**Title:**

Wordle: Using Trees,

Hashes, Graphs, and Recursions

**Class:**

CIS-17C

**Due Date:**

June 12, 2022

**Author:**

Colby Brunk

**Introduction**

For this project I decided to code my own version of the game Wordle. Wordle is already an online game that is designed to test your knowledge on English vocabulary and your ability to recognize patterns. I chose Wordle as my project because it is a game that I regularly play and have always been interested in the concept behind the game. The game rapidly grew in popularity and received many copycat applications on the App Store and Google Play, which received millions of dollars of advertisement revenue. When this project was assigned, I wanted to test myself to see if I too could create a version of this game. The project took me around 30-40 hours in total. The first four to five hours were spent figuring out how I was going to create the concept of the game with my knowledge of hashes, trees, and graphs. About 10 hours were spent doing research on different ways to utilize these containers, and the last 10-15hours were spent developing the game. My project reached a manageable 810+ lines, with 540 lines belonging to: functions, variables, and container data. The last 300 lines belong to the main function.

**Approach to Development**

When designing the game I had three main focuses, creating a randomly generated list of words, creating a way to store and output user progress, and a working alphabet. All of these ideas were easily implemented with the trees and hashes. I used a tree of four, five, and six letter words to be randomly generated, and I used a Hash for the alphabetical list. To do this I set the key to be the letter in the alphabet and a secondary value to be a true or false value representing whether or not the letter is in the word. The progress functions use the hash’s Boolean data to manipulate the characters in the words. The mainframe of the game relies on these containers, functions, and classes to manipulate and store data.

**Game Rules**

Wordle is an online game where you have 6 attempts to guess a randomly generated word. The objective of the game is to guess the word in as little attempts as possible. When you guess a word, if any of the letters are incorrect it will be removed from the list of available of letters to guess. When you guess a letter correctly, it will be added to the list of correct letters, and when you guess the correct location of a letter in the word, it will appear in the spot the letter is located.

Example:

If the letter you are guessing is BEACH. If you guessed the word BENCH, the letter N would be removed from the list of available letters, and the letters BECH would all be added to the list of correct letters. The word that you are attempting to guess will show BE\_CH. The underscore represents a letter that has not been guessed correctly.

**Description of Code**

500 lines of my code are dedicated solely to classes, functions and variables. The other 300 lines are in the main file, mainly focused on character output explaining your progress in the game. Each function within my code operates exactly as how it is labeled. For instance, my printProgress() function is used to display the progress the user has made throughout the game. I setup my project using all five of the required code: using a hash, a tree, a graph, a recursive sort, and recursive functions. The use of my hash was to store the letters of the alphabet, and assign them a Boolean value. If this value was true it was inside the word, and if it was false, it wasn’t located in the word. My AVL tree was used to store a list of words in a .dat file, and then a random node is generated from that tree for the word the user is trying to guess. My graph was used to store the correct data inputted by the user, and is outputted as the user plays along. I changed some of my important functions from my last project into some recursive functions, such as the upperLower() function which takes the string the user inputs and turns it into all capital letters. Lastly, I used the recursive sort function on an array which saves data from the users input, so it can be outputted in alphabetical order.

**Sample Input/Output**

In this input/output demonstration, you can see how a random word is generated, it is checked with the user’s input, and then outputs a keyboard display with incorrect letters missing (B, C, E, and H). A list of correct letters is displayed below that. Also, since no letters were guessed in the correct spot, the word being guessed shows up as

“\_ \_ \_ \_ \_ “

Text

Description automatically generated

Input

Input

Diagram

Description automatically generated

**Pseudo-Code**

*Initialize*

*While selection doesn’t equal 5*

*Input selection*

*Case 1:*

*Swap letters in alphabet map*

*Swap letters in letters map*

*While attempts is less than 6*

*Generate random word*

*Change word being guessed to capital characters*

*Display letters being used*

*Store values from last guess*

*Display values from last guess*

*User inputs a guess*

*Change users input to capital letters*

*While input size is not equal to 5*

*User inputs a guess*

*If input contains false letter*

*User inputs a guess*

*Changing non-used letters to false*

*Order the keyboard display*

*If guess is correct, go back to Input Selection*

*If attempts = 5 go back to Input Selection*

*Case 2:*

*Swap letters in alphabet map*

*Swap letters in letters map*

*While attempts is less than 6*

*Generate random word*

*Change word being guessed to capital characters*

*Display letters being used*

*Store values from last guess*

*Display values from last guess*

*User inputs a guess*

*Change users input to capital letters*

*While input size is not equal to 5*

*User inputs a guess*

*If input contains false letter*

*User inputs a guess*

*Changing non-used letters to false*

*Order the keyboard display*

*If guess is correct, go back to Input Selection*

*If attempts = 6 go back to Input Selection*

*Case 3:*

*Swap letters in alphabet map*

*Swap letters in letters map*

*While attempts is less than 4*

*Generate random word*

*Change word being guessed to capital characters*

*Display letters being used Store values from last guess*

*Display values from last guess*

*User inputs a guess*

*Change users input to capital letters*

*While input size is not equal to 4*

*User inputs a guess*

*If input contains false letter*

*User inputs a guess*

*Changing non-used letters to false*

*Order the keyboard display*

*If guess is correct, go back to Input Selection*

*If attempts = 4 go back to Input Selection*

*Case 4: Display Tutorial*

*Return to Input Selection*

*Case 5: Exit Game*