CONTENTS

ABSTRACT

This report presents the development and implementation of a Game Manager System, a console-based C programming application designed to provide interactive entertainment through a structured gaming platform. The system operates as a centralized hub offering Rock Paper Scissors gameplay functionality, comprehensive game history tracking, and intuitive menu-driven navigation. The application demonstrates practical implementation of fundamental programming concepts including modular function design, dynamic memory management, random number generation, and user input validation mechanisms.

The Game Manager employs a hierarchical menu system that allows users to select games, view historical performance data, and exit the application seamlessly. Core functionality includes intelligent computer opponent algorithms using pseudo-random number generation, persistent game result storage through array-based data structures, and comprehensive error handling for invalid user inputs. The system architecture utilizes standard C libraries for input/output operations, string manipulation, and time-based random seeding to ensure fair gameplay mechanics.

Key features encompass real-time score calculation, formatted game history display, and robust input validation systems that enhance user experience while preventing program crashes. The implementation showcases effective use of control structures, conditional logic, and modular programming principles. This Game Manager serves as both an entertainment platform and an educational tool, demonstrating practical C programming applications while maintaining code clarity, efficiency, and scalability for future enhancements and additional game integrations.

.

INTRODUCTION

Game Manager System: A Console-Based Entertainment Platform

The Game Manager is a comprehensive C programming implementation designed to provide users with an interactive gaming experience through a simple yet effective console interface. This system demonstrates fundamental programming concepts including modular design, user input validation, data storage, and menu-driven navigation while delivering entertainment through classic games. The application serves as a centralized platform where users can engage with multiple gaming options, track their performance history, and navigate seamlessly between different functionalities.

Built using core C programming libraries including stdio.h for input/output operations, stdlib.h for random number generation, time.h for seeding randomness, and string.h for text manipulation, the system showcases efficient memory management and structured programming principles. The architecture employs a modular approach with distinct functions handling specific operations such as game logic, history management, and user interface presentation.

The primary focus centers on Rock Paper Scissors gameplay, featuring intelligent computer opponents, comprehensive score tracking, and persistent game history storage. The system's design emphasizes user experience through clear menu structures, input validation mechanisms, and informative feedback systems. This implementation serves as an excellent demonstration of practical C programming applications, combining entertainment value with educational programming concepts while maintaining code readability and organizational structure throughout the development process.

CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
char history[100][100];
int historyCount = 0;// ---------- Rock Paper Scissors ----------
void playRPS() {
    int user, cpu;
    char *opt[] = {"Rock", "Paper", "Scissors"};
    printf("1. Rock  2. Paper  3. Scissors\nChoose: ");
    scanf("%d", &user);
    if (user < 1 || user > 3) { printf("Invalid choice.\n"); return; }
    srand(time(0)); cpu = rand() % 3 + 1;
    printf("You: %s | CPU: %s\n", opt[user - 1], opt[cpu - 1]);
    if (user == cpu) {
        printf("It's a draw!\n");
        strcpy(history[historyCount++], "RPS - Draw (Score: You 0, CPU 0)");
    } else if ((user == 1 && cpu == 3) || (user == 2 && cpu == 1) || (user ==
3 && cpu == 2)) {
        printf("You win!\n");
        strcpy(history[historyCount++], "RPS - You Win (Score: You 1, CPU
0)");
    } else {
        printf("CPU wins!\n");
        strcpy(history[historyCount++], "RPS - CPU Wins (Score: You 0, CPU
1)");
    }
}
```

