SNAKE AND LADDERS GAME

A Course End Project Report – ADVANCED DATA STRUCTURESLABORATORY (A8511)

Submitted in the Partial Fulfilment of the

Requirements

for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

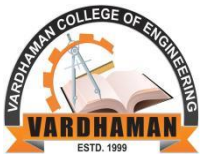
Submitted

By

P Ruthu Yadav **22881A0539**

Under the Esteemed Guidance of

Mr. M. Naresh Goud
Professor
CSE



Department of Computer Science and Engineering

VARDHAMAN COLLEGE OF ENGINEERING, HYDERABAD

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified

Kacharam, Shamshabad, Hyderabad - 501218, Telangana, India

January, 2024

VARDHAMAN COLLEGE OF ENGINEERING, HYDERABAD

An autonomous institute affiliated to JNTUH

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Course End Project titled **"Snakes and Ladders Game" is** carried out by **P Ruthu Yadav,** Roll Number **22881A0539** towards **A8602 – Advanced Data Structures Laboratory** course and submitted to **Department of Computer Science and Engineering**, in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Department of Computer Science and Engineering** during the Academic year 2023-24.

Name & Signature of the Instructor

Mr.M. Naresh Goud
Professor, CSE

Name & Signature of the HOD

Dr. Ramesh Karnati
HOD, CSE

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Mr.M.Rohith Naidu,** Professor, Department of Computer Science and Engineering, Vardhaman College of Engineering, for her able guidance and useful suggestions, which helped us in completing the design part of potential project in time.

We particularly thankful to **Dr. Ramesh Karnati**, Associate Professor & Head, Department of Computer Science and Engineering for his guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honorable Principal **Dr. J.V.R.Ravindra**, for having provided all the facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartfelt thanks to **Dr. Teegala Vijender Reddy**, Chairman and **Sri Teegala Upender Reddy**, Secretary of VCE, for providing a congenial atmosphere to complete this project successfully.

We also thank all the staff members of Computer Science and Engineering for their valuable support and generous advice. Finally, thanks to all our friends and family members for their continuous support and enthusiastic help.

Name of the Student : P.Ruthu Yadav

Roll Number : 22881A0539

INDEX

➢ Introduction

◎This C code implements a simple Hangman game, a classic word-guessing game. The program allows users to input a list of words, selects a word randomly, and then prompts the player to guess the letters of the word. The game provides visual feedback on correct and incorrect guesses, keeping track of used letters and displaying the progress of the guessed word. The player has a limited number of attempts to guess the word correctly, and the game ends either when the word is guessed or when the maximum allowed attempts are reached. The code uses functions, arrays, loops, and conditional statements to structure and execute the Hangman game logic.

➢ Problem Statement

◎ Develop a Hangman game in C where users input a list of words. The program randomly selects a word and prompts the player to guess its letters. Display the progress, track used letters, and limit attempts. The game concludes upon correct word guessing or reaching the maximum allowed attempts.

➢ Objectives

- The objectives of this project are to:
  - Develop a mechanism for random word selection from a user-input list.
  - Create a user input system for guessing letters, ensuring input validity.
  - Implement a clear display for tracking the progress of the guessed word and used letters.
  - Define the logic for concluding the game upon correct word guessing or reaching maximum attempts.
  - Organize code for readability, efficiency, and memory management.
- **Primary objectives are -**

1) Word Selection:

- Randomly select a word from a user-input list.

2) Game Logic:

- Implement the Hangman game logic, allowing players to guess letters and providing feedback.

3) User Interaction:

- Facilitate user input for guessing letters and display the game state.

4) Feedback and Progress:

- Display the progress of the guessed word, track used letters, and provide feedback on correct and incorrect guesses.

5) Game Conclusion:

- End the game when the word is correctly guessed or the maximum allowed attempts are reached.

6) Code Structure:

- Organize the code using functions, appropriate data structures, and clear logic to enhance readability and maintainability.

7) Difficulty Levels:

- Implement multiple difficulty levels, adjusting factors such as word length or the number of allowed incorrect attempts to cater to different player preferences.

8) Score keeping:

- Introduce a scoring system to track and display the player's performance, considering factors like the number of attempts and overall success rate.

