a.

The most notable obstacle I encountered while developing my program was how to find the number of rocks and pebbles. I developed my program by first writing my main function. I wrote all the code for reading input and displaying output appropriately, while handing off the actual managing of a round to the manageOneRound function. Once I completed developing the main function, I wrote the manageOneRound function, which called another function I made, manageOneTurn, for as many times as needed to complete a round. The manageOneTurn function was where I encountered obstacles. I figured out how to find the number of rocks between a probe word and the secret word fairly easily. The tricky part was finding the number of pebbles. I first just looped through every character pair in the words and looked for matches, but that counted many repeats and did not work. I then realized that I needed to create a copy of the probe word and secret word and then change each character that was a match in the copies to something else to prevent recounting. I also had to change my method of finding rocks so any rocks that I had found would not be recounted as pebbles again.

b.

My main routine creates an array wordList of size MAXWORDS and loads it with words using the loadWords function, storing the number of words in nWords. It then declares variables rounds, min, max, average, totalScore to store statistics on the number of tries taken and the number of rounds in the game. It will read in input from the user for how many rounds to play and terminate if rounds is not positive. Using a for loop for as many times as the number of rounds to play, it chooses a random word and calls manageOneRound to play one round of the game. It will take the score from the round and produce and display statistics and messages appropriately.

main()

declares wordList

loads wordList with nWords

if nWords < 1, terminate program

initialize rounds, average, total score to 0, min to largest integer, max to smallest integer

read in rounds from user input

for number of rounds

display round number

display length of random word from wordList

record return value of manageOneRound as score

if score is -1, terminate program

update min and max scores

add score to total score

calculate average score

display how many tries it took for player to guess secret word

display average, min, max scores

My manageOneRound takes in the parameters the array of words, nWords, and wordnum. It checks if nWords is not positive, if wordnum is less than 0, and if wordnum is greater than or equal to nWords. If any of those are true, it returns -1 due to incorrect input. It then declares a boolean variable match to store whether a match is found between the secret word and the probe word. It initializes a score variable to store the player’s score and creates a while loop with match as the condition. While a match is not found, it calls manageOneTurn and depending on the return value of the function, manageOneRound will return error messages or increment the score each time a probe word is compared to the secret word. If a match is found, the score is incremented and the program exits from the while loop and returns the score.

manageOneRound(array of words, nWords, wordnum)

if nWords is not positive or wordnum is less than 0 or greater than or equal to nWords

return -1

declare boolean to track whether player guessed secret word

declare score variable to track score

while a match is not found

check return value of manageOneTurn

if it is 1, display message "Your probe word must be a word of 4 to 6 lower case letters.”

if it is 2, display message "I don't know that word.”

if it is 3, increment score, a match is found

if it is 4, increment score

return score

I made a manageOneTurn function to manage a single turn of gameplay within a round. It takes in the parameters the array of words, nWords, and wordnum. It creates the arrays probe, checkProbe, and secretWord to store the probe word, a copy of the probe word, and a copy of the secret word. It then declares a boolean variable found to track whether a probe word is found in the array of words. Variables are also made to store the number of rocks and pebbles found between a secret word and probe word. The function asks the user to input a probe word and stores it in probe, while also making a copy of the probe word in checkProbe and a copy of the secret word in secretWord. If the length of the inputted probe word is less than 4 or greater than 6, the function returns 1. It then loops through the probe word and checks if each of the characters in the probe word is a letter and is lower case. If not, the function returns 1. It then checks if the probe word can be found in the array of words. If not, the function returns 2. Then, it checks if the probe word matches the secret word. If yes, the function returns 3. The function then loops through both checkProbe and secretWord until one ends and checks if a letter appears at the same position in both arrays. If one is found, rocks is incremented and the letter is changed to ‘A’ in one array and ‘Z’ in the other to prevent recounting of the same letter. The function then uses a nested for loop to check each character in checkProbe against each character in secretWord. If the same letter is found but in different positions, pebbles is incremented and the letter is changed to ‘A’ in one array and ‘Z’ in the other to prevent recounting of the same letter. At the end, the function writes out the number of rocks and pebbles and returns 4 to signal successful comparison of the probe word and secret word to manageOneRound.

manageOneTurn(array of words, nWords, wordnum)

creates arrays probe, checkProbe, secretWord

declare boolean to track if probe word is in word array

declare variables rocks and pebbles

take input from user for probe word

copy probe to checkProbe, copy secret word to secretWord

if length of probe is less than 4 or greater than 6, return 1

if any character in probe is not a lower case letter, return 1

if probe word is not in array of words, return 2

if probe word matches secret word, return 3

for each char in shorter of secretWord and checkProbe

if a letter appears in the same position

increment rocks

change letter in secretWord to ‘A’

change letter in checkProbe to ‘Z’

for each char in secretWord

for each char in checkProbe

if a letter is the same and is not in the same position

increment pebbles

change letter in secretWord to ‘A’

change letter in checkProbe to ‘Z’

break

display number of rocks and pebbles

return 4