Project Report: Hangman Game

1. Abstract

The Hangman Game project is a implementation of the classic word-guessing game. This report provides an overview of the project's objectives, design, development, and testing, as well as future enhancements.

The project's primary objectives are to create an engaging and user-friendly Hangman game interface, implement core game logic for word selection and letter guessing, and ensure a seamless user experience. This report presents a comprehensive overview of the project, detailing its development process, design choices, testing methodologies, and future enhancement possibilities.

Purpose:

The primary purpose of this project is to create an entertaining and educational Hangman game. It offers users an opportunity to enjoy a digital word puzzle while showcasing the developer's programming skills.

Scope:

Gameplay: Users can guess letters to uncover a hidden word.

User Interface: The game features a text-based user interface for simplicity.

Word Selection: Words are randomly selected from a predefined list.

Feedback: Immediate feedback on correct and incorrect guesses.

Testing: Rigorous testing ensures game functionality and reliability.

Future Enhancements: The project lays the foundation for potential future enhancements and features.

Key Features:

Interactive Gameplay: Users can guess letters to reveal the hidden word.

User-Friendly Interface: A clear and intuitive text-based interface.

Word Selection: Words are randomly chosen from a list, providing variety.

Immediate Feedback: Users receive instant feedback on their guesses.

Testing: Comprehensive testing ensures the game's reliability and functionality.

Educational Value: Serves as an educational tool for learning Python programming and game development.

Future Enhancements: Designed to accommodate potential future features and improvements.

In summary, the Hangman Game project offers a digital recreation of a classic word-guessing game, providing both entertainment and educational value while highlighting the developer's programming skills. Its scope encompasses interactive gameplay, an intuitive user interface, word selection, and the potential for future enhancements.

2. Introduction

2.1 Background

The Hangman game is a classic word-guessing game that provides entertainment and tests vocabulary and word recognition skills. This project was developed to create a fun and interactive Hangman game with a unique twist, where players guess the names of popular websites.

2.2 Objectives

The main objectives of this project were to:

Develop an engaging Hangman game with a user-friendly interface.

Create a challenging word-guessing experience with a limited number of incorrect guesses allowed.

Implement a unique theme where players guess website names.

3. Technologies Used

The technologies and tools used in this project include:

C programming language

Standard C libraries (stdio.h, stdlib.h, string.h, etc.)

Random number generation using srand and rand

Time functions for seed initialization

Command-line interface

4. System Architecture

4.1 Front-End

The front-end of the Hangman game is a command-line interface. It handles user input, displays the hangman figure, and updates the word being guessed.

4.2 Back-End

The back-end manages the game logic, including word selection, letter validation, and game state management. It communicates with the front-end to display the game's progress.

4.3 Database

The game does not use a traditional database, as word lists are hardcoded within the code itself.

5. Project Modules

5.1 Module 1: Word Decryption

This module decrypts a randomly selected word from the hardcoded list of website names.

5.2 Module 2: Game Logic

Handles the core game logic, including checking guessed letters, tracking mistakes, and determining the game's outcome.

6. Design and Implementation

6.1 Front-End Design

The front-end is designed for a command-line interface, providing a simple and intuitive user experience. It displays the hangman figure, the word to be guessed, and the player's progress.

6.2 Back-End Design

The back-end is responsible for selecting a random word, validating player guesses, and managing the game's state, including tracking mistakes.

6.3 Database Design

No traditional database is used. Instead, a hardcoded list of website names is stored within the code.

7. Features and Functionality

7.1 Feature 1: Word Guessing

Players can guess letters one at a time to reveal the hidden word.

7.2 Feature 2: Limited Mistakes

The game allows a maximum of 6 incorrect guesses before the player loses.

8. Testing

8.1 Unit Testing

Unit testing was conducted to ensure the correctness of critical functions and game logic.

8.2 Integration Testing

Integration testing verified the interaction between the front-end and back-end components.

