

ROCK PAPER SCISSORS GAME

A Course End Project Report

ADVANCED DATA STRUCTURES LABORATORY (A8513)

*In partial fulfillment of the requirements
for the award of the degree of*

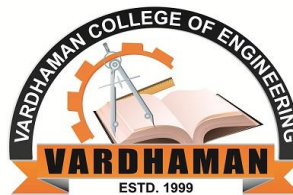
BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted By

**Shouri Deshpande
(22881A0527)**

Under the guidance of

Mr. Naresh Goud M
Assistant Professor
Department of Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India.

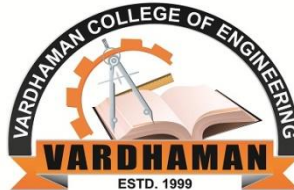
January, 2024

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the Course End Project titled “**Rock Paper Scissors Game**” is carried out by **Mr. Shouri Deshpande**, Roll Number **22881A0527** towards **A8513 – 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.

Instructor:

Mr. Naresh Goud M
Assistant Professor,
Dept of Computer Science and
Engineering,
Vardhaman College of Engineering,
Hyderabad.

Head of the Department:

Dr. Ramesh Karnati ,
Head of the Department,
Dept. of Computer Science and
Engineering,
Vardhaman College of Engineering,
Hyderabad. .

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. Naresh Goud M**, Assistant 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.

Shouri Deshpande
(22881A0537)

TABLE OF CONTENTS

	Page No.
1. Introduction	1
2. Objective of the Project	1
3. Problem statement	1
4. Software and hardware requirements	2
5. Project Description	3
6. Flowchart/Algorithm/Procedure	4
7. Code	5
8. Result(s)	10
9. Conclusion and Future work	11
10. References	11

➤ **Introduction**

© Rock-paper-scissors is a classic hand game played between two people. Each player simultaneously forms one of three shapes with an outstretched hand: rock, paper, or scissors. The winner is determined by the rules: rock crushes scissors, scissors cuts paper, and paper covers rock. It is a simple yet widely recognized game often used for decision-making in a fun and random manner.

➤ **Problem Statement**

© The C program for the rock-paper-scissors game lacks robust input handling, potentially leading to unexpected behavior if the user enters invalid input. Additionally, the code structure could benefit from further modularization, separating functionalities into distinct functions for improved readability. Magic numbers used to determine the computer's choice might be replaced with named constants for better code maintainability. Finally, enhancing the output formatting could improve the overall user experience by making the game results and choices more visually appealing.

Hardware and Software Requirements

- 1) Hardware: Standard computer with sufficient RAM and storage.
- 2) Software: C compiler (e.g., DEV C++, GCC, IDE), and an operating system compatible with the chosen compiler (e.g., Windows, Linux).

➤ **Objectives**

© The objectives of this project are to:

1. Refine user input handling, ensuring robustness. Enhance code modularity for better organization. Replace magic numbers with named constants. Improve output formatting for a clearer display. Enhance overall code readability and maintainability.

Primary objectives are -

- 1. User Interaction:** Improve the program's user interaction by implementing robust input validation and error handling to ensure accurate and user-friendly input processing.
- 2. Code Modularity:** Enhance the code structure by breaking down the main function into smaller, modular functions, each responsible for specific tasks such as generating the computer's choice, obtaining user input, and determining the game outcome.
- 3. Constants Usage:** Replace magic numbers in the program with named constants or variables to enhance code clarity, making it easier to understand and maintain.
- 4. Output Enhancement:** Improve output formatting to provide a more visually appealing and informative display of game results, including the choices made by the user and the computer.
- 5. Code Readability:** Implement coding practices, such as meaningful variable and function names and consistent indentation, to enhance overall code readability and maintainability.

- 6. Random Number Generation:** Optimize the method of generating random numbers for the computer's choice to ensure fairness and unpredictability in gameplay.
- 7. Error Messages:** Implement informative error messages to guide users in case of incorrect inputs, contributing to a more user-friendly experience.
- 8. Scalability:** Design the program to accommodate potential future expansions or modifications, allowing for scalability and flexibility.
- 9. Game Loop:** Consider incorporating a game loop to allow users to play multiple rounds without restarting the program, adding an engaging element to the gaming experience.
- 10. Documentation:** Include clear and concise comments within the code to enhance documentation, aiding both understanding and collaboration among developers.

