***Abstract –* This report provides an update on the status and objectives of the Hangman Game project which involves building a simple, compact and entertaining game for users. To date, we have successfully implemented the core functionality of the game in our code, including random word selection from a predefined list, displaying the game instructions to begin and user input handling for single-character guesses. The game effectively tracks correct and incorrect guesses, updating the display after each user input, and concludes by either congratulating the player or revealing the correct word upon reaching the maximum allowed attempts. Additionally, challenges such as input validation and display formatting have been addressed, laying a solid foundation for further enhancements. Future work will focus on refining user interaction, improving error handling, and incorporating optional features such as scoring and multiple difficulty levels to enrich the overall user experience.**

**Keywords – Hangman, header file, function, ASCII**

1. INTRODUCTION

Using C Programming Language, we developed our version of the classic Hangman Game. This game presents the player with a number of blank spaces depending on a random word selected from a diversified dictionary. The player then guesses letters of the word. If the letter is present, it appears in all its correct positions among the blank spaces. Otherwise, one body part is attached to a human stick figure on the gallows. It allows six guesses, one for each different body part, until the stick figure is completed and the player loses. The player wins if they guess the word before their six chances are over.

1. PROGRAMMING ENVIRONMENT

The Hangman game is developed using Code::Blocks 20.03. It is an IDE (Integrated Development Environment) with a built-in GCC compiler.

1. METHODOLOGY
2. ***HEADER FILES***

The implementation utilizes the following standard library C header files to support required functionalities:

* **<stdio.h>:** Used for standard input and output operations.
* **<stdlib.h>:** Declares a number of standard functions for type conversions, memory allocation and other similar cases.
* **<string.h>:** Contains functions that manipulate strings.
* **<ctype.h>:** Declares character classification functions.
* <**time.h>**: Declares the time and date functions.

1. ***PREDEFINED VALUES***

The following constants were used to specify important game parameters:

* **#define MAX\_TRIES 6:** Sets the maximum number of incorrect guesses a player can make before losing the game.
* **#define WORD\_COUNT 6:** Declares the number of words available in the predefined word list.

1. ***VISUAL REPRESENTATION***

The hangman is going to be represented using a displayHangman() function. This function will store the multiple stages of the drawing using an array of strings progressing from the empty gallows to a fully drawn stick figure. It will execute based on an integer parameter defining how many guesses the player has used so far in the game. In order to add a touch of visual appeal and enhance the creative aspects of the project, we will utilise simple ASCII art for visualisation.

1. ***FUNCTIONS***

* displaytitle(): prints the title sequence upon starting the game
* displayword(): This function is in charge of showing the word’s current state. It takes two parameters:

1. word: A constant character pointer that represents the word to be guessed.
2. guessed: An integer array that keeps tracks of the correct guesses.

Through the for loop, the function iterates through each character in the word and checks if the guess was correct. If the guess is correct, which is indicated by a non-zero value in the guessed array, then it is printed. Otherwise, an underscore (\_) is printed followed by a space in order to represent the missing letter. Furthermore, a new line(\n) is printed to maintain proper formatting.

The user can see the letters they have properly identified while keeping unguessed letters buried thanks to this function, which gives them real-time feedback.

***E.*** ***INITIALIZING THE GAME: WORD SELECTION AND SETUP***

Before the player starts guessing, the code initialises key game components.

The game's title is initially printed by the 'displaytitle();' function, which is followed by a welcome message and an explanation of the rules, which also includes the maximum number of wrong guesses that can be made.

To ensure the word selection is random, the 'srand(time(0));' function seeds the random number generator using current time. Then a word is picked from the predefined wordlist (wordlist) using rand () % WORD\_COUNT. Thus, in every game session, a different word is used.

After word selection, the word length is determined by 'strlen();' and stored in the variable 'word\_length'. An integer array guessed[] is created where each index represents a letter in the selected word. The 'memset():' function initialises all values of guessed[] to zero indicating that no letters have been guessed yet.

The loop continues running as long as the number of tries is less than MAX\_TRIES. This ensures that the player has a limited number of attempts to guess the correct word before the game ends. Each iteration calls displayHangman(tries), which updates the visual representation of the Hangman based on the number of incorrect guesses. The loop also displays the word with correctly guessed letters, allowing the player to see their progress.

The program then prompts the user to enter a letter, ensuring that input is properly formatted by converting it to uppercase. A variable found is initialized to track whether the guessed letter exists in the word. This structure allows the program to check if the user's input matches any letter in the word and updates the display accordingly. The loop continues until the player either correctly guesses the word or reaches the maximum allowed attempts. The tries; variable keeps track of the number of incorrect guesses and the 'correct\_guesses;' variable counts the correct guesses and both of which are set to zero. Finally, to store the player's inputs throughout the gameplay, a character variable 'guess' is declared.

This setup ensures that the game begins with a randomised word and correctly monitors player progress throughout the session.

A for loop goes through every character in the selected word. For each character, it checks if it matches the player's guess and whether that letter hasn't already been revealed (using the guessed array). If the letter matches, the code marks it as guessed (guessed[i] = 1), sets the found flag to 1 to indicate that the guess was correct, and increments the correct\_guesses counter. If no match is found (i.e., found remains 0 after checking all letters), the code increases the number of incorrect tries. It then prints a message informing the player how many attempts remain.

After processing the guess, the program checks if the number of correctly guessed letters equals the total number of letters in the word. If they match, the player has successfully guessed the entire word, and a congratulatory message is printed. The loop then breaks, ending the game. Once the loop finishes (either by running out of tries or by guessing the word), the code checks if the word was fully guessed. If not, it prints a “GAME OVER” message along with the correct word.

1. CHALLENGES

* **Input Handling:** Ensuring that user input is correctly processed remains a focus. Minor issues have been observed with handling extraneous characters and ensuring robust validation.
* **Visual Feedback Integration:** Although the ASCII-art for the hangman display has been developed, its integration into the main game loop is pending final testing. The plan is to uncomment and refine the display function so that players can see a visual representation of their progress after each guess.
* **Code Efficiency:** Optimizing string handling and array management in C has been an area of continued improvement, with a focus on ensuring that the program runs efficiently on various platforms.

1. CONCLUSION

The project is on track, with core gameplay mechanics implemented and functional. Addressing the remaining challenges and integrating visual feedback will enhance the overall user experience. We look forward to completing these improvements in the next development phase.