Adam Barakat
CPSC-39
Final Project

What the Program Does

The app is a simple program that finds the highest scoring words in a 5x5 letter board game, known as Spellcast. The program uses algorithms to identify valid words based on adjacent letters along with letter scoring and multipliers as well as the use of a swap to maximize scoring potential. It's a useful tool to help players find out what words they're missing, and potentially help them identify patterns in future games more reliably.

Algorithms

Depth First Search (DFS)

The program uses DFS, a recursive algorithm that uses backtracking to exhaustively search an entire tree of nodes. The Big O time of this program in particular is **O(8^n)** where n is word length, and 8 is the possible moves.

Here's how it works in this program specifically:

1. Start at each letter on the board.
2. Recursively explore all adjacent and diagonal cells.
3. Check if the current letter matches the next character in the target word.
4. Mark the cell as visited to prevent reuse in the same path.
5. Backtrack after exploring all directions.

Code snippet:

```java
//performs a depth first search for the words
    private static boolean dfs(char[][] board, String word, int x, int y, int index, boolean[][] visited, boolean swapUsed) {
        swapped = false;//resets the swap each call, if swapping isn't allowed, it's as if it's already used
        if (!allowSwap) {
            swapUsed = true;
        }
        if (index == word.length()) return true;//all matched = done
        if (x < 0 || y < 0 || x >= GRID_SIZE || y >= GRID_SIZE || visited[x][y]) return false;//makes sure you're in the boundries and non-duplicate letters

        if (board[x][y] != word.charAt(index) && swapUsed) return false;//if we cant swap and there is no match, word = fail

        boolean usedSwap = board[x][y] != word.charAt(index);

        visited[x][y] = true;//marks visited letters
        int[] dx = {-1, -1, -1, 0, 1, 1, 1, 0};
        int[] dy = {-1, 0, 1, 1, 1, 0, -1, -1};

        for (int dir = 0; dir < 8; dir++) {//each letter can be branched out in 8 directions, this is how all 8 are explored
            if (dfs(board, word, x + dx[dir], y + dy[dir], index + 1, visited, swapUsed || usedSwap)) {
                swapped = swapUsed || usedSwap;
                return true;
            }
        }

        visited[x][y] = false;
        return false;
    }
```

Scoring Algorithm

The scoring algorithm is a simple algorithm to score each word. The big O time is **O(n)** where n is the word length.

Here's how it works:

1. Iterate through each character of the word.
2. Add the score for each character (using the hash map).
3. If the word includes the double letter or length bonus, it's added/doubled later on

Code snippet:

```java
private static int calculateScore(char[][] board, String word, Map<Character, Integer> letterScores) {//calculate score based on letter value
    int score = 0;
    for (char c : word.toCharArray()) {
        Integer val = letterScores.get(c);
        if (val != null) {
            score += val;
        }
    }
    return score;
}
```

Word Validation

The word validation algorithm iterates through the board using DFS to check each word in the dictionary using adjacency and swaps (if allowed) along with checking special cases (swaps). The Big O time is **O(m * n)** where m is the number of words and n is the board size.

Code snippet:

```java
private static boolean canFormWord(char[][] board, String word, int xcoord, int ycoord) {//checks if a word can be formed using dfs, including doubling
    boolean[][] visited = new boolean[GRID_SIZE][GRID_SIZE];
    for (int i = 0; i < GRID_SIZE; ++i) {
        for (int j = 0; j < GRID_SIZE; ++j) {
            if (dfs(board, word, i, j, index:0, visited, swapUsed:false)) {
                if (i == xcoord && j == ycoord) {
                    doubleScore = true;
                } else {
                    doubleScore = false;
                }
                return true;
            }
        }
    }
    return false;
}
```

Data Structures Used

This program uses the following data structures:

1. 2D Arrays: Represents the board allowing simple access to elements.
2. Hash Map: Stores letter scores for word scoring.
3. Boolean Matrix (board): Tracks visited letters during DFS.
4. Strings: For player communication.
5. Lists: For wordlist management.

Opportunities, Challenges, and Future Additions

I was going to ignore the word length bonus initially as it's usually inconsequential, but I realized that it would make the game more interesting so I added it anyway. As I first thought about this in C++, I loved the idea of DFS for these fast searches through big wordlists, and it looks like it works plenty fast in Java too.

The biggest challenge was definitely the DFS algorithm, I was having difficulty implementing a reliable way to check each direction, but eventually I landed on the 8 position coordinate system I used so it wasn't too big of a deal.

In the future, I think I'll add the 2x letter feature, it's usually inconsequential but it might change things up in the right scenarios. I don't think it'll be too difficult to add, I'll just need to track it like the double letter and account for it during the final scoring. I might also make a GUI for it in the future. If I want to make it more efficient, I could also use some dynamic programming for the DFS algorithm to cache intermediate results, but that might be a little too far as it's already pretty efficient.