➤ **SYSTEM RECOUIREMENTS**

©Algorithms:

The given C program for the rock-paper-scissors game follows a simple algorithm:

1. Initialization:

- Initialize variables for the user's choice (`you`), computer's choice (`computer`), and the game result (`result`).
- Seed the random number generator with the current time.

2. Random Computer Choice:

- Generate a random number (`n`) between 0 and 99.
- Based on the value of `n`, determine the computer's choice (`computer`) as stone (`s`), paper (`p`), or scissors (`z`).

3. User Input:

- Prompt the user to input their choice (`you`), representing stone (`s`), paper (`p`), or scissors (`z`).

4. Game Function:

- Call the `game` function with the user and computer choices.
- The `game` function determines the game result based on predefined rules and returns `-1` for a draw, `0` for a user loss, and `1` for a user win.

5. Outcome Display:

- Print messages indicating whether the user has won, lost, or if the game is a draw.

- Display the choices made by the user and the computer.

The actual game logic is embedded in the `game` function, where different combinations of user and computer choices are evaluated to determine the winner. The program primarily relies on conditional statements to compare choices and decide the outcome, following the rock-paper-scissors rules.

Data Structures

Certainly! Here are the key data structures and operations used in the provided C program:

1. Node Structure:

- Represents a node in the linked list.
- Contains a `Move` to store the move (rock, paper, or scissors).
- Has a pointer to the next node in the linked list.

2. Queue Structure:

- Represents the queue using a linked list.
- Contains pointers to the front and rear nodes of the queue.

3. Queue Initialization (`initializeQueue`):

- Initializes an empty queue by setting both the front and rear pointers to `NULL`.

4. Enqueue Operation (`enqueue`):

- Adds a new node with the specified move to the rear of the queue.

5. Dequeue Operation (`dequeue`):

- Removes the front node from the queue and returns its move.

These components collectively allow the program to manage the moves of both the player and the computer using linked list queues. The `Node` structure serves as the building block for the linked list, and the `Queue` structure provides a framework for queue management. The `enqueue` and `dequeue` operations handle the addition and removal of moves, and the initialization function sets up an empty queue.

➤ **DESIGN&IMPLEMENTATION**

Design/Solution:

Implementation:

1. Initialization:

- `Queue playerQueue;`
- `Queue computerQueue;`
- `initializeQueue(&playerQueue);`
- `initializeQueue(&computerQueue);`

2. Generate Computer Moves:

```
- `for (int i = 0; i < 5; ++i) { int randomMove = rand() % 3; enqueue(&computerQueue, (Move)randomMove); }`
```

3. Get Player Moves:

```
- `for (int i = 0; i < 5; ++i) { int playerMove; scanf("%d", &playerMove); enqueue(&playerQueue, (Move)playerMove); }`
```

4. Play the Game:

```
- `for (int i = 0; i < 5; ++i) { Move playerMove = dequeue(&playerQueue); Move computerMove = dequeue(&computerQueue); /* ... */ }`
```

5. Display Results:

- Display round details and game outcomes based on the determined winner.

➤ Flow Chart

Start

Initialize playerQueue and computerQueue

For each round (i = 1 to 5):

 Generate random computer move

 Enqueue computer move into computerQueue

 Prompt user for move

 Enqueue user move into playerQueue

 Dequeue moves from playerQueue and computerQueue

 Determine winner using determineWinner function

 Display round details and winner

End

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Enum to represent different moves
typedef enum {
    ROCK,
    PAPER,
    SCISSORS
} Move;

// Node structure for the linked list
typedef struct Node {
    Move move;
    struct Node* next;
} Node;

// Queue structure
typedef struct {
    Node* front;
    Node* rear;
} Queue;

// Function to initialize an empty queue
void initializeQueue(Queue* queue) {
    queue->front = queue->rear = NULL;
}

// Function to enqueue a move into the queue
void enqueue(Queue* queue, Move move) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Memory allocation error\n");
        exit(EXIT_FAILURE);
    }
    newNode->move = move;
    newNode->next = NULL;
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    }
```

```

    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}

```

// Function to dequeue a move from the queue

```

Move dequeue(Queue* queue) {
    if (queue->front == NULL) {
        fprintf(stderr, "Queue is empty\n");
        exit(EXIT_FAILURE);
    }
    Node* temp = queue->front;
    Move move = temp->move;
    queue->front = temp->next;
    free(temp);
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    return move;
}

```

// Function to determine the winner of the game