8.3 User Acceptance Testing

User acceptance testing involved real players testing the game to ensure a satisfying user experience.

9. Challenges Faced

Challenges during development included managing the game's state, ensuring fair word selection, and creating an intuitive command-line interface.

10. Future Enhancements

Possible future enhancements include adding more words, difficulty levels, and improved user interface elements.

11. Conclusion

This code is an implementation of the classic game "Hangman" in C. In the game, the player tries to guess a hidden word by suggesting letters one at a time. If they make too many incorrect guesses, a hangman figure is gradually drawn. The player wins if they guess all the letters in the word before the hangman is fully drawn, and they lose if the hangman is completed before they guess the word.

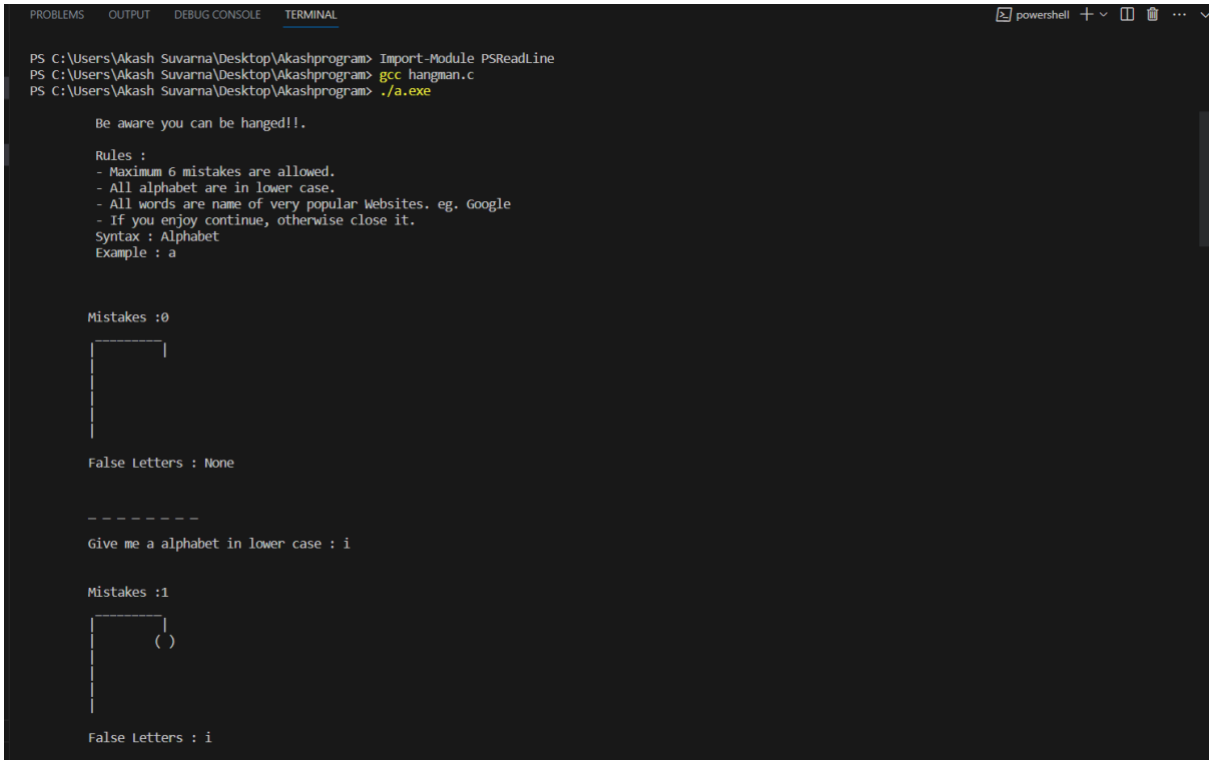
