Douglas Frattini Edwards

Professor David Smallberg

COM SCI 31

14 November 2022

**Part A: Notable Obstacles**

The primary significant obstacle faced when working on this project was surprisingly the function to count the number of silvers given a probe and secret word. My original implementation of the function was messy and designed without a solid grasp of the various reasons why a found pair might not be a valid silver (ie. already gold/paired with something else). This resulted in long, ugly code that had multiple flaws and was continuously being built upon basically through trial and error. At a certain point, I decided to take a break from the main program and rewrite the function using a detailed and comprehensive written plan for how it would work. Once complete and polished, the function no longer posed significant issues and the rest of the project was completed shortly thereafter.

**Part B: Design of the Program**

* Main: Fairly simple implementation. Load in the array of words, check for errors, set up variables to track the gameplay such as score, silver/gold counts, total score, rounds played, collect the number of rounds to be played, and then run the main gameplay loop that many times. The main loop is fairly simple. It starts by resetting the score to 0, displaying the current round, randomly generating the word number and then sending it to playOneRound, storing the resulting score and adding it to the total to later calculate the average. Then it’s as simple as displaying the number of tries to win and the average/min/max stats across all rounds
* playOneRound: First check for any errors, then load in the secret word. I then set up all the variables needed and start the main guessing loop. This loop retrieves the probe word and uses the functions I made to check it for errors and the number of silvers and golds. If numberOfGolds == lengthOfSecret, I return the final score. Otherwise the number of silvers and golds is displayed as written in the spec and the next loop begins
* goldScore is a fairly simple function. It first determines which of probe/target is the shorter string and uses its length as the indexing limit to avoid out of range indexing. It then checks if any probe characters have a match with target characters at the same index.
* silverScore was by far the most challenging aspect of the program, but its final form ended up being quite short and simple (first implementation was ~40 lines, now down to 19!). Besides the basic idea of going through each probe character and checking for a match with each target character, I identified that silvers shouldn’t be counted either if the probe/target character in question could make a gold pair elsewhere, or if the target character had already been paired as a silver with something else. To check for golds elsewhere was quite simple, the program just needs to run a check to ensure it doesn’t perform out of range indexing before checking the probe characters location in target and vice versa. To identify already paired characters, I created an array that stores the location of every paired character in the target, which my script then searches through to determine if a silver is valid to make. If the target character is already paired, the match is ignored and “continue;” is used to skip that character. If it is unpaired, the silver is added to the return count and the location of the paired target character is added to the paired array
* validWord: this function is fairly simple. It first checks to ensure that the word is within the bounds of max and min word length, and then ensures that every character is a lower case letter
* strLength is my own implementation of finding the number of valid characters in a c string, although I’m sure there’s a function I forgot about and was too lazy to search for. It simply iterates through the string until finding a ‘\0’, at which point it returns i