CODE DOCUMENTATION

Programming Project – SS24

JUNE 20, 2024

MOEEZ MUFTI
moeez.mufti@stud.hshl.de

Contents

Abstract

Hangman: Untangling Themes & Quests

This project proposes a hangman game which follows the traditional hangman rules but adds some new twists.

- There are a total of six tries. It has a thematic twist where there will be two levels: First & Second.
- First Level will be decoding the theme of arts which is either literature or movies based on the hint given.
- Second Level will be finding the specific word related to it like an author's work or a director’s popular film. The hints for this will also be given.
- In the second level, for every streak of three, there will be extra life given by deducting one number from the number of misses.

Data Types Used

Char

Char is a data type in C used to store characters and strings. For the latter it is must to use the char type and create an array of characters to make a string. It is of 1 byte. It can range from -128 to 127 or 0 to 255.

Int

Int is a data type in C. It is of 2 or 4 bytes. It can range from -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 (unsigned int)

Pointers

In the code, I have used a single asterisk * and double asterisk ** for few variables. This is because, for those variables, there existed multiple options.

For example, `char *word` is used to store either of two options by storing the memory addresses based on the random choice created two separate pointers for each word. This way calling the specific values won't require additional declaration or initialization of each option. The pointer stores the memory address of the first letter of each word.

For double asterisks, `char **options` was used. This is a pointer that points to another pointer. In this case, it pointed to either `char *literature or moviesOptions` which both are pointers since multiple options were used so I needed a pointer variable that could store memory addresses of either of the two arrays and create multiple pointers for that string of arrays. So, the options pointed to them which stored its memory address which then can be used to manipulate based on the functions.

Libraries Used

#include <stdio.h>

It's used for standard input/output operations like `printf` and `scanf`

#include <stdlib.h>

This library provides general utilities, including memory allocation, random number generation, and program termination. In this program, `srand` and `rand` functions are used for generating random numbers.

#include <string.h>

This library provides string manipulation functions like `strcpy` and `strcmp`. In this program, `strcpy` is used to copy strings, and `strcmp` is used to compare strings

#include <ctype.h>

This library provides functions for character handling. In this program, `tolower` function is used to convert characters to lowercase.

#include <time.h>

In this program, `time` function is used to get the current time, which is then used as a seed for the random number generator.

#include "hangman_functions.h" :

This is a custom header file made to list all the functions made and used for this game.

Functions:

Int main(){…}

The int main() function serves as the entry point of the program. When you execute a C program, the operating system invokes the main function to start the execution of the program. The body of the main function contains the statements that define the program and behaviour of your Hangman game. It consists of:

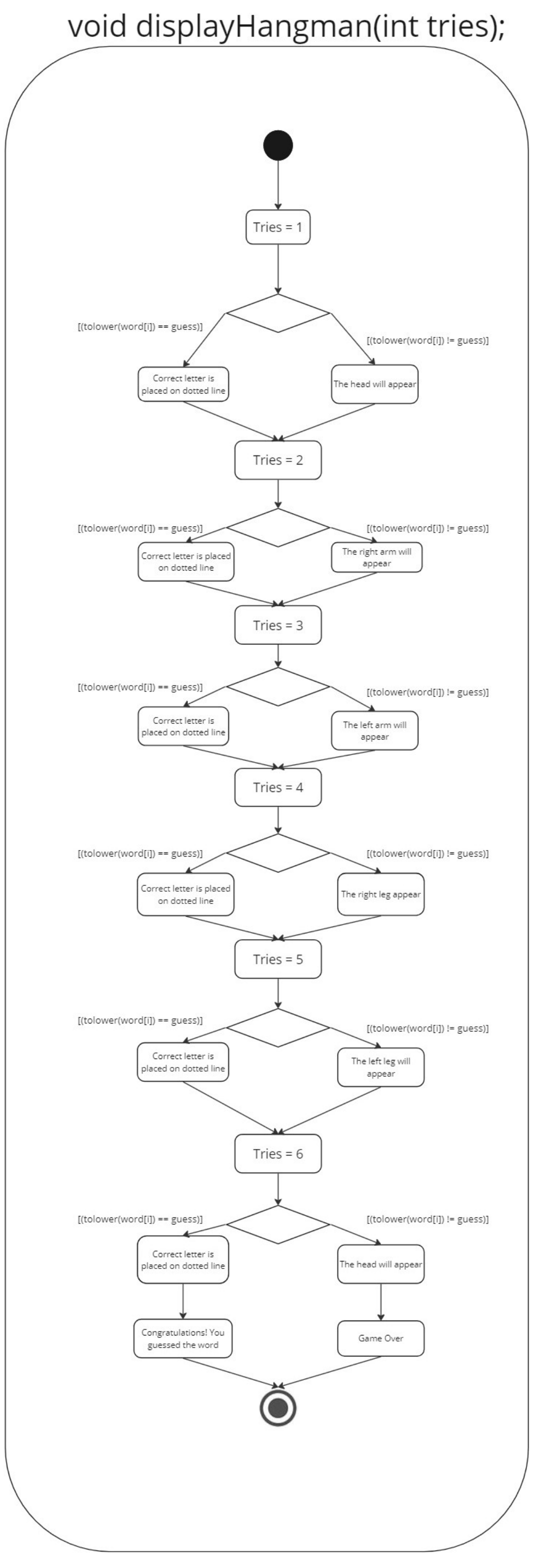
- Variable Declarations
- Initialization
- Game Loop
- Play Again Prompt
- Input Buffer Flushing
- Clearing the Screen
- End of Program

