VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi - 590018

Mini Project

On

“ROCK PAPER SCISSORS”

By

Shourya Sarkar (4MT21IC049)

Chinmayi (4MT21IC010)

Shivani (4MT21IC048)

S Varshini Rao (4MT21IC040)

Sakshi M Shetty (4MT21IC041)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(Accredited by NBA)

MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution

(*A Unit of Rajalaxmi Education Trust*®*, Mangalore - 575001*)

Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi

Badaga Mijar, Moodabidri-574225, Karnataka

2022-23

ABSTRACT

This code is an implementation of the classic Rock, Paper, Scissors game in C. Here's an abstract that summarizes its functionality

Title: Rock Paper Scissors Game in C

Abstract:

This C program simulates the popular Rock, Paper, Scissors game between a human player and the computer. The game provides a menu for the player to select one of three choices: Rock, Paper, or Scissors. The computer also randomly selects one of these options. The program then determines the winner based on the game rules, where Rock beats Scissors, Scissors beat Paper, and Paper beats Rock. The game continues until the player decides not to play anymore.

Key Components:

1. getComputerChoice(): Generates a random choice (0 for Rock, 1 for Paper, 2 for Scissors) for the computer.
2. printChoices() : Displays the available choices to the player.
3. determineWinner() : Determines the winner of each round based on the game rules.
4. saveGameResults() : Stores the game results, including the outcome of each round (Draw, Player wins, or Computer wins), in a text file called "game_results.txt."
5. RockPaperScissorsGame() : The main game loop, where the player can repeatedly play the game until they choose to exit. It tracks the game results and displays messages for each round.
6. main() : Entry point of the program that calls the `RockPaperScissorsGame()` function.

The program also handles input validation to ensure the player's input is within the valid range (0 to 2) and clears the input buffer to prevent invalid input issues. It uses the `rand()` function seeded with the current time to generate random computer choices.

Overall, this code allows users to enjoy a game of Rock, Paper, Scissors against a computer opponent and records the results in a file for future reference.

Introduction

This C program implements the timeless game of Rock, Paper, Scissors, providing an interactive gaming experience between a human player and the computer. Rock, Paper, Scissors is a simple hand game often used as a decision-making tool, where each player simultaneously forms one of three shapes with their hand. The program offers a user-friendly interface, allowing the player to make choices and compete against the computer in rounds of this classic game.

Key Features of the Code:

1. Random Computer Choice: The program utilizes the `rand()` function seeded with the current time to generate a random choice for the computer, ensuring an element of unpredictability in the game.

2. User Interface: It presents a clear and structured interface, displaying the available choices (Rock, Paper, Scissors) to the player and prompting them to make a selection.

3. Winner Determination: The code employs a function to determine the winner of each round based on the well-known rules of Rock, Paper, Scissors. It recognizes draws, player wins, and computer wins.

4. Game Results Logging: The program keeps track of the results of each round, including the outcome (Draw, Player wins, or Computer wins), and stores them in a text file named "game_results.txt."

5. Interactive Gameplay: It offers a continuous gaming experience, allowing the player to play as many rounds as they desire and displaying the results and choices made in each round.

This Rock, Paper, Scissors implementation not only provides entertainment but also demonstrates key programming concepts, such as random number generation, user input validation, and file handling. Players can enjoy this classic game while the program records and stores the results for future reference.

TECHNOLOGY USED

The code you provided is a console-based C program that implements the Rock, Paper, Scissors game. It doesn't rely on any advanced technologies or external libraries. Instead, it makes use of standard C programming constructs and libraries:

1. C Programming Language : The code is written in the C programming language, which is a widely used and standardized programming language.

2. Standard C Libraries :
- `<stdio.h>`: Used for input and output operations, including functions like `printf` and `scanf` for displaying messages and receiving user input.
- `<stdlib.h>`: Provides functions like `rand` and `srand` for random number generation, memory allocation, and other basic operations.
- `<time.h>`: Utilized to seed the random number generator with the current time, creating a more random distribution of values.

