Kevin Meehan

October 18, 2023

CSE 491

Professor Ofria

# Wordle (By ChatGPT)

As of November 29th, 2022, AI for software engineers was hard to come by that could immensely help them write code. However, with the launch of ChatGPT, this was about to change. Software engineers could now interact with this AI and ask it almost any question about software, ranging from introductory knowledge of a new language, to debugging hundreds of lines of code and getting an answer within seconds. Although ChatGPT is extremely smart, it is only as smart as the person asking it questions is. Today, I will be using ChatGPT to write a medium sized program and analyzing what it does good, what it does bad, and what can be improved on.

Wordle is a popular game that people can play where there is a five letter word and the goal is to guess the word in six guesses. If a letter is guessed correctly, it will be in green; if the letter guessed is in the word, but it is in the wrong location, it will be yellow; and if the letter is guessed incorrectly, it will be in gray. Because writing a software program that runs in c++ will be difficult to exactly replicate the original Wordle game, I had to come up with a slightly different way the player could interact with the program and play the game. This was my initial

statement to ChatGPT.

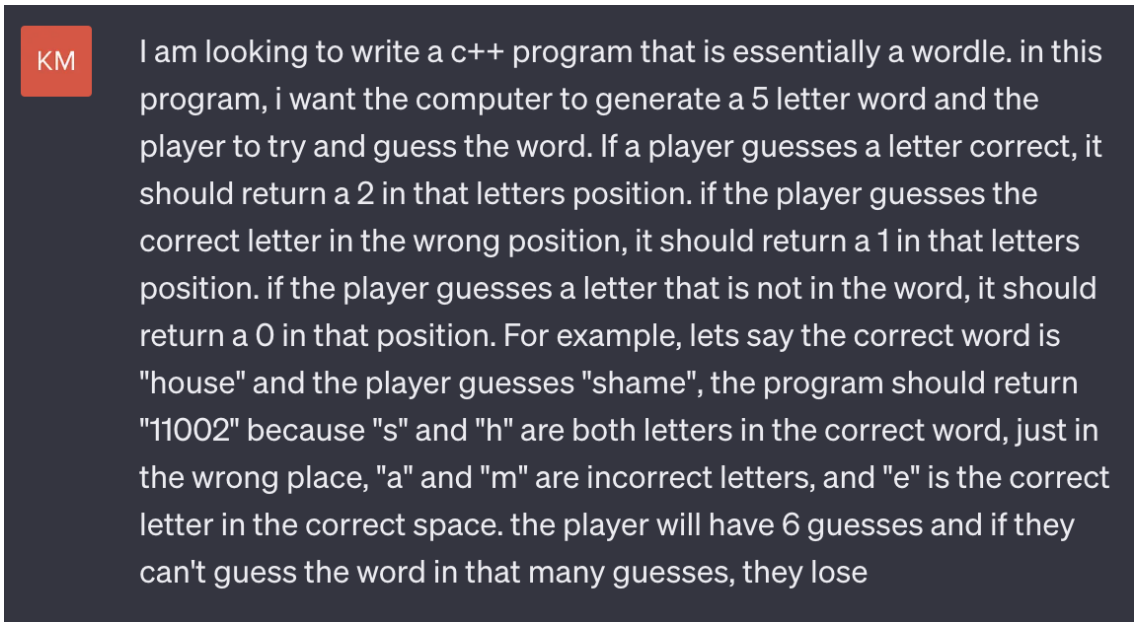


Figure 1

With this statement, I got a partially functioning Wordle game with some errors, but ChatGPT has the right idea about the whole game. The first error was in the response of the program to the player guessing the word. For example, I tried this initial project out by guessing the word “swamp” as my first word and the correct word was “world”. The program returned “10000” and it should have returned “01000” because the letter ‘w’ was in the word, but in the wrong location. ChatGPT responded with this code to compare the word guessed to the target word.

```cpp
// Function to compare the guessed word with the target word
std::string CompareWords(const std::string& target, const std::string& guess) {
    if (target.size() != guess.size()) {
        return "Invalid guess length";
    }

    std::string result;
    std::vector<bool> matched( n: target.size(),  x: false);

    // Check for exact matches (correct letter in the correct position)
    for (int i = 0; i < target.size(); i++) {
        if (target[i] == guess[i]) {
            result += '2';
            matched[i] = true;
        }
    }

    // Check for partial matches (correct letter in the wrong position)
    for (int i = 0; i < target.size(); i++) {
        if (result.size() == target.size()) {
            break;  // All positions are filled
        }
        if (!matched[i]) {
            for (int j = 0; j < target.size(); j++) {
                if (!matched[j] && target[j] == guess[i]) {
                    result += '1';
                    matched[j] = true;
                    break;
```