12. References

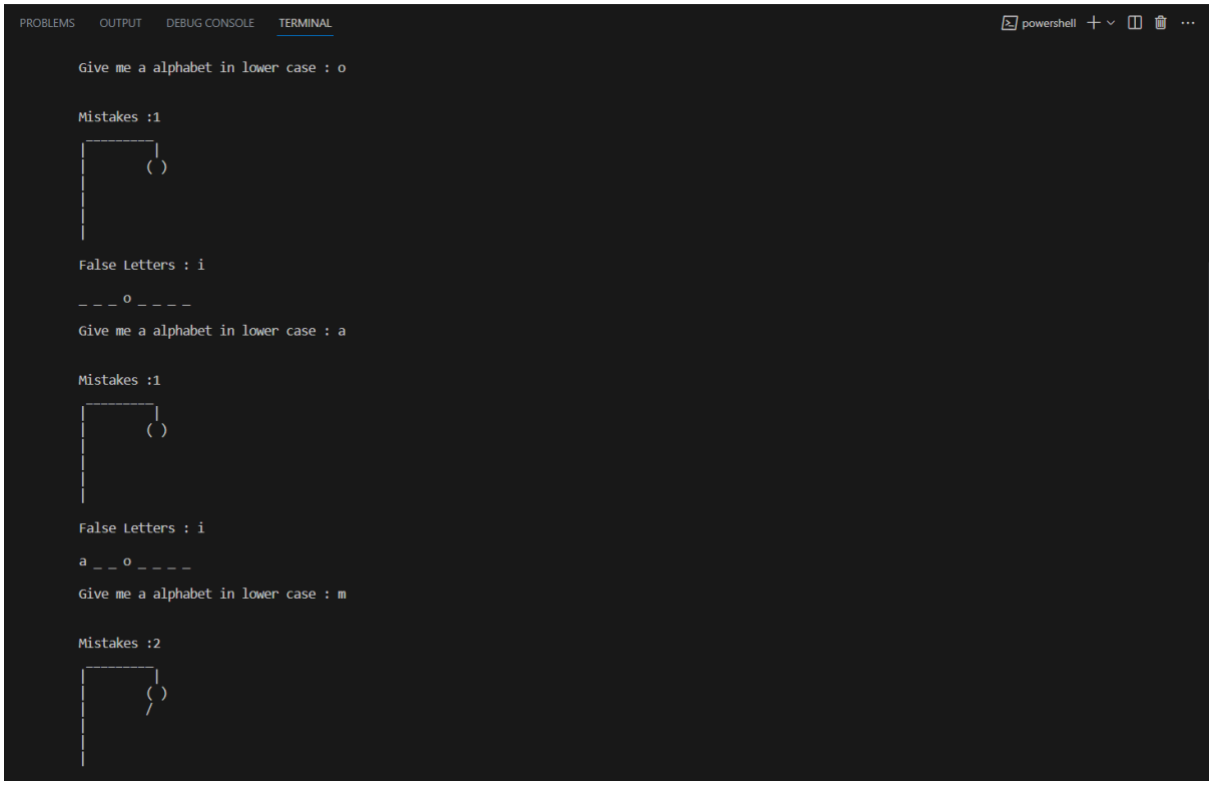
1. https://www.hackerrank.com/

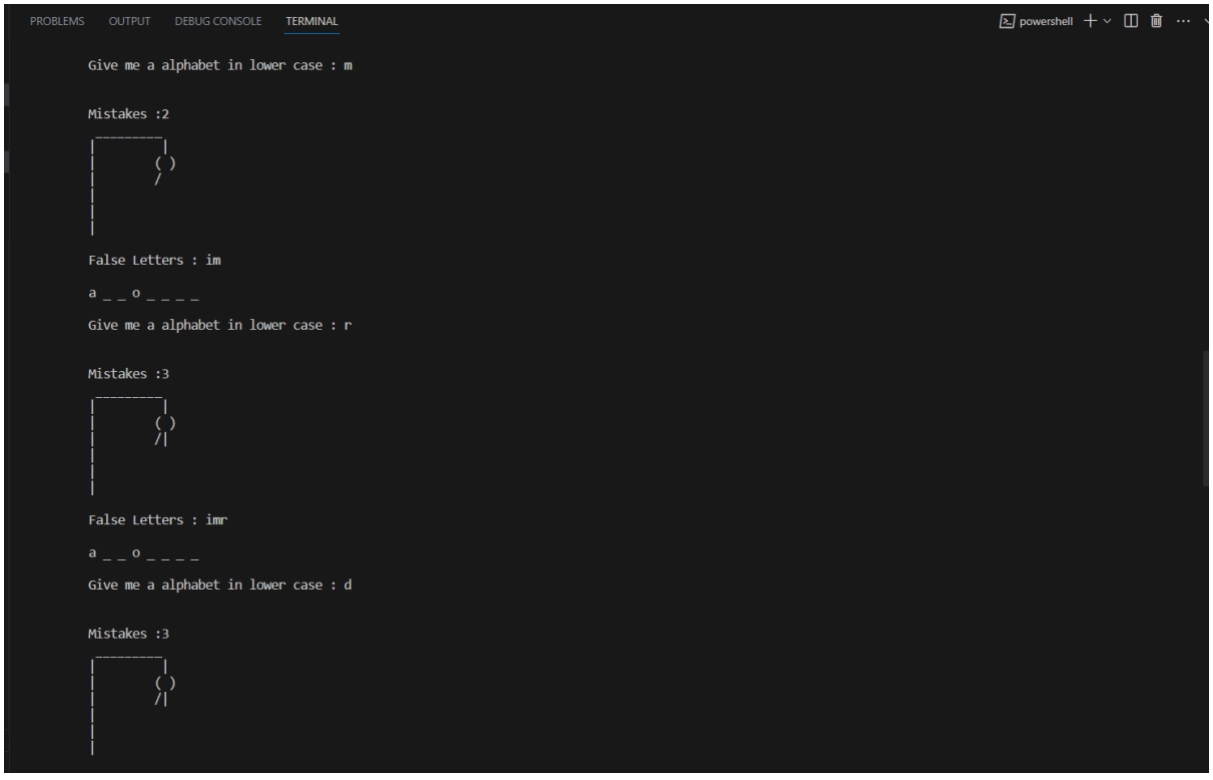
2. https://leetcode.com/

13. Appendices

13.1 Screenshots

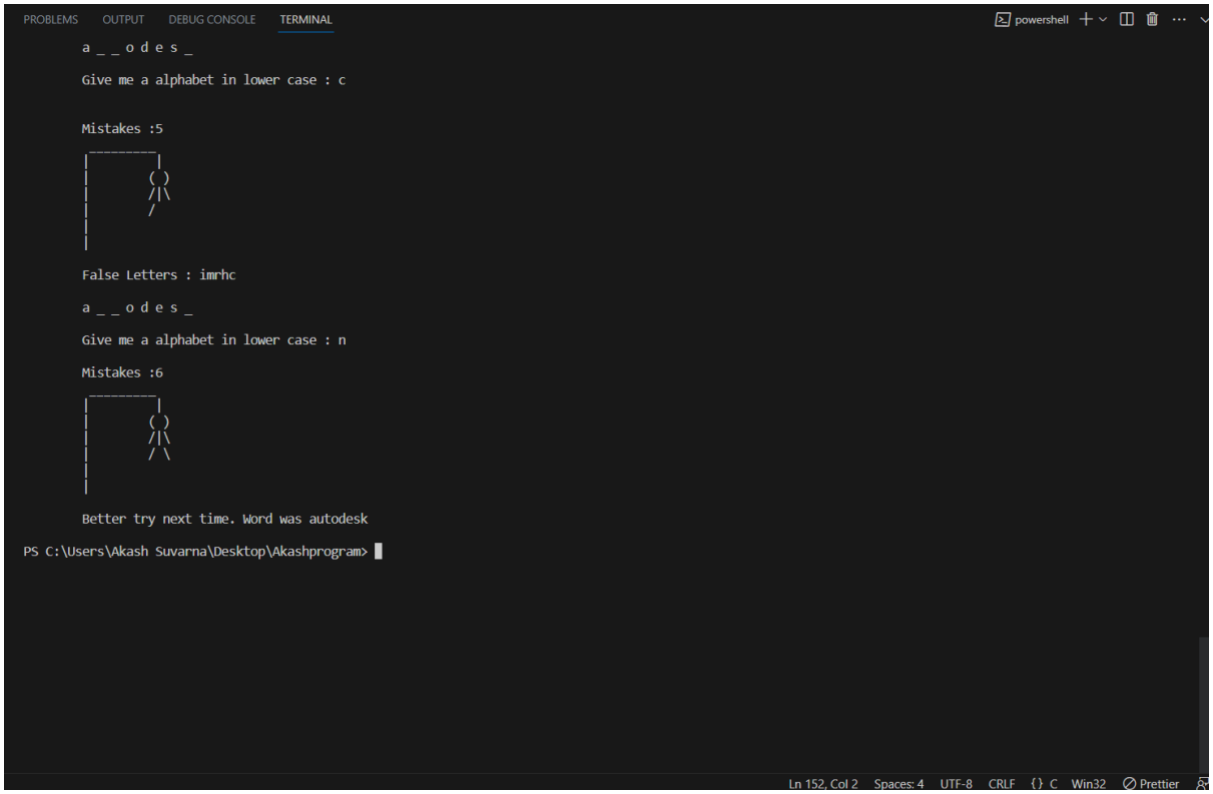



PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
powershell
Give me a alphabet in lower case : o
Mistakes :1
False Letters : i
_ _ _ o _ _ _ _
Give me a alphabet in lower case : a
Mistakes :1
False Letters : i
a _ _ o _ _ _ _
Give me a alphabet in lower case : m
Mistakes :2


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
powershell
Give me a alphabet in lower case : m
Mistakes :2
False Letters : im
a _ _ o _ _ _ _
Give me a alphabet in lower case : r
Mistakes :3
False Letters : imr
a _ _ o _ _ _ _
Give me a alphabet in lower case : d
Mistakes :3

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  powershell

a _ _ o _ _ _ _