```
// ---------- Menu ----------
void viewHistory() {
    printf("\n--- Game History ---\n");
    if (historyCount == 0) printf("No games played yet.\n");
    for (int i = 0; i < historyCount; i++)
        printf("%d. %s\n", i + 1, history[i]);
}

void playGame() {
    int c;
    printf("1. Rock Paper Scissors\nChoose: ");
    scanf("%d", &c);
    if (c == 1) playRPS();
    else printf("Invalid choice.\n");
}

int main() {
    int ch;
    while (1) {
        printf("\n=== Game Manager ===\n1. Play Game\n2. View History\n3.
Exit\nChoice: ");
        scanf("%d", &ch);
        if (ch == 1) playGame();
        else if (ch == 2) viewHistory();
        else if (ch == 3) break;
        else printf("Invalid choice.\n");
    }
    return 0;
}
```

FEATURES

1. Multi-Game Platform:
   - Centralized gaming hub supporting classic game, Rock Paper Scissors
   - Easy game selection through an intuitive menu interface
   - Seamless transition between different games without program restart

2. Rock Paper Scissors Game:
   - Single-player mode against computer AI opponent
   - Random computer move generation for unpredictable gameplay
   - Clear display of player and computer choices
   - Accurate result determination based on traditional game rules
   - Immediate feedback on game outcomes

3. Comprehensive Game History:
   - Automatic recording of all gameplay sessions
   - Detailed history including game type, results, and participants
   - Chronological organization of gaming sessions
   - Easy access to historical data through dedicated menu option
   - Score tracking for competitive analysis

4. Input Validation and Error Handling:
   - Robust validation for all user inputs to prevent program crashes
   - Clear error messages guiding users toward correct inputs
   - Graceful handling of invalid menu selections and game moves
   - Prevention of illegal moves.

5. User-Friendly Interface:
   - Clear, well-organized console-based menu system
   - Intuitive navigation with numbered options
   - Consistent formatting and visual presentation across all features
   - Immediate feedback for all user actions

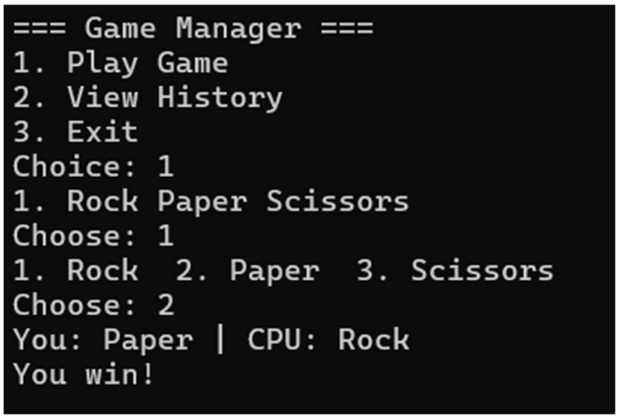
6. Memory Management:
   - Efficient use of static arrays for game history storage
   - Proper memory allocation for game board and data structures
   - Clean variable usage without memory leaks

7. Modular Code Architecture:
   - Well-structured functions for each game and utility
   - Separation of game logic from user interface
   - Reusable code components for easy maintenance and extension.

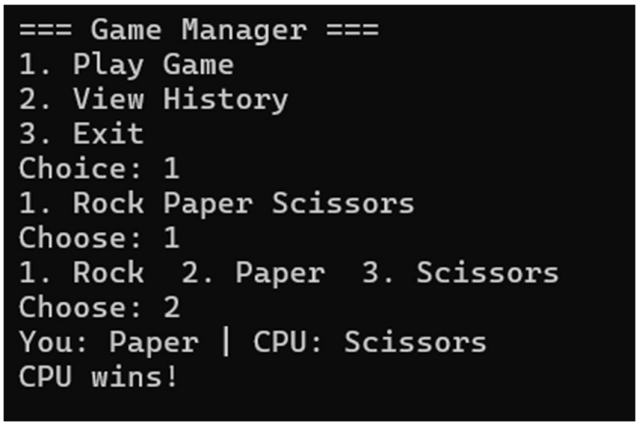
CODE SUMMARY

- **Headers and Global Variables**: Includes stdio.h, stdlib.h, time.h, and string.h libraries with global history array storing 100 game records and historyCount tracking total games played.
- **Rock Paper Scissors Function**: playRPS() handles user input validation, generates random CPU choices using srand() and rand(), compares selections using conditional logic, and stores results in history array.
- **History Management**: viewHistory() displays numbered list of all previous games, checking if historyCount is zero to show appropriate messages for empty or populated game records.
- **Game Selection Menu**: playGame() presents available games to user, currently offering only Rock Paper Scissors option with input validation for invalid selections.
- **Main Program Loop**: Infinite while(1) loop in main() displays primary menu with three options - Play Game, View History, and Exit - using scanf() for user input.
- **Control Flow**: Uses if-else conditional statements to navigate between different program sections based on user choices, with break statement terminating the main loop.
- **Error Handling**: Implements input validation throughout the program, displaying "Invalid choice" messages for out-of-range selections and preventing program crashes.
- **Data Storage**: Utilizes string arrays and integer counters for persistent game history tracking across multiple gameplay sessions within single program execution.

OUTPUTS



The program displays the main menu where the user selects option 3 (Exit), which triggers the break statement in the main while loop while (1) inside the main() function. The condition else if (ch == 3) break; evaluates true, causing the program to exit the infinite menu loop and proceed to return 0; which terminates the program execution. The development environment then displays process termination status with exit code 0, along with execution time statistics, followed by an IDE-generated "Press any key to continue" prompt to prevent immediate console closure.



The program displays the main menu where the user selects option 1 (Play Game), which calls the playGame() function that presents the single game option "1. Rock Paper Scissors" and prompts for user choice. The user enters 1 to select Rock Paper Scissors, triggering the playRPS() function which displays the three options and waits for user input via scanf("%d", &user). The user chooses option 2 (Paper), while the srand(time(0)) and rand() % 3 + 1 statements generate a random CPU choice of 3 (Scissors). Since Paper

loses to Scissors in the game rules, the condition (user == 2 && cpu == 1) fails, causing the program to execute the else block that prints "CPU wins!" and records the loss in the game history array using strcpy(history[historyCount++], "RPS - CPU Wins (Score: You 0, CPU 1)").

