 words:** The program is still very fast (fractions of a second) because modern computers handle simple text arrays very easily.

d. Code Documentation

Here is a description of the most important functions in the code:

1. `loadDictionary`
   a. **Purpose:** Opens the text file and loads valid 5-letter words into our list. It prepares the game to start.
2. `filterPossibleWords`
   a. **Purpose:** This is the main logic. It takes the old list of words and creates a new, smaller list containing only the words that match the current clues.
3. `isWordCompatible`
   a. **Purpose:** A helper function. It checks a single word against the clues to see if it is a valid candidate.