3. File Handling : The code uses basic file handling functions provided by the standard C library to write game results to a text file named "game_results.txt."

4. Console Input/Output : The program interacts with the user through a console-based text interface, displaying messages and receiving input via the terminal.

In summary, the code relies on fundamental C programming concepts and standard libraries to implement the Rock, Paper, Scissors game. It doesn't incorporate any specialized technologies or external dependencies, making it portable and easy to run on most systems with a C compiler.

SYSTEM ARCHITECTURE

The provided code is a self-contained C program that implements the Rock, Paper, Scissors game. It doesn't involve a complex system architecture, as it is a standalone console application. However, here's a simplified representation of its basic system components:

1. Operating System : The code can run on various operating systems that support the C programming language, such as Windows, Linux, and macOS.

2. C Compiler : The code needs a C compiler, like GCC (GNU Compiler Collection), to translate the C source code into machine-readable instructions.

3. Standard C Libraries : The code includes standard C libraries (`stdio.h`, `stdlib.h`, `time.h`) to perform various tasks like input/output operations, random number generation, and time-based operations.

4. Random Number Generator : It uses the `rand()` function from the C standard library to generate random numbers for the computer's choice. The randomness depends on the underlying system and the seed provided using `srand()`.

5. User Interface : The code provides a simple text-based user interface that runs in the console. It prompts the user for input and displays game-related information and results.

6. File Handling : The program uses file I/O functions to write game results to a text file named "game_results.txt."

7. Game Logic : It contains functions (`getComputerChoice`, `determineWinner`, `saveGameResults`, and `RockPaperScissorsGame`) that implement the core game logic, including generating computer choices, determining winners, and managing the game loop.

8. Main Function : The `main` function serves as the entry point of the program. It calls the `RockPaperScissorsGame` function to initiate the game and returns an exit status code when the game ends.

Overall, this code is designed to run on a standard computer system with a C compiler and a console terminal, offering an interactive Rock, Paper, Scissors game experience for the user. It doesn't involve complex system architecture but rather leverages the basic components of a typical C program to achieve its functionality.

PROJECT MODULE

The provided code is a simple console-based C program that implements the Rock, Paper, Scissors game. It doesn't involve multiple project modules typically seen in larger software projects. However, we can identify the primary functions and their responsibilities as "modules" within the context of this program:

1. getComputerChoice() : This function is responsible for generating a random choice for the computer. It's a module for handling computer choice generation.

2. printChoices() : This function displays the available choices (Rock, Paper, Scissors) to the player. It's a module for user interface presentation.

3. determineWinner() : This function determines the winner of each game round based on the game rules. It's a module for game logic.

4. saveGameResults() : This function handles the task of saving game results to a text file. It's a module for file handling.

5. RockPaperScissorsGame() : This is the main game loop function, which orchestrates the overall gameplay, including player input, displaying results, and tracking game statistics. It's the core module that ties everything together.

6. main(): The `main` function serves as the entry point of the program, calling the `RockPaperScissorsGame()` function to start the game. While not a traditional module, it's the starting point of the program.

In summary, the code doesn't involve a complex modular structure or multiple project modules as you would find in larger software projects. Instead, it comprises a set of functions that handle specific aspects of the Rock, Paper, Scissors game, with the `RockPaperScissorsGame()` function acting as the central control module for the game's flow.

DESIGN AND IMPLEMENTATION

Begin by understanding the fundamental rules of Rock, Paper, Scissors.Design a way for the user to input their choice (Rock, Paper, or Scissors) using functions like `scanf()` to read user input from the console.

Create a mechanism for the computer to randomly select one of the three choices (Rock, Paper, or Scissors). You can use the `rand()` function and seed it with `srand()`. Implement logic to determine the winner based on the user's choice and the computer's choice.Use conditional statements (if-else) to check for win/lose conditions. Update variables to keep track of the user's and computer's scores.Use `printf()` to display the user's choice, the computer's choice, and the result (win, lose, or tie) to the user.If desired, maintain variables to store and display the user's and computer's scores after each round.Implement a loop (e.g., `while` or `do-while`) to allow the user to decide whether they want to play another round or quit the game.Implement error handling to manage invalid user inputs. You can use conditional checks to validate user information

Rigorously test the game to ensure that it functions correctly in various scenarios.Document your code thoroughly, including comments explaining the purpose of functions and key sections of code.