9) User Feedback:

- Enhance user feedback by providing hints, additional information, or encouraging messages to keep players engaged and motivated.

10) **Multiplayer Support**:

- Extend the game to support multiplayer functionality, allowing multiple players to take turns guessing or competing against each other

➢ SYSTEM RECQUIREMENTS

Algorithms:

1) **Word Selection Algorithm:**

   - **Objective:** Randomly select a word from the user-input list.

- **Algorithm:**

1. Obtain the total number of words in the list.

2. Generate a random index within the range of the list size.

2) **Guess Validation Algorithm:**

- **Objective:** Validate and process user input for guessing letters.

- **Algorithm:**

1. Ensure the input is a single alphabetical character.
2. Convert the input to lowercase for case-insensitive matching.
3. Check if the input has not been guessed before.

3) **Game Progress Update Algorithm:**

- **Objective:** Update and display the progress of the guessed word.

- **Algorithm:**

1. Iterate through the characters of the target word.
2. If a guessed letter matches, update the display; otherwise, keep underscores.

4) **Game Conclusion Algorithm:**

- **Objective:** Determine when the game should conclude.

- **Algorithm:**

1. Track the number of incorrect guesses.
2. Check if the guessed word matches the target word.
3. Check if the maximum allowed incorrect attempts have been reached.

5) **Randomization Algorithm (if not using built-in functions):**

- **Objective:** Generate random numbers for word selection.

- **Algorithm:**

1. Seed the random number generator with a unique value.
2. Use the generated random number within the specified range.

These algorithms collectively form the core logic of the Hangman game, providing functionality for word selection, user input validation, game progress tracking, and determining game conclusion conditions.

➢ Data Structures

In the implementation of a Hangman game, you can use several data structures to manage the game state and perform various operations efficiently. Here are some data structures that can be employed:

1. Arrays :

- **Usage:** To store the list of words, track guessed letters, and represent the progress of the guessed word.

- **Example:** An array to hold the list of words, an array to track guessed letters, and an array to represent the progress of the word.

2. Linked Lists:

- **Usage:** To dynamically manage and store the linked nodes representing the characters of the word.

- **Example:** A linked list to represent the characters of the target word.

3. Queue:

- **Usage:** To enqueue and dequeue letters, especially useful for games with a predefined order of guesses.

- **Example:** A queue to manage the order of guessed letters.

4. Hash Set:

- **Usage:** To efficiently check if a letter has been guessed before, preventing duplicate guesses.

- **Example:** A hash set to store guessed letters for quick lookups.

5. Dynamic Memory Allocation (Heap):

- **Usage:** To allocate memory dynamically, especially when the length of words is not fixed.

- Example: Use `malloc` for dynamic memory allocation when dealing with words of varying lengths.

6. Structures:

- **Usage:** To encapsulate related data elements, providing a convenient way to organize game-related information.

- **Example:** A structure to represent the game state, including the target word, guessed letters, and game progress.

- These data structures can be chosen based on the specific requirements of your Hangman game implementation, considering factors such as memory efficiency, ease of manipulation, and the overall design of your program.

➢ DESIGN&IMPLEMENTATION

Design/Solution:

Certainly, let's outline a high-level design for the Hangman game solution. This will include key components, functions, and their interactions:

1. Main Game Loop:

- Manages the overall flow of the game.
- Calls functions for word selection, user input, game progress update, and checking game conclusion.

2. Word Selection:

- Randomly selects a word from the user-input list.
- Utilizes an algorithm to ensure fair randomization.

3. User Input Handling:

- Accepts and validates user input for guessing letters.
- Converts input to lowercase for case-insensitive matching.
- Ensures input is a single alphabetical character.

4. Game Progress Update:

- Updates and displays the progress of the guessed word.
- Utilizes a data structure (e.g., array or linked list) to represent the word and its progress.

5. Game Conclusion Logic:

- Determines when the game should conclude.
- Tracks the number of incorrect guesses.
- Checks if the guessed word matches the target word.
- Checks if the maximum allowed incorrect attempts have been reached.

6. Data Structures:

- Arrays for word lists, tracking guessed letters, and representing the progress of the word.
- Linked lists or structures to manage the characters of the target word.
- A queue for managing the order of guessed letters.
- A hash set for efficient checking of duplicate guesses.

