The following describes the Lexathon Final Project for CS 3340.501, Computer Architecture.

**Section 1: Program description**

This MIPS assembly program simulates the Android word game Lexathon. The rules for our game are as follows:

* Upon startup, a randomly generated 3x3 grid of letters is printed to the screen.
* The user must manually type in valid English words (in all caps) that can be made from the letters in the grid, and press enter for the word to be validated.
* For each grid, there is exactly one nine-letter word.
* For every correct word, a point is awarded. No points are awarded for duplicate words, incorrect words, or words under 4 letters.

- note: handling of duplicate words does not currently work.

* A timer is set to count down from 60 seconds at the game’s start. For every correct word entered by the user, 5 seconds is added to that time. When that time reaches zero, the game ends upon the next user input.
* When the game ends, the screen displays the user’s score, the correct words that the user entered, and all possible valid words for the game session. The program then asks the user to begin another game.

NOTE: In the original Lexathon game, a rule exists that the user must always include the middle letter in choosing the word. Due to the nature of the dictionary files used with our program, no such rule is in place in our game.

Once the user begins entering words, a timer counts down. Each correct word results in time being added to the timer. The game will end one of three ways: the timer reaches 0 seconds, the user enters “0” to quit, or the user guesses all possible words for the grid.

Once the game ends, a report is printed on the screen detailing the following:

* Number of correct words and their corresponding points
* Number of duplicate words
* Number of incorrect words
* All possible words for the grid

The program will then prompt the user to either start another game or exit the program.

**Section 2: Challenges**

The challenges that our team had were mostly related to the syntax of coding this game in MIPS Assembly language. Logically, the flow of the program was relatively simple to understand, but implementing it in this language required a lot of experimentation and understanding.

An example of a challenge that I personally faced was the process of scanning a file for a nine-letter word. It took me maybe ten minutes to write code for this process in a high-level language, but to translate that to MIPS was much more difficult. The result (as can be seen in the code) was that I used several counters and a loop to count each instance of a character until a newline was reached, and compare a specific counter to a constant each time a newline was encountered. When the counter’s value was equal to 9 (9 characters) then the program would branch to a different area of code to handle that.

**Section 03: Things learned**

I have learned that collaboration is extremely important in completing a project of this scale. Also, this project taught me a lot about how MARS operates, and about how assembly code works. Coding this game taught me about manipulating strings in MIPS, making decisions based on input, and stack manipulation.

**Section 04: Algorithms and techniques**

Here I will explain two algorithms used in the program that I wrote myself: the process of finding a nine-letter word in a file and validating user input.

To find a nine letter word:

1. Find a *random* word in the file by jumping to a random index, scanning the file for a newline character, and stopping at the first index after the newline.
2. Run a loop that counts the number of character preceding a newline, and reports that number to a branch instruction. When that number is 9, the nine letter word has been found. Move the index nine letters back to start reading to the word list at the beginning of the nine letter word.

To validate user input:

1. Receive user input as a string of characters.
2. Compare each character of the user input to the legalWords array. If two characters don’t match, reset the index for the user input and start comparing again at the next word (words in legalWords are separated by newlines).
3. If two characters match AND are newlines, that means that all preceding characters matched, and the word has been found in legalWords. Print that the word is legal and store the word in correctArray (different process).

**Section 05: Team contributions**

|  |  |  |
| --- | --- | --- |
| Lucas | Rohit | Zana |
| File input, game setup, word validation, printing, correctArray, duplicates (attempt) | Video capture,  printing, point counter, team organization | Timer/countdown, correctArray, duplicates (attempt) |