**Lexathon Project Report**

**By:** Amanda Truong

**Team:** Amanda Truong, Patrick Deng, Mai Tran, Sterling Meyers

**My Contribution:**

Amanda: Coded the real-time in-game timer (ends the game when time runs out, adds 20 seconds for every correct word entered) and the score calculator at the end of each game (total words, score, and words per minute)

**Group Members’ Contribution:**

Patrick: Wrote Java code to generate dictionaries from a larger dictionary.txt file. Coded the dictionary loading and reading, random letter grid generation, and the possible solutions array.

Mai: Coded user’s input reading and validation (checking if the word entered is long enough, not already used, a valid English word, included the center letter,...etc)

Sterling: Assisted in designing the algorithms used in the project, debugging and syncing modules of code, and generally keeping the group organized and on-track.

**The Project:**

An emulation of the increasingly popular Android word game Lexathon using MIPS assembly language and the MARS emulator. The goal of this emulation is to use game features such as user input, loading a dictionary of words for future searching, character array checking, runtime interaction with a timer, words per minute calculation, and random word grid generation, and to compile them together to form a fully-functioning game of Lexathon in the console.

**The Implementation:**

The dictionary creation itself was implemented using an internet list of 50,000 + English words in .txt file format, and using proprietary Java program (DictionaryConverter.java) to splice from the dictionary words with fewer than 4 or greater than 9 letters, and output them to dictionary2.txt. It also padded words with spaces so that each word contains 10 characters for searching convenience.

Also a list of only 9 letter words was made from the Java code, output to dictionary9.txt, which is used as a list for all possible letter combinations for the Lexathon grid. A random number generator syscall is used from 0 to the size of the word list, and that result is multiplied by ten to obtain the position of the word, which we read, scramble, and output to the console as a 3x3 grid.

The scrambling was done via separating the word into two halves, and picking a random character from each half to be swapped with the other several times.

A solutions array is generated by reading through the dictionary2.txt until a null character is hit, checking each 10 byte word along the way. Whenever there is a match between the grid and the word, the letter in the grid is replaced with a \*. When the number of \* matches the number of non-space characters in the word, and the center character of the grid (stored at index 4) is also \*, the word is stored in the solutions array, and the grid refreshed to its original state before starting the word search for the next word.

The timer was created using MIPS’s syscall function 30 to get the number of milliseconds since January 1, 1970. Two subroutines are created that use this syscall function. The first subroutine is called once at the very beginning of the game and the other subroutine is called at the end of every word entry from the user. The difference between these two values from both subroutines is the divided by 1000 to get the number of seconds that have elapsed between the two subroutine calls. This number is subtracted from the initial 60-second timer to get the remaining time. For every successful word that is found, an additional 20 seconds is added to the maximum time. The timer is displayed by taking the max time subtracted by the total time elapsed.

**The Strategy:**

We decided to divide the project into four parts: Dictionary loading and random grid generation, the main game where the user inputs and the game checks the input for validation/to carry out a function, the timer, and the score summary. In this fashion, we could exploit parallelism through our independently working on the parts to minimize problems when everything comes together.

**The Challenges:**

1. Finding times to meet up in person, as well as times that we are able to work together during with the Fall Break in between.
   1. We solved this through use of a spreadsheet with all of our available times and minimizing the amount of collaborative work needed except for synthesizing our separate portions. That is, the game start, prompt checks, timer, and summary calculation were all partitioned for maximum independence.
2. We had an issue putting together the game initializing code with the main game code, that is, we had conflicting registers
   1. We solved this issue by having the game initializer store only two global registers (for center character and timer), and compartmentalizing registers into the start game subroutines so subroutines can be
3. We had difficulties getting the game to end as it would constantly loop from the end when told via input to exit.
   1. We got through this issue by debugging as a team, placing print string syscalls at points of interest, finding that we simply misplaced a register address assignment.
4. We ran into trouble with register usage when integrating the timer and words per minute code into the main game code.
   1. To bypassed this issue by resetting and reusing registers for multiple purposes.
5. We got trouble with checking if a word entered is a valid English word.
   1. We realize that the problem was due to difference in the way we format the words storing in solutions array. We resolve it by including another format subroutine to format user’s input into the same format.

**What I Learned:**

This project has been quite a journey. I was fortunate enough to get an amazing group that cooperated and collaborated so well. Although to me programming in MIPS is quite tedious, making Lexathon was surprisingly fun. Our group met up during early November and we outlined everything and set goals. It was great to have checkpoints every week and have open communication with each other. All the parts of the project were split among us so no one would have an overwhelming task to complete. We all helped each other with our individual parts. The challenging part was combining code but after meeting up several times, we were able to put all of our programs together.

In addition to becoming more familiar with Assembly language, learning MIPS forced me to think of what happens behind all the abstraction of a high-level language. For example, things like if/else statements and for loops have to be created using jumps and branches. I think creating the timer for this game was like a fun puzzle. The creation of the timer itself was simple to come up with but the implementation with everyone else’s code was a little difficult to resolve. Also, tiny bugs in MIPS are sometimes hard to detect because you have to keep reading through the code by yourself and following through the registers over and over again if the exceptions that they give you are not helpful enough. I learned how to isolate the problem within my code and debug. Overall, this project allowed me to learn the technicalities of MIPS and how to collaborate with a team.