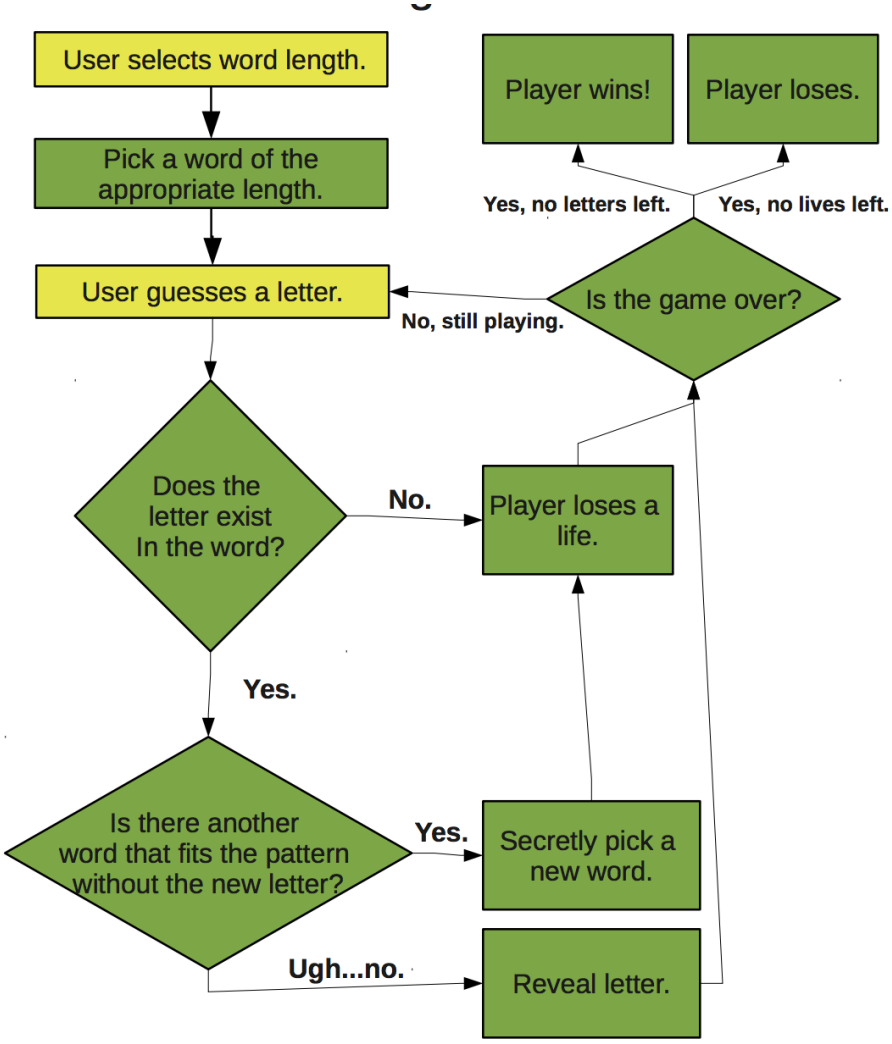
7. Randomization:

- Seeds the random number generator with a unique value.
- Uses the generated random number within the specified range for word selection.

8. Dynamic Memory Allocation:

- Utilizes dynamic memory allocation (e.g., `malloc`) for variable-length words.

➢ Flow Chart



SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_WORD_LENGTH 20
// Queue node structure
struct Node {
    char data;
    struct Node* next;
};
// Queue structure
struct Queue {
    struct Node* front;
    struct Node* rear;
};
// Initialize an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    if (queue == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
    queue->front = queue->rear = NULL;
    return queue;
}
// Enqueue a character to the queue
void enqueue(struct Queue* queue, char data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }

    newNode->data = data;
```

```
    newNode->next = NULL;
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
// Dequeue a character from the queue
char dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        fprintf(stderr, "Queue is empty\n");
        exit(EXIT_FAILURE);
    }
    char data = queue->front->data;
    struct Node* temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    return data;
}
// Display the word with underscores for unguessed letters
void displayWord(const struct Queue* wordQueue, const char* guessedLetters) {
    struct Node* current = wordQueue->front;
    while (current != NULL) {
        if (strchr(guessedLetters, current->data) != NULL) {
            printf("%c ", current->data);
        } else {
            printf("_ ");
        }
        current = current->next;