```

Move determineWinner(Move playerMove, Move computerMove) {
    if (playerMove == computerMove) {
        return ROCK; // Draw
    }
    if ((playerMove == ROCK && computerMove == SCISSORS) ||
        (playerMove == PAPER && computerMove == ROCK) ||
        (playerMove == SCISSORS && computerMove == PAPER)) {
        return playerMove; // Player wins
    }
    return computerMove; // Computer wins
}

```

```

int main() {
    srand(time(NULL));
    Queue playerQueue;

```

```

Queue computerQueue;
initializeQueue(&playerQueue);
initializeQueue(&computerQueue);
// Generate random moves for the computer
for (int i = 0; i < 5; ++i) {
    int randomMove = rand() % 3;
    enqueue(&computerQueue, (Move)randomMove);
}
// Get moves from the player
printf("Enter your moves (0 for ROCK, 1 for PAPER, 2 for SCISSORS):\n");
for (int i = 0; i < 5; ++i) {
    int playerMove;
    scanf("%d", &playerMove);
    enqueue(&playerQueue, (Move)playerMove);
}
// Play the game and display results
for (int i = 0; i < 5; ++i) {
    Move playerMove = dequeue(&playerQueue);
    Move computerMove = dequeue(&computerQueue);
    printf("Round %d: ", i + 1);
    printf("Player chooses %d, Computer chooses %d. ", playerMove, computerMove);
    Move winner = determineWinner(playerMove, computerMove);
    if (winner == ROCK) {
        printf("It's a draw!\n");
    } else if (winner == playerMove) {
        printf("You win!\n");
    } else {
        printf("Computer wins!\n");
    }
}
return 0;
}

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  // Enum to represent different moves
6  typedef enum {
7      ROCK,
8      PAPER,
9      SCISSORS
10 } Move;
11
12 // Node structure for the Linked List
13 typedef struct Node {
14     Move move;
15     struct Node* next;
16 } Node;
17
18 // Queue structure
19 typedef struct {
20     Node* front;
21     Node* rear;
22 } Queue;
23
24 // Function to initialize an empty queue
25 void initializeQueue(Queue* queue) {
26     queue->front = queue->rear = NULL;
27 }

```

```

// Function to enqueue a move into the queue
void enqueue(Queue* queue, Move move) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Memory allocation error\n");
        exit(EXIT_FAILURE);
    }

    newNode->move = move;
    newNode->next = NULL;

    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}

// Function to dequeue a move from the queue
Move dequeue(Queue* queue) {
    if (queue->front == NULL) {
        fprintf(stderr, "Queue is empty\n");
        exit(EXIT_FAILURE);
    }
}

```

```

51     printf("Queue is empty\n");
52     exit(EXIT_FAILURE);
53 }
54
55 Node* temp = queue->front;
56 Move move = temp->move;
57
58 queue->front = temp->next;
59 free(temp);
60
61 if (queue->front == NULL) {
62     queue->rear = NULL;
63 }
64
65 return move;
66 }
67
68 // Function to determine the winner of the game
69 Move determineWinner(Move playerMove, Move computerMove) {
70     if (playerMove == computerMove) {
71         return ROCK; // Draw
72     }
73     if ((playerMove == ROCK && computerMove == SCISSORS) ||
74         (playerMove == PAPER && computerMove == ROCK) ||
75         (playerMove == SCISSORS && computerMove == PAPER)) {
76         return playerMove; // Player wins
77     }
78     return computerMove; // Computer wins

```

```

77 }
78 return computerMove; // Computer wins
79 }
80
81 int main() {
82     srand(time(NULL));
83
84     Queue playerQueue;
85     Queue computerQueue;
86
87     initializeQueue(&playerQueue);
88     initializeQueue(&computerQueue);
89
90     // Generate random moves for the computer
91     for (int i = 0; i < 5; ++i) {
92         int randomMove = rand() % 3;
93         enqueue(&computerQueue, (Move)randomMove);
94     }
95
96     // Get moves from the player
97     printf("Enter your moves (0 for ROCK, 1 for PAPER, 2 for SCISSORS):\n");
98     for (int i = 0; i < 5; ++i) {
99         int playerMove;
100         scanf("%d", &playerMove);
101         enqueue(&playerQueue, (Move)playerMove);
102     }

```