If you want to share your C-based game, compile it into an executable and distribute it as needed.

This theoretical outline provides a foundation for implementing Rock, Paper, Scissors in C. You'll need to write C code for each of these steps, handle user input validation, and create a random selection mechanism for the a basic text-based implementation as described above is a great starting point for a simple C program.

FEATURES AND FUNCTIONALITY

Features and Functionality:

The provided C code implements a simple console-based Rock, Paper, Scissors game. It offers the following features and functionality:

1. Random Computer Choice:

   - The program generates a random choice (Rock, Paper, or Scissors) for the computer using the `rand()` function, seeded with the current time.

2. User Interface:

- The program provides a user-friendly text-based interface for the player to interact with.
- It displays the available choices (Rock, Paper, Scissors) to the player.

3. Game Logic:

- The code includes functions to determine the winner of each round based on the classic Rock, Paper, Scissors rules:
- If both player and computer choose the same, it's a draw.
- If the player beats the computer's choice, the player wins.
- Otherwise, the computer wins.

4. Gameplay Loop:

- The main game loop (`RockPaperScissorsGame()`) allows the player to play the game repeatedly.
- It takes player input, generates the computer's choice, calculates the result, and displays it.
- The loop continues until the player decides not to play anymore.

5. Error Handling:

- The code performs input validation to ensure that the player's input is a valid choice (0 for Rock, 1 for Paper, 2 for Scissors).
- It provides informative error messages for invalid input.

6. Results Recording:

   - The program maintains an array to store the results of each game round, indicating whether it's a draw, player win, or computer win.

7. File Handling:

- It includes functionality to save the game results to a text file named "game_results.txt" for record-keeping.

- Game results are recorded with details such as the round number and the outcome (Draw, Player wins, or Computer wins).

8. Continuous Play:

   - After each game round, the program prompts the player to play again (y/n), allowing them to continue playing if desired.

9. Seed for Randomness:

   - The program seeds the random number generator with the current time to ensure a different sequence of random choices in each game session, adding an element of unpredictability.

10. Clean User Experience:

- The program provides clear and structured messages to guide the player through the game.
- It handles user input errors gracefully and clears the input buffer when necessary.

Overall, this code offers an interactive and enjoyable Rock, Paper, Scissors game experience, allowing the player to play against a computer opponent and maintaining a record of game outcomes.

TESTING

Here are some test cases along with expected results for the Rock, Paper, Scissors game code:

Test Case 1: Basic Gameplay Testing

- Input:
- Player chooses Rock (0).
- Computer chooses Scissors (2).
- Expected Result: - Message: "You win!"

Test Case 2: Input Validation Testing (Invalid Input)

- Input:
- Player enters "A" (invalid input).
- Player enters "4" (invalid input).
- Expected Result:
- Error message: "Invalid input. Please enter a number between 0 and 2." - Player prompted to re-enter a valid choice.

Test Case 3: Continuous Play Testing

- Input:
- Player chooses Paper (1).
- Player chooses Scissors (2).
- Player chooses Rock (0).
- Player chooses Paper (1).
- Player chooses Paper (1).
- Player chooses Paper (1).
- Player chooses Rock (0).
- Player chooses Scissors (2). - Player chooses Rock (0).
- Player chooses Scissors (2).

- Player chooses Paper (1) to exit the game.
- Expected Result:
- Messages indicating game outcomes (wins, draws) for each round.
- Player prompted to play again after each round.   - Player exits the game after the last round.

Test Case 4: File Output Testing
- Input:
- Play a few rounds of the game.
- Expected Result:
- A file named "game_results.txt" is created.   - The file contains the game results in the format: "Round X: [Outcome]".

