**Boggle Game using Trie Data Structure & SDL Graphics**

DSA Project

Wakeel Furqan Ahmed 01-134212-192

# Introduction

The Boggle game is a popular word search game that challenges players to find as many words as possible within a grid of letters. The game is played on a square board with a 4x4 grid of letters, and player has to find as many words as they can by connecting adjacent letters in any direction. In this project, the Trie data structure is used to store and check words in the dictionary, making the word validation process much more efficient.

The project also utilizes the Simple DirectMedia Layer (SDL) library to provide a graphical user interface (GUI) for the game.

# Problem Statement

The traditional approach of validating words in Boggle game is to check each word individually in a dictionary. This approach becomes impractical as the size of the dictionary increases. The task of validating words needs to be made more efficient and faster.

The project aims to achieve the following objectives:

* Design and implement a Trie data structure to store and check words in the dictionary.
* Implement a depth-first search algorithm to check for valid words.
* Use the SDL library to create a GUI for the game.
* Use UML diagrams to model the design of the program.
* Generate random boards and keep track of the score and words found by the player.

By using Trie data structure, the program will be able to validate words in an efficient and faster way. The use of the SDL library will provide a user-friendly graphical interface for the game. The UML diagrams will help in understanding the structure and behavior of the system.

# Objective(s)

The objective of this project is to design and implement a program in C++ that can play the Boggle game using **Trie data structure**. The program should be able to generate a random board, accept input from the user, and check the validity of words against a dictionary using Trie data structure.

# UML Diagram

**Refer to Github/Folder**

# Source Code

**Refer to Github/Folder**

# Data Files

**Refer to Github/Folder**

# Sample Outputs

**Refer to Github**

# Conclusion

**Main Working on how Trie Data Stucture is being used:**

A words dictionary is insert into the trie tree.

For efficient use of Trie Data Structure, The complete string is not checked but it checked letter by letter. In the following way:

Every Time user presses a letter/button, its Node is checked in tree.

(A Board class contains a data member Temp Node),

* Incase, Temp Node is null (1st Letter Pressed), it is assigned to root of Tree.
* Otherwise, the Temp Node is supposed to move down (to its child), the char which is pressed.

1. Incase, the Child Node does not exist (is **Null**). It means that **Such word can never** be found, so **Invalid Word Message** is displayed.
2. Incase, the child Node exists, the Temp Node is updated to it (i.e moves down to it).
3. Incase, the Node(Trie\_Node) **bool** **is\_word** is 1(true) **then hint it given to player (orange buttons**). Now If user pressed any of these Orange button, It is accepted as a word.
4. In case the user un-press a button, then the **parent of previously pressed letter** is **assigned** to Temp Node.

I’ve also used the **same TIRE** tree for **storing registered word(User found words**) instead of another storage for them by using another custom bool is\_registered in the Trie Tree.

The use of Trie data structure made the word validation process more efficient and faster. The program was able to generate Multiple boards, accept input from the user, and check the validity of words against a dictionary using Trie data structure. The SDL library was used to create a user-friendly graphical interface for the game. The UML diagrams were used to model the design of the program. The program also has provision for exception handling and also keeps track of the last button pressed to check if the letters selected by the user are adjacent. The Program, for the ease of user show the Current String and also draws a ordered line behind the pressed Letters. It also keeps track of score and updates Game Player high score accordingly. The program was tested on various input cases and the results were found to be accurate. Words Found list was made Responsive i.e list responses to the words of words

The project met its objectives of designing and implementing a Trie data structure to store and check words in the dictionary, implementing a algorithm mentioned above to check for valid words, using the SDL library to create a GUI for the game, using UML diagrams to model the design of the program, and geneirating random boards and keeping track of the score and words found by the player.