Give me a alphabet in lower case : d

Mistakes :3
 _________
|         |
|        ( )
|        /|
|
|
|

False Letters : imr

a _ _ o d _ _ _

Give me a alphabet in lower case : e

Mistakes :3
 _________
|         |
|        ( )
|        /|
|
|
|

False Letters : imr

a _ _ o d e _ _

Give me a alphabet in lower case : s

Mistakes :3
 _________
|         |
|        ( )
|        /|
|
|
|
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  powershell

a _ _ o d e _ _

Give me a alphabet in lower case : s

Mistakes :3
 _________
|         |
|        ( )
|        /|
|
|
|

False Letters : imr

a _ _ o d e s _

Give me a alphabet in lower case : h

Mistakes :4
 _________
|         |
|        ( )
|        /|\
|
|
|

False Letters : imrh

a _ _ o d e s _

Give me a alphabet in lower case : c

Mistakes :5
 _________
|         |
|        ( )
|        /|\
|        /
|

Ln 152, Col 2   Spaces: 4   UTF-8   CRLF   {} C   Win32   Prettier
```



13.2 Code Snippets

```
/*
* @name Hangman

*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <string.h>

#define WORDS 10
#define WORDLEN 40
```

```
#define CHANCE 6

bool srand_called = false;

int i_rnd(int i) {
   if (!srand_called) {
      srand(time(NULL) << 10);
      srand_called = true;
   }
   return rand() % i;
}

char* decrypt(char* code) {
        int hash = ((strlen(code) - 3) / 3) + 2;
        char* decrypt = malloc(hash);
        char* toFree = decrypt;
        char* word = code;
        for (int ch = *code; ch != '\0'; ch = *(++code))
        {
                if((code - word + 2) % 3  == 1){
                        *(decrypt++) = ch - (word - code + 1) - hash;
                }
        }
        *decrypt = '\0';
        return toFree;
}

void printBody(int mistakes, char* body) {
        printf("\tMistakes :%d\n", mistakes);
```