Figure 2 (Chat GPT's first version of CompareWords() function)

```cpp
// Function to compare the guessed word with the target word
std::string CompareWords(const std::string& target, const std::string& guess) {
    if (target.size() != guess.size()) {
        return "Invalid guess length";
    }

    std::string result( n: target.size(),  c: '0');
    std::vector<bool> matched( n: target.size(),  x: false);

    // Check for exact matches (correct letter in the correct position)
    for (int i = 0; i < target.size(); i++) {
        if (target[i] == guess[i]) {
            result[i] = '2';
            matched[i] = true;
        }
    }

    // Check for partial matches (correct letter in the wrong position)
    for (int i = 0; i < target.size(); i++) {
        if (!matched[i]) {
            for (int j = 0; j < target.size(); j++) {
                if (!matched[j] && target[j] == guess[i]) {
                    result[i] = '1';
                    matched[j] = true;
                    break;
                }
            }
        }
```

Figure 3 (ChatGPT's correct and final version of CompareWords() function)

These two functions are very similar, but do a couple things differently when assembling the response string. This function gets called after the user types in a word. The 'target' string is the string of the hidden word the player is trying to guess, and the 'guess' string is the word the player inputted as its guess. The first function does not take into consideration the location of the given number in the string 'result' like the final function. It strictly finds a letter that is either correct, incorrect, or partially correct and adds it to the end of the string result. In the final function, however, it specifies the index of where each number should be. It does this by making a string filled with zeros and then updating each number as it goes on. The first function, however, just creates an empty string and adds numbers onto it as it goes. As we can see in line 48 in Figure 3 compared to line 121 in Figure 2, both functions check for matched letters first. In the first function, if it finds a matched letter, it adds a '2' to the end of the string. In the second function, however, it modifies the string of zeros to change the correct position of the correct letter to a '2'.

The next problem I had was that the code provided only gave a handful of words to use as the hidden word. I wanted to have more words to be used so the player wouldn't be getting the same words every time. This brought up a problem, however, because the c++ standard library does not have a way to import English words, let alone five letter English words. This was not a problem for ChatGPT though, because it mentioned a couple ways of solving this issue. The way I chose was to use a text file and fill up the text file with as many five letter words I wanted. ChatGPT recommended a few different sites, but I chose this website "https://7esl.com/5-letter-words/#5_Letter_Words_with_A". From here, I was able to copy and

paste all the five letter words into a text file and have ChatGPT write a function to pull a random word from the text file and use that as the hidden word.

```
110  std::vector<std::string> LoadWordList(const std::string& filename) {
111      std::vector<std::string> wordList;
112      std::ifstream file( s: filename);
113      if (file.is_open()) {
114          std::string word;
115          while (std::getline( &: file,  &: word)) {
116              if (word.size() == 5) { // Filter 5-letter words
117                  wordList.push_back(word);
118              }
119          }
120          file.close();
121      } else {
122          std::cerr << "Failed to open word list file." << std::endl;
123      }
124      return wordList;
125  }
```

Figure 4

```
31  std::string GenerateRandomWord(const std::vector<std::string>& wordList) {
32      int randomIndex = rand() % wordList.size();
33      return wordList[randomIndex];
34  }
```

Figure 5

These two functions in Figure 4 and 5 were produced by ChatGPT after I filled up the text file with five letter words and it worked perfectly. What happens in the main function is it calls LoadWordList() with the file path to the text file I created and loads each individual word into a vector of strings. It then calls the GenerateRandomWord() function and returns a randomized word inside the list it just created.