    }
```

```
    printf("\n");
}
// Check if the word has been completely guessed
int isGameOver(const struct Queue* wordQueue, const char* guessedLetters) {
    struct Node* current = wordQueue->front;
    while (current != NULL) {
        if (strchr(guessedLetters, current->data) == NULL) {
            return 0; // Game is not over
        }
        current = current->next;
    }
    return 1; // Game is over
}
// Free memory allocated for the queue
void freeQueue(struct Queue* queue) {
    struct Node* current = queue->front;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
    free(queue);
}
int main() {
    // Initialize a queue for the word
    struct Queue* wordQueue = createQueue();
    // Get the word to guess from the player
    char word[MAX_WORD_LENGTH];
    printf("Enter the word to guess: ");
    scanf("%s", word);
    // Enqueue each character of the word to the queue
    for (int i = 0; word[i] != '\0'; i++) {
        enqueue(wordQueue, word[i]);

    }
```

```
    const int maxAttempts = 6;
    char guessedLetters[maxAttempts + 1]; // +1 for null terminator
    int incorrectAttempts = 0;
    memset(guessedLetters, 0, sizeof(guessedLetters));
    printf("Welcome to Hangman!\n");
    while (1) {
        printf("Guessed letters: %s\n", guessedLetters);
        displayWord(wordQueue, guessedLetters);
        // Ask for a letter
        char guess;
        printf("Enter a letter: ");
        scanf(" %c", &guess);
        // Check if the letter has already been guessed
        if (strchr(guessedLetters, guess) != NULL) {
            printf("You've already guessed that letter.\n");
            continue;
        }
        guessedLetters[strlen(guessedLetters)] = guess;
        // Check if the letter is in the word
        if (strchr(word, guess) == NULL) {
            printf("Incorrect guess!\n");
            incorrectAttempts++;
            // Check if the player has reached the maximum allowed attempts
            if (incorrectAttempts == maxAttempts) {
                printf("Sorry, you've run out of attempts. The word was: %s\n", word);
                break;
            }
        } else {
            printf("Correct guess!\n");
        }
        // Check if the game is over
        if (isGameOver(wordQueue, guessedLetters)) {
            printf("Congratulations! You've guessed the word: %s\n", word);

    break;
```

```
    }
    }
    // Free memory allocated for the word queue
    freeQueue(wordQueue);
    return 0;
}
```

Output

```
Enter the word to guess:

RUTHU YADAV
Welcome to Hangman!
Guessed letters:

_ _ _ _ _
Enter a letter: Incorrect guess!
Guessed letters: Y

_ _ _ _ _
Enter a letter: Incorrect guess!
Guessed letters: YA

_ _ _ _ _
Enter a letter: Incorrect guess!
Guessed letters: YAD

_ _ _ _ _
Enter a letter: You've already guessed that letter.
Guessed letters: YAD

_ _ _ _ _
Enter a letter: Incorrect guess!
Guessed letters: YADV

_ _ _ _ _
Enter a letter:
```

```
_ _ _ _ _
Enter a letter: R
Correct guess!
Guessed letters: YADVR
R _ _ _ _
Enter a letter: U
Correct guess!
Guessed letters: YADVRU
R U _ _ U
Enter a letter: T
Correct guess!
Guessed letters: YADVRUT
R U T _ U
Enter a letter: H
Correct guess!
Congratulations! You've guessed the word: RUTHU
```

Conclusion and Future Work

- In conclusion, the designed Hangman game offers a well-structured and modular solution. Utilizing a variety of data structures and algorithms, including randomization for word selection and efficient user input handling, the game provides an engaging and dynamic experience.
- The organized main loop, coupled with clear functions, ensures a logical flow throughout the gameplay. The design prioritizes memory management, error handling, and scalability, aiming to create a versatile and enjoyable Hangman game.

References

1) *https://www.geeksforgeeks.org/hanger-game-using-data-structures/*
2) *https://www.edureka.co/blog/hanger-game-project-in-java/*