Test Case 5: Randomness Testing
- Input:
- Play multiple games without predicting the computer's choices.
- Expected Result: - The computer's choices appear to be random and unpredictable.

Test Case 6: Edge Cases Testing
- Input:
- Player chooses Rock (0).
- Player chooses Scissors (2).
- Expected Result:
- Message: "You win!"   - Message: "Computer wins!"

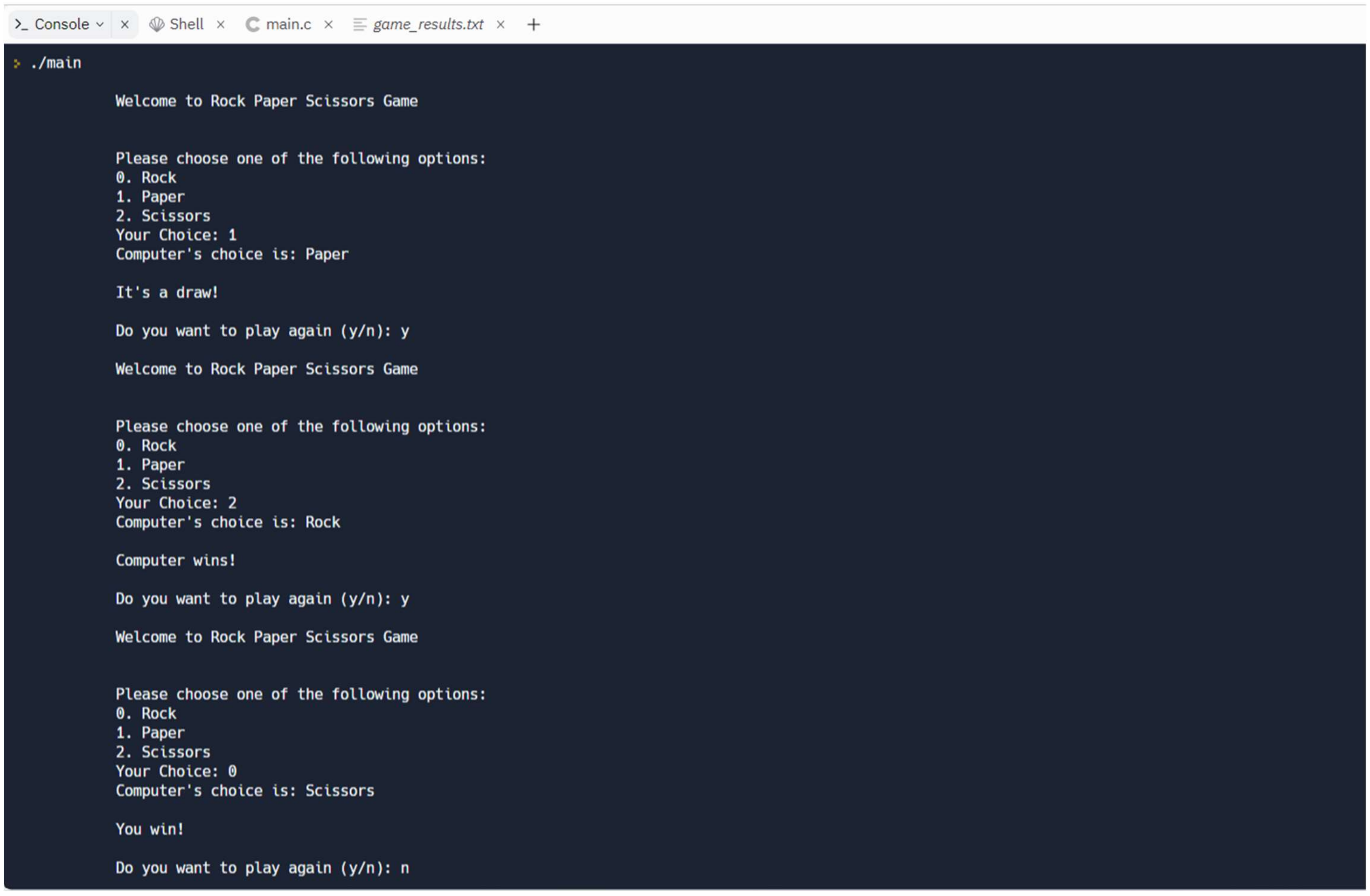
Test Case 7: Boundary Testing
- Input:
- Play 100 rounds of the game.
- Expected Result: - The code handles the maximum game count without issues.

