Boggle Solver Project Write-Up

Ethan Ferguson
9/12/2025

1. Solution Approach/ Strategy

My boggle solver uses a recursive depth-first search algorithm to explore all valid words on a given NxN boggle board. The program begins by starting a search from each letter on the board. From there, it checks each adjacent letter (including diagonals) and builds words by moving from one letter to the next following Boggle rules.

At each letter, the program forms a new letter sequence and checks if the current path matches any prefix in the given dictionary. If the sequence forms a valid word, it is added to the list of found words. The search then continues from the current position to explore potential extension of the word by adding more letters form longer words. The program ensures that it does not revisit previous letters and does not move backwards according to Boggle rules.

Due to time constraints, I was not able to fully implement the depth first search portion. However, the plan was to use depth first search combined with checking the prefixes against the dictionary, and to get rid of invalid paths to keep the search efficient. The program would recursively explore the board from every starting point, keeping track of visited cells and building words dynamically.

2. Analysis and Reflection

a) How many possible combinations of letters exist on an NxN board?

Each cell on a Boggle board can be filled with one of 26 letters from the English alphabet. Since the board has $N\,x\,N\ =\ N^2$ cells, the total number of possible letter combinations for the board is 26^{N^2}. For example, on a standard 4 x 4 boggle board there are $26^{4x4}\ =\ 26^{16}\ =\ 4.5\,x\,10^{22}$ possible combinations. This is way too many combinations for a human to compute so we have to create a program to do it for us.

b) Explain how solving Boggle is a search problem.

Boggle is a search problem because it requires finding all possible sequences of adjacent letters on the board that form valid words from the dictionary. The goal is to explore every path on the board, checking each sequence against the dictionary

Depth-first search is great for this problem because it explores one path full before moving to the next. DFS also uses backtracking, which allows the algorithm to abandon paths that don't lead to valid words, and allows it to continue from a previous point and move on from there.

c) Strategy for time constraints

To solve a boggle board when there is a time constraint, the boggle has to be fast and know when to abandon certain paths to avoid wasting precious time.

From my analysis, having the program focus on finding shorter words first allows it to be both fast and efficient. By quickly scanning the entire board for shorter words, the solver can build up a lot of points early on. This also leaves time for the solver to go back through the board and attempt to find longer words.

With more advanced strategies and time, the solver could prioritize certain letter groups that frequently appear in valid words. Additionally the solver could adjust its depth of search based on the time remaining searching for longer words if possible.