```
	switch(mistakes) {

		case 6: body[6] = '\\'; break;
		case 5: body[5] = '/'; break;
		case 4: body[4] = '\\'; break;
		case 3: body[3] = '|'; break;
		case 2: body[2] = '/'; break;
		case 1: body[1] = ')', body[0] = '('; break;
		default: break;

	}

	printf("\t _________\n"
	     "\t|        |\n"
	     "\t|       %c %c\n"
	     "\t|       %c%c%c\n"
	     "\t|       %c %c\n"
	     "\t|           \n"
	     "\t|           ", body[0], body[1], body[2],
	     body[3], body[4], body[5], body[6]);
}

void printWord(char* guess, int len) {
	printf("\t");
	for (int i = 0; i < len; ++i)
	{
		printf("%c ", guess[i]);
	}
	printf("\n\n");
```

```
}

int main() {

	printf("\n\t Be aware you can be hanged!!.");

	printf("\n\n\t Rules : ");
	printf("\n\t - Maximum 6 mistakes are allowed.");
	printf("\n\t - All alphabet are in lower case.");
	printf("\n\t - All words are name of very popular Websites. eg. Google");
	printf("\n\t - If you enjoy continue, otherwise close it.");

	printf("\n\t Syntax : Alphabet");
	printf("\n\t Example : a \n\n");

	char values[WORDS][WORDLEN] =
{"N~mqOlJ^tZletXodeYgs","gCnDIfFQe^CdP^^B{hZpeLA^hv","7urtrtwQv{dt`>^}FaR]i]XUug^GI",

	"aSwfXsxOsWAlXScVQmjAWJG","cruD=idduvUdr=gmcauCmg]","BQt`zncypFVjvIaTl]u=_?Aa}F",

	"iLvkKdT`yu~mWj[^gcO|","jSiLyzJ=vPmnv^`N]^>ViAC^z_","xo|RqqhO|nNstjmzfiuoiFfhwtdh~",

	"OHkttvxdp|[nnW]Drgaomdq"};
	char *body = malloc(CHANCE+1);

	int id = i_rnd(WORDS);
	char *word = decrypt(values[id]);
	int len = strlen(word);
	char *guessed = malloc(len);
```

```
char falseWord[CHANCE];

memset(body, ' ', CHANCE+1);
memset(guessed, '_', len);
char guess;
bool found;
char* win;

int mistakes = 0;
setvbuf(stdin, NULL, _IONBF, 0);

do {

        found = false;
        printf("\n\n");
        printBody(mistakes, body);
        printf("\n\n");
        printf("\tFalse Letters : ");
        if(mistakes == 0) printf("None\n");
        for (int i = 0; i < mistakes; ++i)
        {
                printf("%c", falseWord[i]);
        }
        printf("\n\n");
        printWord(guessed, len);
        printf("\tGive me a alphabet in lower case : ");
        do {scanf("%c",&guess);} while ( getchar() != '\n' );
        for (int i = 0; i < len; ++i)
        {
```

```
                        if(word[i] == guess) {
                                found = true;
                                guessed[i] = guess;
                        }
                }
                if(!found) {
                        falseWord[mistakes] = guess;
                        mistakes += 1;
                }
                win = strchr(guessed, '_');
        }while(mistakes < CHANCE && win != NULL);

        if(win == NULL) {
                printf("\n");
                printWord(guessed, len);
                printf("\n\tCongrats! You have won : %s\n\n", word);
        } else {
                printf("\n");
                printBody(mistakes, body);
                printf("\n\n\tBetter try next time. Word was %s\n\n", word);
        }

        free(body);
        free(word);
        free(guessed);
        return EXIT_SUCCESS;
}
```