void displayHangman(int tries);

This function was used to generate the hangman figure for the game. The hangman was made using ASCII.

The figure is completed in six steps which is directly linked to the condition of number of tries.

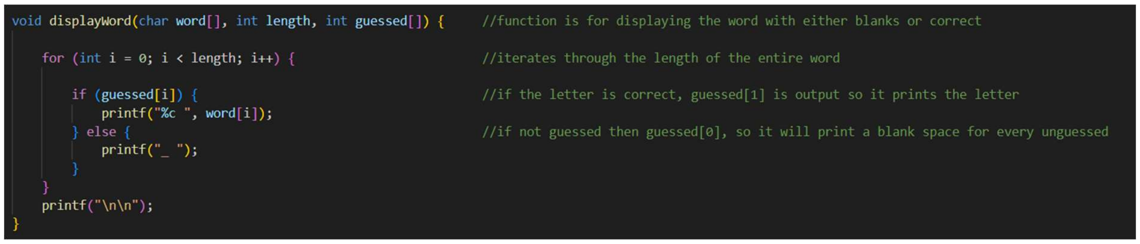
At 0 tries, only the gallows is displayed. With each subsequent try, a body part is shown until the sixth miss which is game over and the entire body is shown.


void displayHangman(int tries);
Tries = 1
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The head will appear
Tries = 2
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The right arm will appear
Tries = 3
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The left arm will appear
Tries = 4
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The right leg appear
Tries = 5
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The left leg will appear
Tries = 6
[(tolower(word[i]) == guess)]
[(tolower(word[i]) != guess)]
Correct letter is placed on dotted line
The head will appear
Congratulations! You guessed the word
Game Over

void displayWord(char word[], int length, int guessed[]);

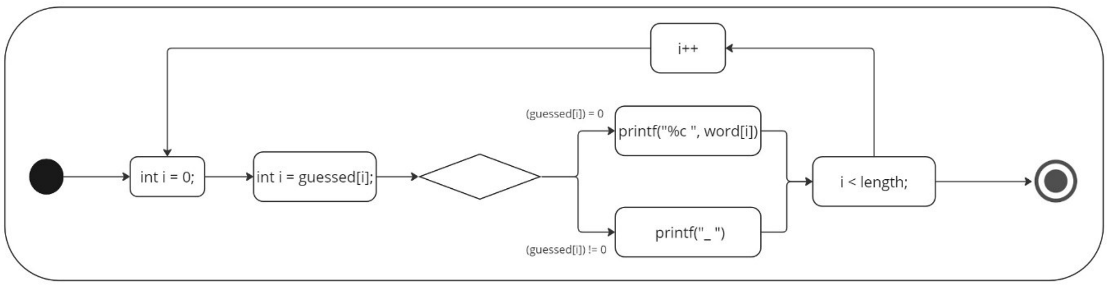
This function is used to display to the player the number of letters they have correctly guessed.