This code works perfectly, but there are a few things that I think it could improve on. The first being the ‘if’ statement in line 116 of Figure 4. Every word in the text file is already five

letters, so someone might say that it is redundant, but one argument could be made that it is always good to double check as it could cause the program to crash if there was a typo of some sort. Another thing is that each time the program is run, it has to loop through every single word in the text file (561 to be exact). Because this number is relatively small, performance issues are irrelevant. However, if I was going about solving this problem, I would have made a database that stored all the words and then generated a random number inside the GenerateWord() function and directly pulled a word from the database. This way, I wouldn't have to worry about recreating a list of all the words every time the program is run.

```
80      std::string target = GenerateRandomWord(wordList);
81      int maxAttempts = 6;
82      int attempts = 0;
83
84      std::cout << "Welcome to Wordle! Try to guess the 5-letter word." << std::endl;
85
86      while (attempts < maxAttempts) {
87          std::string guess;
88          std::cout << "Attempt " << (attempts + 1) << ": ";
89          std::cin >> guess;
90
91          std::string result = CompareWords(target, guess);
92          std::cout << result << std::endl;
93
94          if (result == "22222") {
95              std::cout << "Congratulations! You guessed the word: " << target << std::endl;
96              break;
97          }
98
99          attempts++;
100     }
101
102     if (attempts == maxAttempts) {
103         std::cout << "Sorry, you've run out of attempts. The word was: " << target << std::endl;
104     }
105
106     return 0;
107 }
```

Figure 6 (Main Function Created by ChatGPT)

The main function for this program isn't too complicated because no comparing or pulling data needs to be done. What ChatGPT does nicely is for the whole program, it makes

helper functions for the information processing of the program. Whether it be comparing strings, loading in data, or simply choosing a random word, this code is put into helper functions. The code in main actually worked perfectly the very first time as well. The issues I had were in the helper functions. The main code had to be updated a couple times for when the helper functions got changed, but the structure and functionality of the code inside main worked perfectly. The logic for the main is extremely straightforward and I cannot think of a way to improve ChatGPT's code.

```
// Templated function to load a word list from a file
template <typename T>
std::vector<T> LoadWordList(const std::string& filename, size_t wordLength) {
    std::vector<T> wordList;
    std::ifstream file( s: filename);
```

Figure 7 (Templated LoadWordList function)

```
// Templated function to generate a random word from a word list
template <typename T>
T GenerateRandomWord(const std::vector<T>& wordList) {
    if (wordList.empty()) {
```

Figure 8 (Templated GenerateRandomWord function)

```
// Templated function to compare two words
template <typename T>
std::string CompareWords(const T& target, const T& guess) {
    if (target.size() != guess.size()) {
```

Figure 9 (Templated CompareWords function)

One way of improving this project would be by using templated functions. This would improve everything if someone were to want to change the game to a number guessing game, or

anything not involving words. In order to do this, I asked ChatGPT to modify the entire project and use templated functions instead. ChatGPT came up with a new project, and correctly modified all the helper functions. I plugged everything into CLion and the project worked just as it did before. In order to test to see if it really worked, I deleted the text file with the five letter words and plugged in a bunch of random five number integers. After doing this, I also had to make a couple more small changes, such as deleting the transform function that turns every letter to lowercase, but besides that, the new number guessing game worked as well. Although this new game with numbers does not have much skill involved like a Wordle game does (it consists of randomly guessing numbers and using process of elimination for numbers guessed in the wrong place), it demonstrates that ChatGPT's new templated functions work correctly. As seen in figures 7-9, ChatGPT didn't have to change very much to the project to change the functions to templated functions. The main thing it had to do was add "template <typename T>" above the function declaration, as well as change the variable type to a 'T' instead of 'string'.

By using ChatGPT, I was able to make this program work correctly in about 30 minutes. If I were to write the code myself, it would probably take at least three hours, and would definitely not be as efficient compared to ChatGPT's code. However, unless there was a user talking with ChatGPT while writing this program, this program would not work correctly. ChatGPT does not have a mind of its own and is only as smart as the user. This brings a great balance between human and AI interaction and the way they have to interact in order to achieve a common goal.