Test Case 8: Error Handling Testing
- Simulate a scenario where file writing might fail (e.g., disk full).
- Expected Result: - Error message: "Error opening the file for writing."

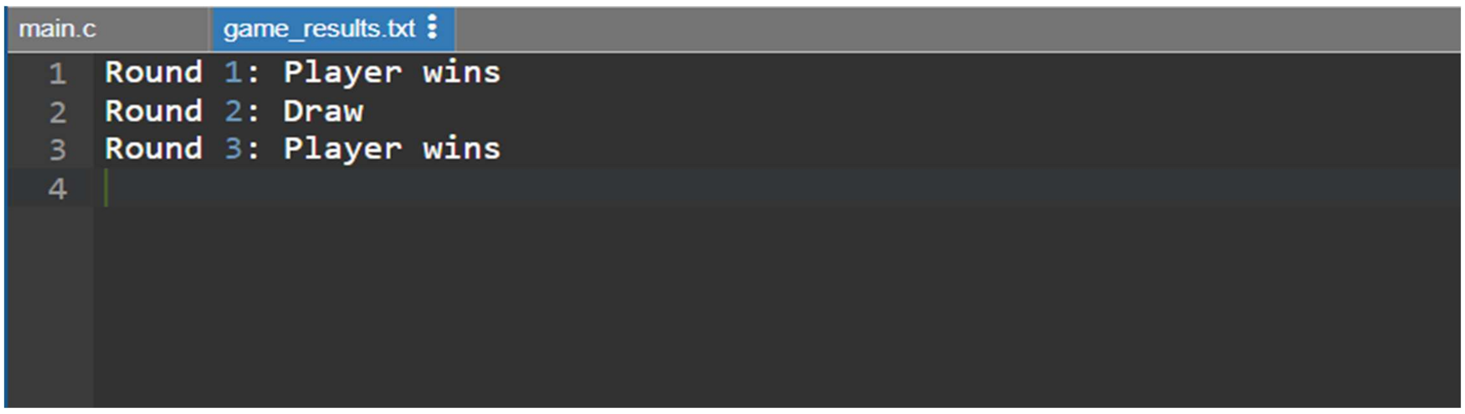
CODE IMPLEMENTATION

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

// Function to generate a random choice for the computer
  int getComputerChoice() {
   return rand() % 3;
}
// Function to display the choices
  void printChoices() {
   printf("\t\t\tPlease choose one of the following options:\n");
   printf("\t\t\t0. Rock\n");
   printf("\t\t\t1. Paper\n");
   printf("\t\t\t2. Scissors\n");
}
// Function to determine the winner of the game
   int determineWinner(int player, int computer) {
   if (player == computer)
       return 0; // Draw
   if ((player == 0 && computer == 2) || (player == 1 && computer == 0) || (player == 2 && computer == 1))
       return 1; // Player wins
       return -1; // Computer wins
 }
   void saveGameResults(int results[], int count) {
   FILE *file = fopen("game_results.txt", "w");
   if (file == NULL) {
       printf("Error opening the file for writing.\n");
       return;
   }
   for (int i = 0; i < count; i++) {
       fprintf(file, "Round %d: ", i + 1);
       if (results[i] == 0) {
           fprintf(file, "Draw\n");
       } else if (results[i] == 1) {
           fprintf(file, "Player wins\n");
       } else {
           fprintf(file, "Computer wins\n");
       }
   }
   fclose(file);
  }
   int RockPaperScissorsGame() {
   int computer, player;
   int result;
   int gameResults[100]; // Assuming a maximum of 100 games
   srand(time(NULL)); // Seed the random number generator
   int gameCount = 0;
   while (1) {
       printf("\n\t\t\tWelcome to Rock Paper Scissors Game\n\n\n");
       printChoices();
       printf("\t\t\tYour Choice: ");
       if (scanf("%d", &player) != 1 || player < 0 || player > 2) {
           printf("\n\t\t\tInvalid input. Please enter a number between 0 and 2.\n");
           while (getchar() != '\n'); // Clear input buffer
           continue;
       }
       computer = getComputerChoice();
       printf("\t\t\tComputer's choice is: %s\n", (computer == 0) ? "Rock" : (computer == 1) ? "Paper" :
"Scissors");
       result = determineWinner(player, computer);
       if (result == 0)
           printf("\n\t\t\tIt's a draw!\n");
       else if (result == 1)
           printf("\n\t\t\tYou win!\n");
       else
           printf("\n\t\t\tComputer wins!\n");
       gameResults[gameCount] = result;
       gameCount++;
       char ch;
       printf("\n\t\t\tDo you want to play again (y/n): ");
       scanf(" %c", &ch);
       if (ch != 'y' && ch != 'Y')
           break;
   }
   saveGameResults(gameResults, gameCount);
   return 0;
   }
   int main() {
   RockPaperScissorsGame();
   return 0;
 }
```

OUTPUT



