Case Study: Wordle Game Implementation in C

DESCRIPTION: Wordle is a popular word puzzle game where a player attempts to guess a hidden word within a limited number of attempts. The game provides feedback on each guess by indicating correct letters in the correct positions and correct letters in the wrong positions. The objective is to deduce the hidden word through intelligent guessing.

AIM: The objective of this project is to implement a command-line version of the Wordle game using the C programming language. The implementation would allow players to interactively play the game by entering their guesses and receiving feedback until they either correctly guess the hidden word or exhaust their allowed attempts.

SCOPE:

The scope of this project involves creating a functional Wordle game that meets the following requirements:

1. ***User Interface:***
   - The game is implemented as a command-line application.
   - The player will be able to input their guesses using the keyboard.
2. ***Game Rules:***
   - The hidden word, to be guessed by the player, will be predetermined by the program.
   - The hidden word will consist of a fixed number of characters (e.g., 5, 6, 7 or 8 characters).
   - The player will have a limited number of attempts to guess the hidden word (e.g., 6 attempts).
3. ***Feedback Mechanism:***
   - After each guess, the player will receive feedback indicating the number of correct letters in the correct positions and the number of correct letters in the wrong positions.
   - The feedback will be displayed to the player.
4. ***Game Flow:***
   - The player's guess and feedback will be displayed after each attempt.
   - The player is informed whether their guess is correct or if they have exhausted their attempts.
   - If the player guesses the hidden word correctly, the game will terminate with a congratulatory message.
5. ***Error Handling:***

The program can handle invalid input gracefully, providing appropriate messages to the player.

SCREENSHOTS

COMPILATION:

```
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$ make
clang -ggdb3 -O0 -Qunused-arguments -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -Wno-unused-parameter -Wno
-unused-variable -Wshadow -lm -o wordle wordle.c helper.c cs50.c
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$
```

ERROR HANDELING:

```
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$ ./wordle
Usage: ./wordle wordsize
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$ ./wordle 10
Error: wordsize must be either 5, 6, 7, or 8
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$
```

GAME PLAY:

```
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$ ./wordle 5
This is WORDLE50
You have 6 tries to guess the 5-letter word I'm thinking of
Input a 5-letter word: clear
Guess 1: c l e a r
Input a 5-letter word: empty
Guess 2: e m p t y
Input a 5-letter word: shake
Guess 3: s h a k e
Input a 5-letter word: demon
Guess 4: d e m o n
Input a 5-letter word: women
Guess 5: w o m e n
You Won!
suryansh@Suryansh:/mnt/c/Users/ASUS/Desktop/WORDLE$
```

wordle.c

```c
#include <cs50.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include"helper.h"

// each of our text files contains 1000 words
#define LISTSIZE 1000

// values for colors and score (EXACT == right letter, right place; CLOSE == right letter, wrong place; WRONG == wrong letter)
#define EXACT 2
#define CLOSE 1
#define WRONG 0

// ANSI color codes for boxed in letters
#define GREEN   "\e[38;2;255;255;255;1m\e[48;2;106;170;100;1m"
#define YELLOW  "\e[38;2;255;255;255;1m\e[48;2;201;180;88;1m"
#define RED     "\e[38;2;255;255;255;1m\e[48;2;220;20;60;1m"
#define RESET   "\e[0;39m"



int main(int argc, string argv[])
{
    // ensure proper usage
    // TODO #1
    if(argc != 2){
        printf("Usage: ./wordle wordsize\n");
        return 1;
    }

    int wordsize = 0;

    // ensure argv[1] is either 5, 6, 7, or 8 and store that value in wordsize instead
    // TODO #2
    for(int i = 5; i <= 8; i++){
        if(atoi(argv[1]) == i){
            wordsize = i;
        }
    }

    if(wordsize == 0){
        printf("Error: wordsize must be either 5, 6, 7, or 8\n");
        return 1;
    }

    // open correct file, each file has exactly LISTSIZE words
    char wl_filename[6];
    sprintf(wl_filename, "%i.txt", wordsize);
    FILE *wordlist = fopen(wl_filename, "r");
    if (wordlist == NULL)
    {
        printf("Error opening file %s.\n", wl_filename);
        return 1;
```

```
    }

    // load word file into an array of size LISTSIZE
    char options[LISTSIZE][wordsize + 1];

    for (int i = 0; i < LISTSIZE; i++)
    {
        fscanf(wordlist, "%s", options[i]);
    }

    // pseudorandomly select a word for this game
    srand(time(NULL));
    string choice = options[rand() % LISTSIZE];

    // allow one more guess than the length of the word
    int guesses = wordsize + 1;
    bool won = false;

    // print greeting, using ANSI color codes to demonstrate
    printf(GREEN"This is WORDLE50"RESET"\n");
    printf("You have %i tries to guess the %i-letter word I'm thinking of\n", guesses, wordsize);

    // main game loop, one iteration for each guess
    for (int i = 0; i < guesses; i++)
    {
        // obtain user's guess
        string guess = get_guess(wordsize);

        // array to hold guess status, initially set to zero
        int status[wordsize];

        // set all elements of status array initially to 0, aka WRONG
        // TODO #4

        for(int j = 0; j < wordsize; j++){
            status[j] = WRONG;
        }

        // Calculate score for the guess
        int score = check_word(guess, wordsize, status, choice);

        printf("Guess %i: ", i + 1);

        // Print the guess
        print_word(guess, wordsize, status);

        // if they guessed it exactly right, set terminate loop
        if (score == EXACT * wordsize)
        {
            won = true;
            break;
        }
    }

    // Print the game's result
    // TODO #7
    if(won == true){
        printf("You Won!\n");
    }else{
```

```c
        printf("You Lose!\n");
        printf("The correct word was %s\n", choice);
    }

    // that's all folks!
    return 0;
}