The for loop allows the function to first iterate through the entire length of the word. Then it checks whether (`word[i]`) has been guessed by looking at the corresponding value in the guessed array (`guessed[i]`). If (`guessed[i]`) = 1, then the guessed character is correct and then the statement is true so the game will replace the dotted line with correct letter. Otherwise, the game will print '_' with a space because the character hasn't been guessed yet.



```
void displayWord(char word[], int length, int guessed[]) {     //function is for displaying the word with either blanks or correct

    for (int i = 0; i < length; i++) {                         //iterates through the length of the entire word

        if (guessed[i]) {                                      //if the letter is correct, guessed[1] is output so it prints the letter
            printf("%c ", word[i]);
        } else {                                               //if not guessed then guessed[0], so it will print a blank space for every unguessed
            printf("_ ");
        }
    }
    printf("\n\n");
}
```

displayWord(char word[], int length, int guessed[])

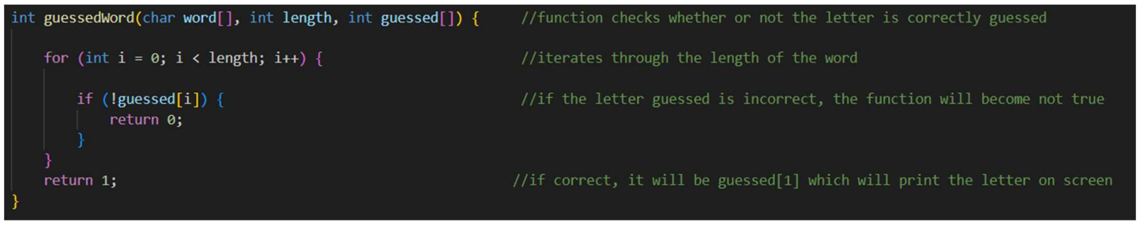


int guessedWord (char word[], int length, int guessed[]);

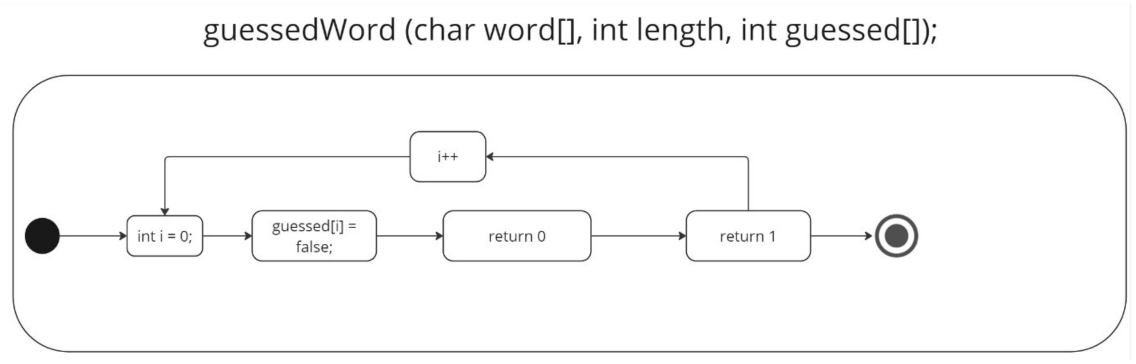
This function is used in the game logic to determine if the player has correctly guessed the word or not so that they can proceed to the next level or not.

In this function, the loop iterates through each character of the word from the first letter to the last. The `guessed[i]` is a Boolean condition so if it is 0 (it is false), the letter at position `i` (index) is has not been guessed. If any character in the word hasn't been guessed correctly, the function immediately returns 0, indicating that the word hasn't been guessed completely.

Hence, when the loop iterates through the word and finds that if every letter has been guessed correctly and if there are no unguessed letters left (meaning `guessed[i]` is 1) the function will return 1 and the word is correctly guessed.