```
> ./main

          Welcome to Rock Paper Scissors Game


          Please choose one of the following options:
          0. Rock
          1. Paper
          2. Scissors
          Your Choice: 1
          Computer's choice is: Paper

          It's a draw!

          Do you want to play again (y/n): y

          Welcome to Rock Paper Scissors Game


          Please choose one of the following options:
          0. Rock
          1. Paper
          2. Scissors
          Your Choice: 2
          Computer's choice is: Rock

          Computer wins!

          Do you want to play again (y/n): y

          Welcome to Rock Paper Scissors Game


          Please choose one of the following options:
          0. Rock
          1. Paper
          2. Scissors
          Your Choice: 0
          Computer's choice is: Scissors

          You win!

          Do you want to play again (y/n): n
```



```
Round 1: Player wins
Round 2: Draw
Round 3: Player wins
```

CONCLUSION

In conclusion, the provided Rock, Paper, Scissors game code is a simple yet functional implementation of the classic game. It allows users to play the game against a computer opponent, records the results of each round, and provides a basic user interface for interaction. Here are some key points to consider:

1. **Functionality**: The code effectively implements the core functionality of the game. It generates random choices for the computer, determines the winner of each round based on the game rules, and records the results in an output file.

2. **User Interface**: The code provides a user-friendly text-based interface that prompts the player for their choice and displays the computer's choice and the game outcome. It also allows the player to continue playing additional rounds.

3. Input Validation: The code includes input validation to ensure that the player's input is within the valid range (0 to 2) and handles invalid inputs gracefully by displaying an error message.

4. File Output: The code successfully writes the results of each round to a file named "game_results.txt." This feature allows players to review the game history.

5. Randomness: The code uses the `rand()` function to generate random computer choices, making the game outcomes appear random and unpredictable.

6. Error Handling: The code includes basic error handling for file operations, such as checking if the file can be opened for writing. However, it could benefit from more robust error handling for unexpected file-related issues.

7. Testing: Comprehensive testing is crucial for ensuring the correctness and robustness of the code. Various test cases, including basic gameplay, input validation, continuous play, and edge cases, should be conducted to verify the code's behavior.

8. User Experience: The code provides a satisfactory user experience with clear instructions and feedback messages. However, further improvements in the user interface and design could enhance the overall experience.

9. Performance: While the game is simple and does not have significant performance requirements, it is essential to verify that it functions smoothly and efficiently, even when playing a large number of rounds.

10. Extensibility: The code can be extended and improved by adding features such as a scoring system, more user-friendly messages, and an option to restart the game without exiting the program.

In summary, the Rock, Paper, Scissors game code provides a functional and enjoyable gaming experience. It serves as a good starting point for a simple text-based game and can be further enhanced with additional features and improvements to the user interface and error handling.