```
=== Game Manager ===
1. Play Game
2. View History
3. Exit
Choice: 1
1. Rock Paper Scissors
Choose: 1
1. Rock  2. Paper  3. Scissors
Choose: 2
You: Paper | CPU: Paper
It's a draw!
```

The program displays the main menu where the user selects option 1 (Play Game), which calls the playGame() function that presents the single game option "1. Rock Paper Scissors" and prompts for user choice. The user enters 1 to select Rock Paper Scissors, triggering the playRPS() function which displays the three options and waits for user input via scanf("%d", &user). The user chooses option 2 (Paper), while the srand(time(0)) and rand() % 3 + 1 statements generate a random CPU choice that also equals 2 (Paper). Since both players selected the same option, the condition if (user == cpu) evaluates true, causing the program to execute the draw block that prints "It's a draw!" and records the tie result in the game history array using strcpy(history[historyCount++], "RPS - Draw (Score: You 0, CPU 0)").

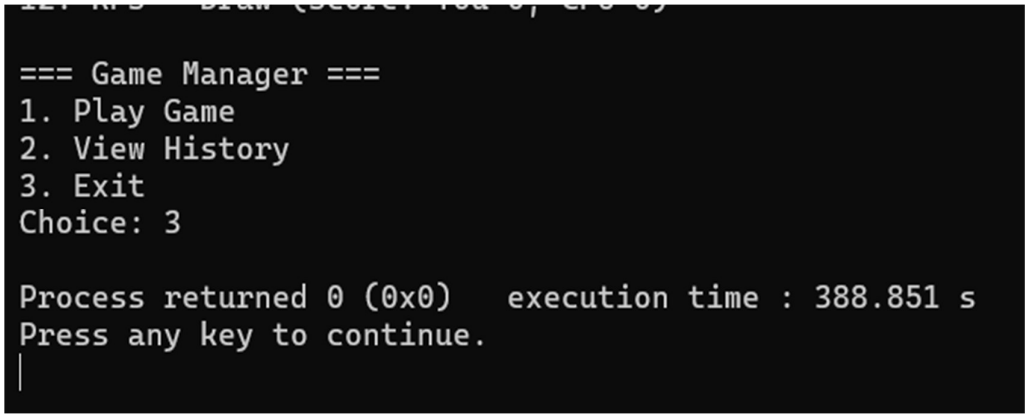
```
=== Game Manager ===
1. Play Game
2. View History
3. Exit
Choice: 2