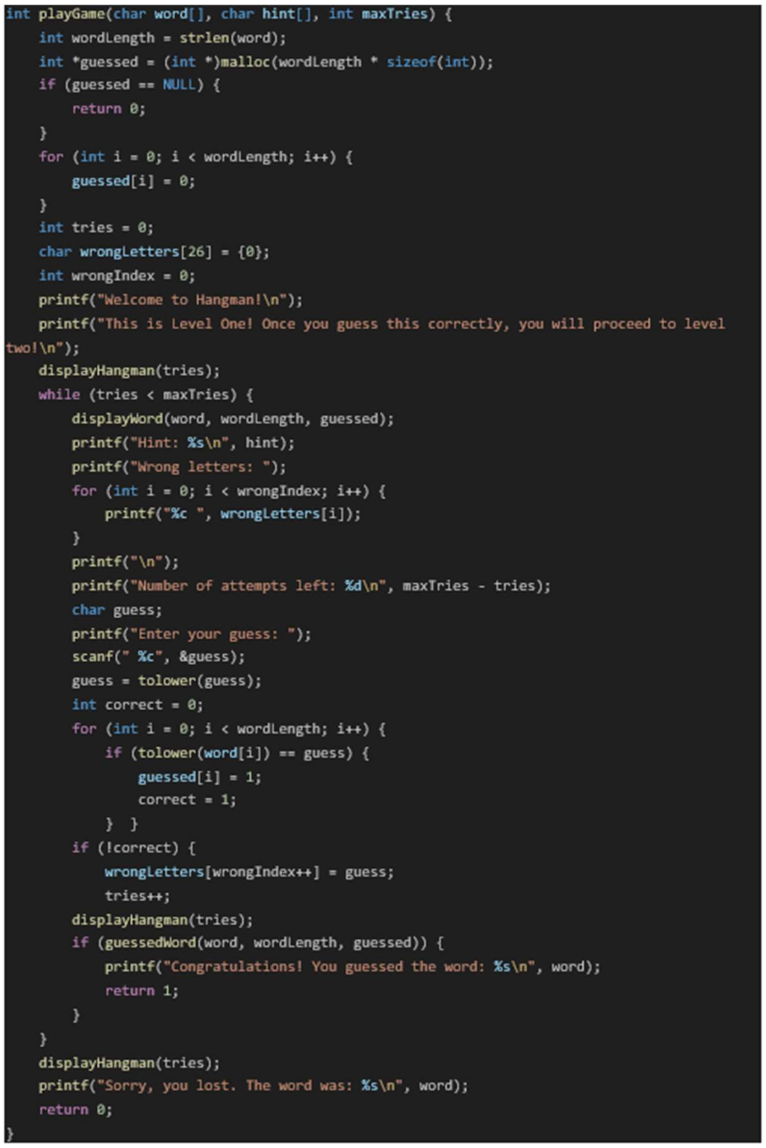


```
int guessedWord(char word[], int length, int guessed[]) {      //function checks whether or not the letter is correctly guessed

    for (int i = 0; i < length; i++) {                         //iterates through the length of the word

        if (!guessed[i]) {                                     //if the letter guessed is incorrect, the function will become not true
            return 0;
        }
    }
    return 1;                                                 //if correct, it will be guessed[1] which will print the letter on screen
}
```



int playGame(char word[], char hint[], int maxTries);

This is the main game logic of the game.



```c
int playGame(char word[], char hint[], int maxTries) {
    int wordLength = strlen(word);
    int *guessed = (int *)malloc(wordLength * sizeof(int));
    if (guessed == NULL) {
        return 0;
    }
    for (int i = 0; i < wordLength; i++) {
        guessed[i] = 0;
    }
    int tries = 0;
    char wrongLetters[26] = {0};
    int wrongIndex = 0;
    printf("Welcome to Hangman!\n");
    printf("This is Level One! Once you guess this correctly, you will proceed to level 
two!\n");
    displayHangman(tries);
    while (tries < maxTries) {
        displayWord(word, wordLength, guessed);
        printf("Hint: %s\n", hint);
        printf("Wrong letters: ");
        for (int i = 0; i < wrongIndex; i++) {
            printf("%c ", wrongLetters[i]);
        }
        printf("\n");
        printf("Number of attempts left: %d\n", maxTries - tries);
        char guess;
        printf("Enter your guess: ");
        scanf(" %c", &guess);
        guess = tolower(guess);
        int correct = 0;
        for (int i = 0; i < wordLength; i++) {
            if (tolower(word[i]) == guess) {
                guessed[i] = 1;
                correct = 1;
            } }
        if (!correct) {
            wrongLetters[wrongIndex++] = guess;
            tries++;
        displayHangman(tries);
        if (guessedWord(word, wordLength, guessed)) {
            printf("Congratulations! You guessed the word: %s\n", word);
            return 1;
        }
    }
    displayHangman(tries);
    printf("Sorry, you lost. The word was: %s\n", word);
    return 0;
}
```

The function would first calculate the entire length of the word and store in the `wordLength` variable which will then be used in the guessed array (`guessed[]`). The malloc function is used to calculate the size of the `guessed[]` which is not fixed since it is known after compilation as the word generated is random. This allows dynamic memory allocation.

