Project report

Contents

[a. Description of the program 2](#_gjdgxs)

[b. Challenges 2](#_30j0zll)

[c. What I have learned 2](#_1fob9te)

[d. Discussion of algorithms 2](#_3znysh7)

[Randomly select a word](#_2et92p0)

[Input Validation](#_dbc8q7dzm1tp)

[The Cow and Bull algorithm 2](#_3dy6vkm)

[Music and Sound in the game 3](#_1t3h5sf)

[e. Peer evaluation 3](#_4d34og8)

[f. Suggestions 3](#_2s8eyo1)

# a. Description of the program

The program is basically a Bull and Cow game, which randomly selects a word from a list of eligible words, total of 110 words, each one is a 4 non-repeated letters word and has a legitimate meaning. The program then proceeds on to ask for the user’s input (restrains to 4 non-repeated letters). First scenario, the user’s input passes the validation test, which checks whether any letter in the words is repeated, or each character has to be a valid letter (our program will turn uppercase letter into lowercase letter). After the word has passed all the tests, it will then be separated into single letter in order to get compared with the random word chosen by the program in the beginning. If the letter matches and it’s in the correct position, Bull will get incremented by 1, or if the letter matches but it’s in the wrong position, Cow will get incremented by 1. The result will be recorded and printed out before the user can try another guess. Second scenario will be when the user has typed in an invalid input, the program then notifies the user with the rules of the game and skip the last input. Last scenario is when the user decides to give up on the game, type in “!END” will trigger this option and end the game. Our program ending phrase contains the amount of time it takes for the user to completes (or end) the game, and the result word. We also added a sound into our program, which plays a quick tone right after it takes in the user’s input.

# b. Challenges

# c. What I have learned

# d. Discussion of algorithms

### Randomly select a word

We first created a function to randomly return an integer ranged from 0 to 110 using syscall with $v0 = 42 and $a0 = 110 (max value)

After that, we load a string .asciiz off all continuous 4-letter words without seperator called ‘dict’. we used the generated integer as the initial index of the word and add 4 to it to get the end index of the word. Finally, we run a loop from start index to end index to load each byte from the ‘dict’ to the ‘correct\_word’. The output of this is a ‘correct\_word’ with 4-letter word

### Input Validation

The size of the input is restricted by passing the input syscall a value of 5, so when the user inputs 4 characters, they are automatically submitted along with a null terminator. Then, three loops work to verify that the input string follows the input rules. The first loop checks each byte to see if it’s capitalized, and adds 32 if it is, which, in ASCII, is the offset between the upper and lowercase letters. The word can then be compared against the exit sequence, !end, which is done by loading the guess and exit sequence into memory as words, and comparing them with the beq instruction. Finally, the last 2 loops are nested; the 2nd loop checks whether each byte is a letter, using the knowledge that all letters have been made lowercase, and the 3rd loop, nested inside the 2nd loop, loops through all previous bytes to make sure none of them are the same as the current byte. The result is that each character has been made lowercase, guaranteed to be a letter and checked against all of the other letters for duplicates.

### The Cow and Bull algorithm

By extracting a single letter for both the result and guess word, we then proceed on to comparing them. If they are the same, increment Bull by 1. Otherwise if they are not, keeping track of that letter in guess word, we then iterate through the rest of the result word, compare them 1 by 1, if any matches, increment Cow by 1. After each try, the result index iteration and guess index iteration reset. If Bull = 4, then the game ends.

### Music and Sound in the game

We used a series of syscall with $v0=31 and different values to $a0-$a4 to represent pitch, duration, instrument, and volume of each note respectively .This generates different MIDI sounds in the game. When the game start, we call a ‘welcome\_sound’ procedure to play 16 different notes. When user enter a valid guess, we call the ‘success\_sound’ procedure to play 2 notes.

# e. Peer evaluation

# f. Suggestions