Mounika Narayanan

Dr. Smallberg

CS 31 Section 1

November 17, 2015

Project 5 Report

1. While doing this project, it was tricky getting the words to load and to check if the word the user entered was in the word list array. I was thinking about the solution from the negative perspective, but once I changed my point of view, it made coding that segment easier. Also, it was tricky ensuring that pebbles does not recount rocks or the same letter. I had to play around with the code and understand how to manipulate the words to get the correct output.
2. Pseudocode

**bool isValidLength**

checks if probeWord is 4-6 characters long by iterating through with for loop

**bool isLower**

checks if probeWord has lowercase letters by iterating through with for loop

**bool isLetter**

checks if probeWord is all alpha characters with for loop

**bool knowsWord**

checks if array words[] knows the probeWord the user entered by iterating through array for nWords and comparing each row to the probeWord

**int manageOneRound**

* if there are no words loaded, the random word is at a negative position or at a position larger than what the array is holding, returns -1
* reads in probeWord from user after prompting
* via while loop, keeps asking user for probe word until it matches the random word
* checks if probeWord is of valid length, has only lowercase letters otherwise prints appropriate output
* checks if probeWord is known in array otherwise prints appropriate output
* otherwise, declares copy of the random word
* checks for rocks via for loop; if character matches at same location in both arrays, adds 1 to rocks and changes characters to avoid recounting
* checks for pebbles via 2 for loops (1 is nested); if any character in probeWord matches the one character looped for in secretWord, adds 1 to pebbles and changes characters to avoid recounting
* outputs rocks and pebbles
* adds 1 to how many tries it took to get the word
* reads in another probe word
* returns # of tries it took

**int main()**

* declares wordlist, maxWords, and nWords
* loads words via loadWord
* checks if no words were loaded and prints out appropriate response
* otherwise plays game
* records # of rounds the user wants to play after prompting
* if # entered is negative, prints out appropriate response and ends program
* executes # of rounds via for loop
* generates random number and retrieves random word in array
* does one round and records # of tries it took
* outputs # of tries it took (if 1 = “try”) (if >1 = “tries”)
* establishes min and max tries after respective round via if statements
* adds # of tries of that round to total sum of tries
* computes average # of tries for rounds thus far
* outputs average, minimum, maximum for that round so far