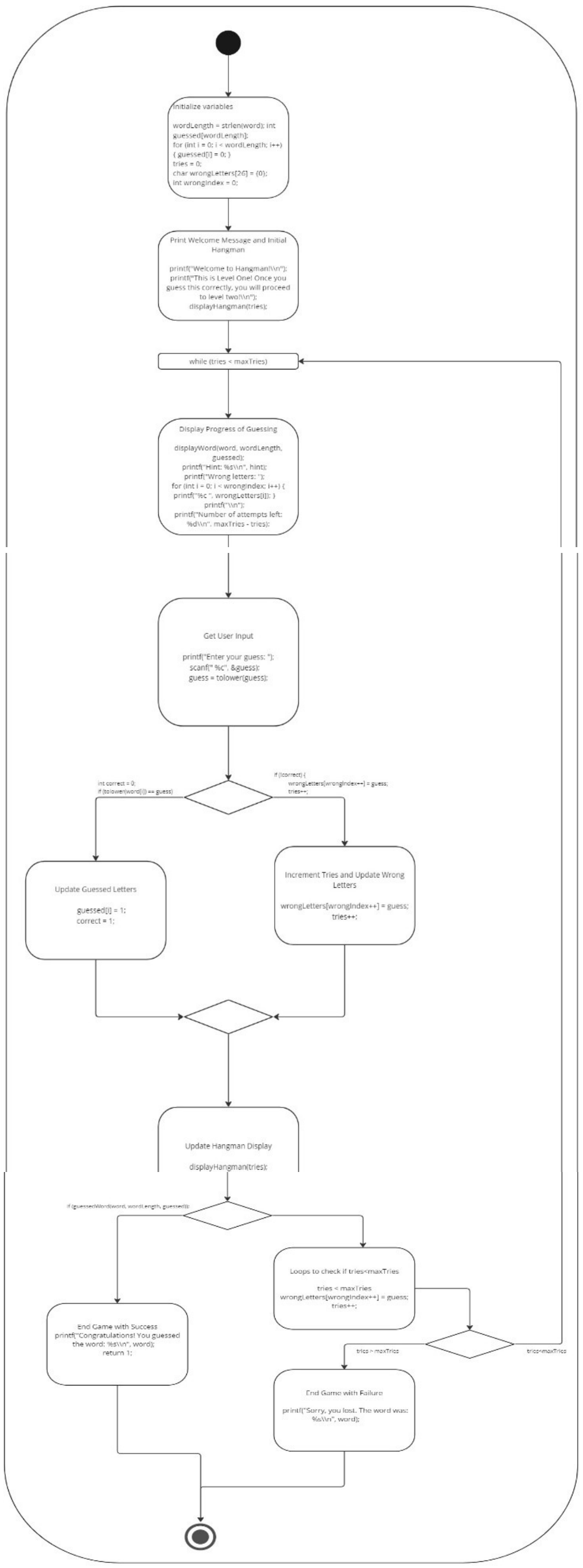
The `for` loop, iterates through the entire length of the word with the `guessed[]` initialized to zero. This is because at the start of the game none of the word is guessed.

The `while` loop ensures that the game runs as long as `tries` remain less than the `maxTries` which is the only condition we need. Then, the `displayWord` function is called to display the current state of the word. Then, the incorrect letters entered are shown which is used through a for loop which iterates through the entire word and tries to find that letter, if not found it is stored in the `wrongLetters` array. Likewise, the number of attempts left is shown through subtracting tries by `maxTries`.

The next block of code asks for the players input. The input is then went through a for loop where the letter is iterated with the characters in the word. A correct variable is initialized to zero assuming the game hasn't started yet and is incorrect. The loop searches for the letter in the word array and if it is found it is stored in the `guessed[]` which is then printed on screen, replacing the dashed line. The guess variable is incremented to one and so is correct variable. If not, then the letter is stored in the `wrongLetters` array and the `wrongIndex` is incremented by 1 to move to next position in the array. This increments `tries` to one.

Once correctly guessed, it returns the value of 1.

int playGame(char word[], char hint[], int maxTries);



Initialize variables

```
wordLength = strlen(word); int
guessed[wordLength];
for (int i = 0; i < wordLength; i++)
{ guessed[i] = 0; }
tries = 0;
char wrongLetters[26] = {0};
int wrongIndex = 0;
```

Print Welcome Message and Initial Hangman

```
printf("Welcome to Hangman!\\n");
printf("This is Level One! Once you
guess this correctly, you will proceed
to level two!\\n");
displayHangman(tries);
```

```
while (tries < maxTries)
```

Display Progress of Guessing

```
displayWord(word, wordLength,
guessed);
printf("Hint: %s\\n", hint);
printf("Wrong letters: ");
for (int i = 0; i < wrongIndex; i++) {
printf("%c ", wrongLetters[i]); }
printf("\\n");
printf("Number of attempts left:
%d\\n", maxTries - tries);
```