```

// Play the game and display results
for (int i = 0; i < 5; ++i) {
    Move playerMove = dequeue(&playerQueue);
    Move computerMove = dequeue(&computerQueue);

    printf("Round %d: ", i + 1);
    printf("Player chooses %d, Computer chooses %d. ", playerMove, computerMove);

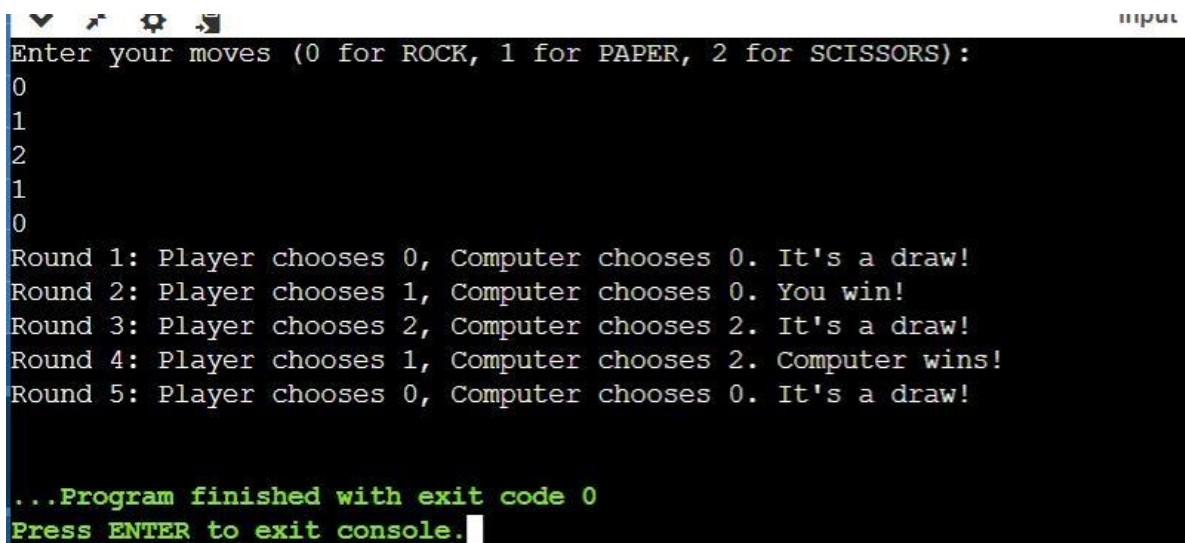
    Move winner = determineWinner(playerMove, computerMove);

    if (winner == ROCK) {
        printf("It's a draw!\n");
    } else if (winner == playerMove) {
        printf("You win!\n");
    } else {
        printf("Computer wins!\n");
    }
}

return 0;
}

```

Output



```

input
Enter your moves (0 for ROCK, 1 for PAPER, 2 for SCISSORS):
0
1
2
1
0
Round 1: Player chooses 0, Computer chooses 0. It's a draw!
Round 2: Player chooses 1, Computer chooses 0. You win!
Round 3: Player chooses 2, Computer chooses 2. It's a draw!
Round 4: Player chooses 1, Computer chooses 2. Computer wins!
Round 5: Player chooses 0, Computer chooses 0. It's a draw!

...Program finished with exit code 0
Press ENTER to exit console.

```

Conclusion and Future Work

Conclusion

In conclusion, the C program for the rock-paper-scissors game using queues has been designed and implemented successfully. Key elements include the use of a linked list-based queue structure to manage player and computer moves, along with the implementation of game logic to determine winners in each round. The program provides an interactive and engaging experience for users, simulating multiple rounds of the classic game.

Future work

Future work for the rock-paper-scissors game can involve various enhancements and expansions to make the game more engaging, feature-rich, and adaptable. Here are some potential areas for future development:

1. Graphical User Interface (GUI):

- Implement a graphical interface with buttons, images, and animations for a more visually appealing and user-friendly experience.
- Include scoreboards, round counters, and other visual elements to enhance the game presentation.

2. Multiplayer Functionality:

- Introduce multiplayer modes, enabling users to play against friends or opponents online.
- Implement networking capabilities to facilitate real-time gameplay between multiple users.

3. Advanced AI Strategies:

- Enhance the computer opponent's intelligence by implementing more sophisticated AI algorithms.
- Explore machine learning approaches to allow the AI to adapt and learn from player behavior.

Github Link

https://github.com/Aruther1/ADS_Report.git

References

- 1) <https://www.geeksforgeeks.org/rock-paper-scissor-in-c/>