**Lexathon Project Report**

1. **Program Description:**
   1. Lexathon is a word game where you must find as many words of four or more letters in a given board in the allotted time. Each word must contain the central letter exactly once, and other tiles can be used once. Scores are determined by the number of words found. The player starts the game with 60 seconds, and the game ends when time runs out or the player gives up. The program initially reads words from a dictionary file and stores the characters in a continuous buffer. The buffer is delineated by carriage returns and stored in a dictionary array. Subroutines are called through two main loops in the printMenu() and startGame() methods, printing appropriate prompts and receiving user input for choice and words to check against the board and dictionary. The program prints randomly generated letters in a 3x3 printed matrix, stored as an array with 9 character elements. Input validation with respect to word length and the use of the board’s middle letter are used for determining if a word is in the dictionary array. On program exit, the final score is determined.
2. **Challenges:**
3. **Learned:**
4. **Design (Algorithms/techniques)**
   1. The program is split into various subroutines that operate on argument data and global variables stored in the .data section. Prompts are allocated .asciiz labels, and the board is assigned a .space of 9 bytes, with 1 byte for each character.
   2. *printMenu:* This subroutine controls the flow of the program through choices 1 (Start the Game), 2 (instructions), and 3 (Exit). Globally stored prompts are printed and integer input is read using syscall 5 to determine the user choice and jump to the correct subroutine. printMenu loops, only branching to the exit when user input is equal to ‘3’.
   3. *randomizeBoard:* sets the elements of the gameBoard[] array. The game board is stored in an array ofcharacters, whose content is created by this subroutine. While looping through the 9 elements of the board, randomizeBoard generates a random integer between 65 and 90, corresponding to the ASCII characters A-Z capital letters. Additionally, the subroutine stores a random vowel at the center of the board (position 4), so that a given board has at least one vowel to form words from.
   4. *printBoard:* prints the contents of gameTable[] by iterating from the starting address of the array and adding spacing and new lines where appropriate for presentation.
   5. *loadDictionary:* called initially: reads from the LexathonDictionary.txt file to populate a .space buffer using syscalls 13, 14, and 16. The contents of the buffer are filtered into an array with the carriage return [CR] character as the delimiter such that every element of the array is given 10 bytes of space in the array. Counters iterate through the buffer and array, reading the raw data from the buffer in parallel with assigning values to the array and skipping spaces until each 10byte slot is filled.
   6. *getPlayerAnswerLength:* validates a given input for the length and prints the verified result. If the input is not a valid answer, it loops again so that the dictionary array is not checked until a string of valid length 4 or greater is entered.
   7. *checkDictionary:* checks if a given word is in the dictionary by validating the format and comparing the input with each item in the Dictionary Array. If a word is found, the function returns true (1), otherwise it returns false (0) and prints a message that the word was not found.
   8. *checkMiddle:* determines if the middle letter of a given board is used in the player’s answer by comparing each ASCII value of the input character to the 4th (middle) element of the board. The function returns a true (1) or false (0) in the $v0 register. In *checkDictionary,* if checkMiddle returns a false, the program prints an invalid input message and skips checking the dictionary because of the invalidity of the input.
   9. *startGame:* controls the instruction flow through the game, checking the dictionary, adding the score, and printing the appropriate prompts. The subroutine loops until the program is ended. Initially the function prints the board and receives input using the printBoard() method and the getPlayerAnswerLength() method to get the appropriate information. 4 Options are then presented. Option 1 prompts for a word using only the letters displayed on the board that is at least 3 letters long and does not exceed 9 letters, calling checkDictionary(). If the entered word passes all the condition checks, the user is awarded points into the $s4 register based on the length of the word (5\*length) and asked to enter another word in the next turn. Choose option 2 to display the instructions for the game. Choose option 3 to shuffle the entire board and give the user a board with different letters, calling the randomizeBoard() method. Additionally manages the timer, ending the game if an option is selected and the time is up.
   10. *shuffleboard:* Rearranges the current letters on the board, keeping the middle letter. The algorithm parses the gameTable array and creates the new board by taking random elements from the array in a temporary buffer until the buffer is empty. The middle letter is stored and assigned to maintain continuity of the board.
   11. *getCurTime:* returns the time left since the game is started. When the game starts, *storeInitialTime* is called, using the system time syscall (30) to get the initial time in milliseconds. This value is divided by 1000 to convert to seconds and then stored for comparision. *getCurTime* again uses syscall 30 to find the current time and subtracts it from the initial time to find how long the game has been running. The value is subtracted from 60 to get the time remaining for the game.
   12. *Exit: ends the game, printing the final score.*
5. **Member Contributions**
   1. Project overview understanding of Lexathon
      1. Nischant – responsible for initial research and analysis of project implementation.
   2. Prototyping in C++
      1. Thach – primary C++ coder, responsible for the base structure (including prompts) and algorithms behind the game.
      2. Marco – additional functionality via letterRepeat() and checkMiddle()
      3. Kevin – code optimization for MIPS
      4. Nischant – letterRepeat() condition checking
   3. Subroutines in MIPS
      1. Marco – addScore, shuffleBoard, letterRepeat
      2. Kevin – printMenu, printInstructions, randomizeBoard, printBoard, loadDictionary, getCurTime, storeInitialTime
      3. Thach – startGame, checkDictionary
      4. Nischant – load/checkDictionary prototyping, checkMiddle
   4. Finalization
      1. Kevin– integration of subroutines into Lexathon.asm, debugging, project report design, video recording, dictionary implementation.
      2. Thach - integration of subroutines into Lexathon.asm, debugging
      3. Nischant – User Manual
      4. Marco – shuffleboard integration