Get User Input

```
printf("Enter your guess: ");
scanf(" %c", &guess);
guess = tolower(guess);
```

```
int correct = 0;
if (tolower(word[i]) == guess)
```

```
if (!correct) {
    wrongLetters[wrongIndex++] = guess;
    tries++;
```

Update Guessed Letters

```
guessed[i] = 1;
correct = 1;
```

Increment Tries and Update Wrong Letters

```
wrongLetters[wrongIndex++] = guess;
tries++;
```

Update Hangman Display

```
displayHangman(tries);
```

```
if (guessedWord(word, wordLength, guessed));
```

Loops to check if tries<maxTries

```
tries < maxTries
wrongLetters[wrongIndex++] = guess;
tries++;
```

```
tries >= maxTries
tries<maxTries
```

End Game with Success

```
printf("Congratulations! You guessed
the word: %s\\n", word);
return 1;
```

End Game with Failure

```
printf("Sorry, you lost. The word was:
%s\\n", word);
```

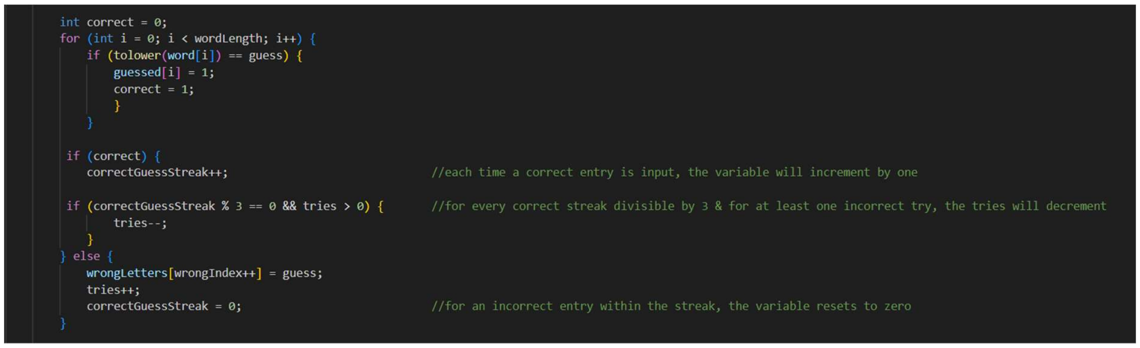
void levelTwo(char *options[], char *hints[], int numSpaces);

This follows the same logic as `int playGame(char word[], char hint[], int maxTries);`, but the main difference is the addition of another life when the user inputs correct characters three times in a row.

The variable is initially initialized to zero, and for every correct guess it is incremented by 1. In order for the addition of another life to happen, two conditions need to be met:

- the number of correct guesses in a row must be divisible by 3
- The number of `tries>0`