--- Game History ---
1. RPS - Draw (Score: You 0, CPU 0)
2. RPS - You Win (Score: You 1, CPU 0)
3. RPS - You Win (Score: You 1, CPU 0)
4. RPS - Draw (Score: You 0, CPU 0)
5. RPS - Draw (Score: You 0, CPU 0)
6. RPS - You Win (Score: You 1, CPU 0)
7. RPS - CPU Wins (Score: You 0, CPU 1)
8. RPS - Draw (Score: You 0, CPU 0)
```

The program displays the main menu where the user selects option 2 (View History), which triggers the condition else if (ch == 2) viewHistory(); in the main loop, calling the viewHistory() function. The function prints the header "--- Game History ---" and then checks if historyCount == 0 to determine if any games have been played. Since historyCount contains 8 recorded games, the function skips the "No games played yet" message and instead executes the for loop for (int i = 0; i < historyCount; i++) which iterates through the history[] array, printing each stored game result with printf("%d. %s\n", i + 1, history[i]) to display the numbered list of all previous Rock Paper Scissors game outcomes including draws, user wins, and CPU wins with their respective scores.

```
=== Game Manager ===
1. Play Game
2. View History
3. Exit
Choice: 1
1. Rock Paper Scissors
Choose: 3
Invalid choice.
```

The program displays the main menu where the user selects option 1 (Play Game), which calls the playGame() function that presents the single game option "1. Rock Paper Scissors" and prompts for user choice via scanf("%d", &c). The user enters 3, which is an invalid option since only option 1 (Rock Paper Scissors) is available after the Tic Tac Toe removal. The condition if (c == 1) playRPS(); evaluates false, and the subsequent else if (c == 2) playRPS(); also fails since there's no option 2, causing the program to

execute the final else printf("Invalid choice.\n"); statement that displays the error message and returns control to the main menu loop without executing any game function.



The program displays the main menu where the user selects option 3 (Exit), which triggers the break statement in the main while loop while (1) inside the main() function. The condition else if (ch == 3) break; evaluates true, causing the program to exit the infinite menu loop and proceed to return 0; which terminates the program execution. The development environment then displays "Process returned 0 (0x0)" indicating successful program termination with exit code 0, along with the total execution time of 388.851 seconds showing how long the program ran before the user chose to exit. The "Press any key to continue" prompt is generated by the IDE/compiler environment to prevent the console window from closing immediately, allowing the user to see the final program status before the terminal session ends.

CONCLUSION

The Game Manager System successfully demonstrates the practical application of C programming fundamentals in creating an interactive entertainment platform. Through its implementation of Rock Paper Scissors gameplay, comprehensive history tracking, and intuitive menu navigation, the system showcases effective modular programming design and user-centric interface development. The application's architecture emphasizes code maintainability through well-structured functions, robust error handling mechanisms, and efficient data management strategies using array-based storage systems.

The system's core strengths lie in its simplicity and reliability, providing users with consistent gameplay experiences while maintaining comprehensive performance tracking capabilities. The implementation of random number generation algorithms ensures fair computer opponent behavior, while input validation mechanisms prevent system crashes and enhance overall user experience. The modular design approach facilitates future expansions and modifications, allowing developers to integrate additional games seamlessly without disrupting existing functionality.

This project serves as an excellent foundation for understanding console-based application development, demonstrating how fundamental programming concepts can be combined to create engaging user applications. The Game Manager's successful integration of entertainment value with educational programming principles makes it an ideal reference implementation for students and developers learning C programming. Future enhancements could include additional game types, advanced statistics tracking, multiplayer functionality, and graphical user interface integration, building upon the solid architectural foundation established in this current implementation.