```

helper.h

```c
#include<string.h>
#include<cs50.h>

// user-defined function prototypes
string get_guess(int wordsize);
int check_word(string guess, int wordsize, int status[], string choice);
void print_word(string guess, int wordsize, int status[]);
```

helper.c

```c
#include<string.h>
#include<stdio.h>
#include <stdlib.h>
#include"cs50.h"
#include"helper.h"

// each of our text files contains 1000 words
#define LISTSIZE 1000

// values for colors and score (EXACT == right letter, right place; CLOSE == right letter, wrong place; WRONG == wrong letter)
#define EXACT 2
#define CLOSE 1
#define WRONG 0

// ANSI color codes for boxed in letters
#define GREEN   "\e[38;2;255;255;255;1m\e[48;2;106;170;100;1m"
#define YELLOW  "\e[38;2;255;255;255;1m\e[48;2;201;180;88;1m"
#define RED     "\e[38;2;255;255;255;1m\e[48;2;220;20;60;1m"
#define RESET   "\e[0;39m"

// Promts the user to enter the guessed word
// Also error checks if the length of the word is of right size, otherwise re-promts the user to enter
string get_guess(int wordsize)
{
    string guess = "";

    // ensure users actually provide a guess that is the correct length
    // TODO #3

    do{
        guess = get_string("Input a 5-letter word: ");
    }while(strlen(guess) != 5);

    return guess;
}


// Compares the guessed word at each stage and adds a score value,
// Correct letter, rigth position = 2 (EXACT), Correct letter, wrong position = 1 (CLOSE), Wrong letter = 0 (WRONG)
int check_word(string guess, int wordsize, int status[], string choice)
{
    int score = 0;

    // compare guess to choice and score points as appropriate, storing points in status
    // TODO #5

    // iterate over each letter of the guess
    for(int i = 0; i < wordsize; i++){

        // iterate over each letter of the choice
        for(int j = 0; j < wordsize; j++){
            // compare the current guess letter to the current choice letter
            if(guess[i] == choice[j]){
```

```
                // if they're the same position in the word, score EXACT points (green) and break so you don't compare that letter further
                if(i == j){
                    score += EXACT;
                    status[i] = EXACT;
                    break;
                }else{
                    score += CLOSE;
                    status[i] = CLOSE;
                }
                // if it's in the word, but not the right spot, score CLOSE point (yellow)
            }
            // keep track of the total score by adding each individual letter's score from above
        }
    }
    return score;
}


// print_word - Prints the feedback to the user using different color codes (GREEN, YELLOW, RED)
// - Highlights the letters with GREEN for correct letter correct position,
// - With RED for wrong letters,
// - With YELLOW for correct letter wrong position.

void print_word(string guess, int wordsize, int status[])
{
    // print word character-for-character with correct color coding, then reset terminal font to normal
    // TODO #6

    for(int i = 0; i < wordsize; i++){

        if(status[i] == 2){
            printf(GREEN " %c " RESET, guess[i]);
        }else if (status[i] == 1){
            printf(YELLOW " %c " RESET, guess[i]);
        }else{
            printf(RED " %c " RESET, guess[i]);
        }
    }

    printf("\n");
    return;
}

```