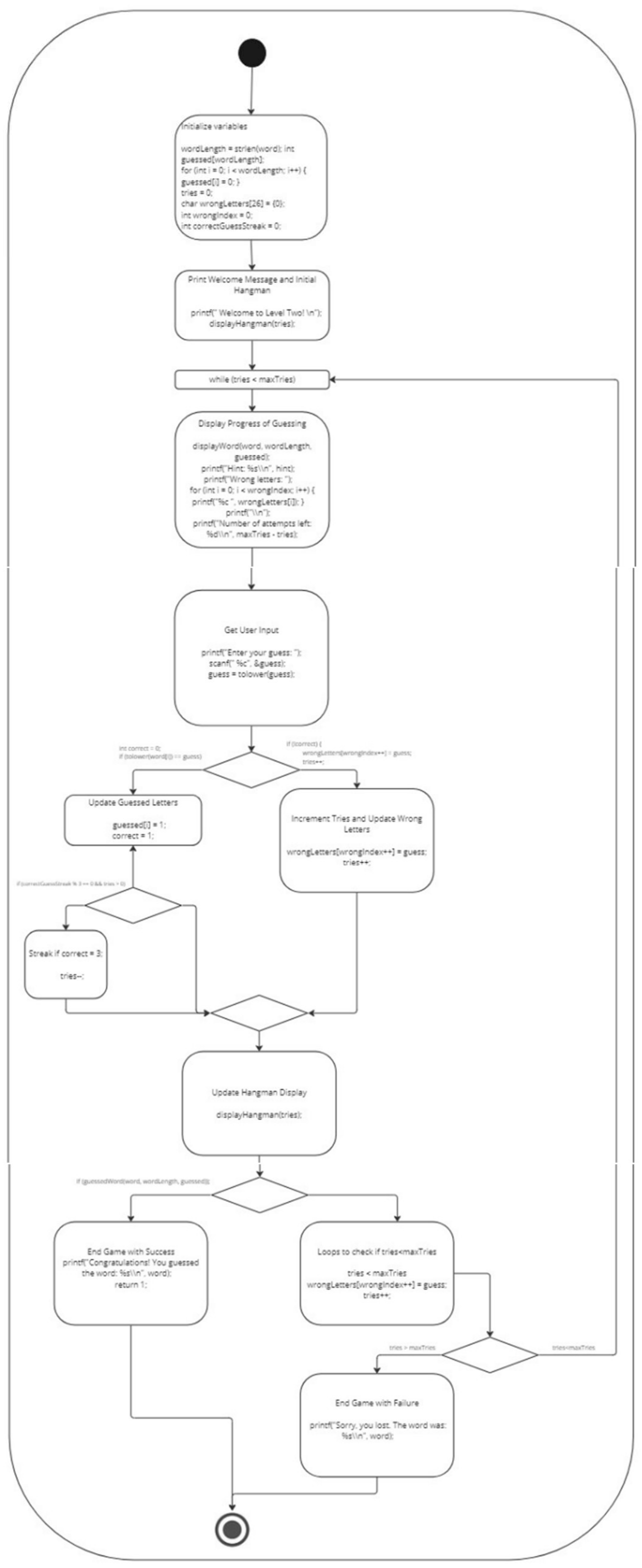
Once met, the number of tries is decremented by 1. Otherwise, a wrong guess will reset the variable to zero.



```c
    int correct = 0;
    for (int i = 0; i < wordLength; i++) {
        if (tolower(word[i]) == guess) {
            guessed[i] = 1;
            correct = 1;
            }
        }

     if (correct) {
        correctGuessStreak++;                              //each time a correct entry is input, the variable will increment by one

     if (correctGuessStreak % 3 == 0 && tries > 0) {       //for every correct streak divisible by 3 & for at least one incorrect try, the tries will decrement
            tries--;
        }
    } else {
        wrongLetters[wrongIndex++] = guess;
        tries++;
        correctGuessStreak = 0;                            //for an incorrect entry within the streak, the variable resets to zero
    }
```

0

void levelTwo(char *options[], char *hints[], int numSpaces);



```
wordLength = strlen(word); int
guessed[wordLength];
for (int i = 0; i < wordLength; i++) {
guessed[i] = 0; }
tries = 0;
char wrongLetters[26] = {0};
int wrongIndex = 0;
int correctGuessStreak = 0;
```

```
printf(" Welcome to Level Two! \n");
displayHangman(tries);
```

```
displayWord(word, wordLength,
guessed);
printf("Hint: %s\\n", hint);
printf("Wrong letters: ");
for (int i = 0; i < wrongIndex; i++) {
printf("%c ", wrongLetters[i]); }
printf("\\n");
printf("Number of attempts left:
%d\\n", maxTries - tries);
```

```
printf("Enter your guess: ");
scanf(" %c", &guess);
guess = tolower(guess);
```

```
int correct = 0;
if (tolower(word[i]) == guess)
```

```
if (!correct) {
    wrongLetters[wrongIndex++] = guess;
    tries++;
```

```
guessed[i] = 1;
correct = 1;
```

```
wrongLetters[wrongIndex++] = guess;
tries++;
```

```
if (correctGuessStreak % 3 == 0 && tries > 0)
```

```
tries--;
```

```
displayHangman(tries);
```

```
if (guessedWord(word, wordLength, guessed))
```

```
printf("Congratulations! You guessed
the word: %s\\n", word);
return 1;
```

