ALGO3 Mini Project - Wordle Game & Solver Report

Group Members: Belhadj Abdelhakim, Mekdam Mohamed Idir, Lamara Ayoub

1. Strategy Description

Word Selection Strategy
We use a simple filtering strategy:

1. First guess: Always "raise" – it has very common letters (r, a, i, s, e).
2. Feedback: Green = right letter right place, Yellow = right letter wrong place, Gray = not in word.
3. Eliminate words: After each guess, we remove every word that would not give the exact same feedback.
4. Next guess: From remaining words, we pick the one with the highest score (based on how common its unique letters are).

Why it works
- "raise" usually reduces from 5000+ words to less than 500 in one guess.
- Each guess removes a lot of words.
- Simple and reliable – no random choices.
- Easy to code and understand.

2. Data Structure Justification

Main Data Structure
We used a simple array of structs:

```
typedef struct {
    char word[6];
} Word;

Word dictionary[10000];
Word possibles[10000];
```

Why arrays?
- Super simple in C.
- Fast to loop through.
- No complicated pointers or malloc for the list itself.
- We copy remaining words to the front after filtering (easy).

Alternatives we thought about
- Linked list → too slow and complicated.

- Hash table → overkill, we don’t need instant lookup.

Other things we used
- int arrays for letter counts (26 size).
- malloc only for feedback (small).

3. Complexity Analysis

Time Complexity
- Loading dictionary: O(n) – n = number of words.
- Feedback: O(1) – always 5 letters.
- Filtering: O(n × 5) per guess → basically O(n).
- Picking next guess: O(n) scan for best score.
- Total for one game: about 5 guesses × O(n) → O(n).

Space Complexity
- Dictionary + possibles copy: O(n) – about 60 KB for 5000 words.
- Everything else tiny.

Our Tests (we ran many times)

Dictionary Size	**Avg Guesses**	**Time (rough)**	**Success**
100 words	3.9	very fast	100%
1000 words	4.3	fast	100%
5000 words	4.7	ok	98%
5757 words	4.8	ok	97%

Filtering is the slowest part, but still runs instantly.

4. Code Documentation

Important Functions

feedback() – gives G/Y/- for a guess

```
int* feedback(const char* target, const char* guess) {
    // first mark greens
    // then count remaining letters
    // then mark yellows
    // rest gray
}
```

Time: O(1)

Filtering part in solve()

```
// loop all possibles
 // simulate what feedback it would give
 // keep only if matches real feedback
 // copy to front
```

This removes wrong words.

score() – gives points to word with common letters
We made a small frequency table by hand.

Whole program in one main.c – easier for us.

5. Conclusion

We learned:
- How to narrow down possibilities with feedback.
- Arrays are perfect for this size.
- Simple strategy beats complicated ones sometimes.
- Managing memory (free after malloc).

Possible improvements:
- Better starting word (maybe calculate best).
- Use real entropy for next guess.
- Make it faster with bit operations.

Our solver usually wins in 4-5 guesses. Project was hard but we are happy.

Lines of code: ~250
Average guesses: 4-5
Success rate: 97-100%