Description

This project creates a boggle solver game, where users are given an NxN grid with letters in each square. Users are tasked with finding words from this grid where the next letter of each word is in an adjacent box (left, right, up, down, diagonal), but no square can be used twice. Each word the user guesses must also be at least 3 letters. Additionally, this game of boggle has three special tiles: Ie, Qu, and St. These are the only squares with a value of two letters because the letters I, Q, and S cannot be in a square by themselves.

This code includes a Boggle Class which initializes the game by creating a valid grid and dictionary. A grid is valid when it has an equal number of rows and columns (NxN), every square only contains letters and doesn't have any tiles with only I, S, or Q. The dictionary is initialized as a set that is hashbased so it functions as a hashmap that can be searched using prefixes and Depth-First Search. This class also creates a list of solutions based on a given grid and dictionary, using DFS and prefixes to create a set of all possible words.

Code Review

Code Reviewers: Xavier Green, Reihanna Marrett

Xavier complemented the readability of my code as well as my use of recursion in the search function due to its efficiency, but suggested I add more comments in this specific function to better outline how each line contributes. He notes specifically one line where I update the length of the word taking into account the special tiles, where they count for 2 letters instead of 1.

Reihanna also complemented the structure and readability of my code, specifically how easy it is to understand each function and its purpose. After reviewing tho, she found that my code does not efficiently handle larger dictionaries efficiently and suggested I improve this to improve the code's runtime.

Recommendations Summary:

- Add comments to search function to describe the purpose of the functions and steps taken within it
- Review all docstrings to ensure they match up with code functionality
- Update prefix search process to handle larger dictionaries more efficiently

Review Time + Defects Found

- Xavier’s review was very effective because he noted the strength of my code while also pointing out how it could be made even stronger. This helped me see a small change that I would've otherwise overlooked. Xavier brought two defects to my attention and spent around 20 minutes reviewing my code
- Reihanna’s review was also very effective because she pointed out a small, yet important edge case defect which is when the search function is given an extremely large dictionary, it doesn’t search it efficiently. Reihanna pointed out one defect and her review took 20 minutes as well.