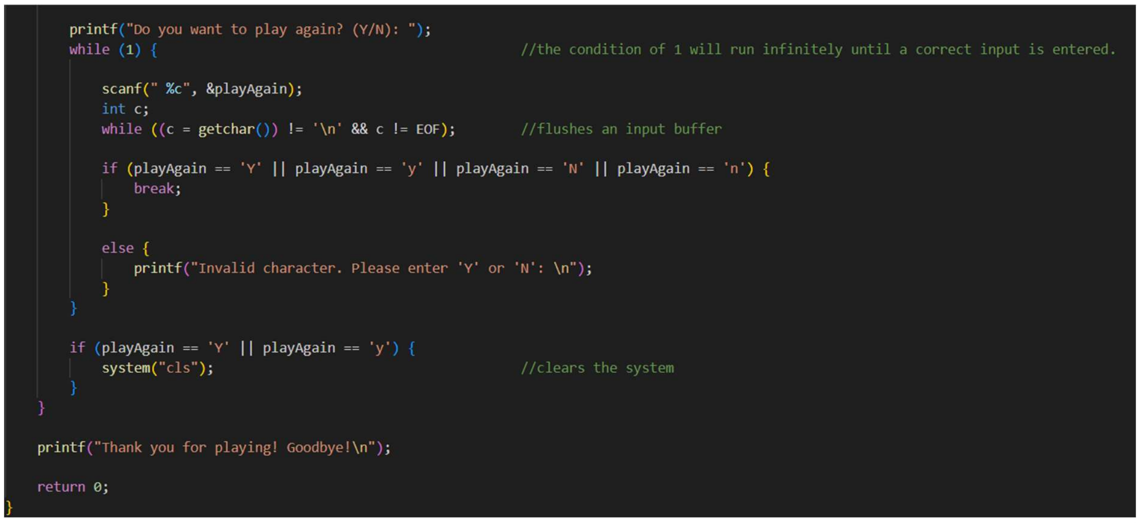
```
tries < maxTries
wrongLetters[wrongIndex++] = guess;
tries++;
```

```
printf("Sorry, you lost. The word was:
%s\\n", word);
```

Flushing input buffer and restarting

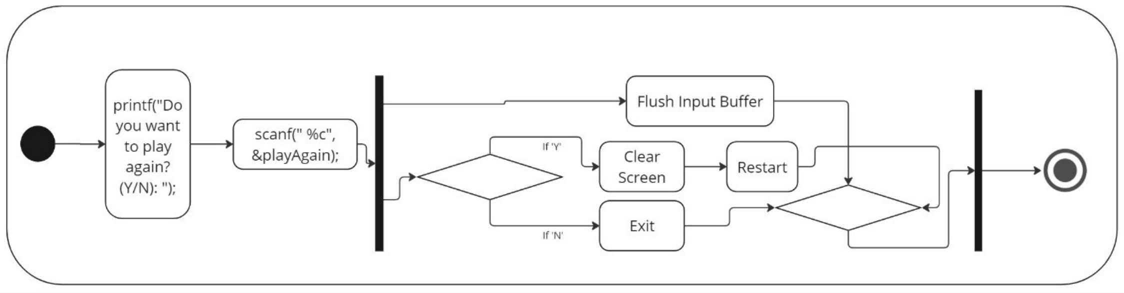
It achieves this by continuously reading characters from the input buffer until it encounters a newline character or the end of file (EOF). After prompting the player if they want to play again, the program reads the player's response using `scanf`. However, `scanf` leaves a newline character in the input buffer so the program calls `while function` clear the input buffer and not read it.

The variable `playAgain` stores the player's response. It's declared as a character and used to capture the player's decision whether to play again or not. After calling `scanf`, it holds the player's input of either Y or N. If the former, the `system("clear")` command clears the screen, providing a clean interface for the next game iteration/session and avoiding any memory rewrite, leak or overload. If the latter, then the game finishes.



```
        printf("Do you want to play again? (Y/N): ");
        while (1) {                                     //the condition of 1 will run infinitely until a correct input is entered.

            scanf(" %c", &playAgain);
            int c;
            while ((c = getchar()) != '\n' && c != EOF);        //flushes an input buffer

            if (playAgain == 'Y' || playAgain == 'y' || playAgain == 'N' || playAgain == 'n') {
                break;
            }

            else {
                printf("Invalid character. Please enter 'Y' or 'N': \n");
            }
        }

        if (playAgain == 'Y' || playAgain == 'y') {
            system("cls");                              //clears the system
        }
    }

    printf("Thank you for playing! Goodbye!\n");

    return 0;
}
```

Flush input buffer and restart


printf("Do you want to play again? (Y/N): ");
scanf(" %c", &playAgain);
Flush Input Buffer
If 'Y'
Clear Screen
Restart
If 'N'
Exit

Code:

Why such implementation?

a) As I was working alone, pointers were notorious for being tricky so I wanted to work with them to have some experience handling them.

b) As a beginner, Hangman seemed like a good project and added my own twists to not only use pointers but also use multiple if statements and nested loops to get a hang of it since I find both of them tricky.

Tools and Technology

a) I used Visual Studio code for writing the code as I find the colour differentiation helpful.

b) OnlineGDB and Visual Studio Code was used to compile the code.

c) Used ASCII value for making the figure hangman. It allows for easy manipulation, integrity, and error handling.

d) I acquired aid from ChatGPT for debugging and problems.

GitHub:

https://github.com/MoeezMufti/Hangman---SS24