Fundamental Computational Strategies, Implementation Methodology, and Experimental Results

Index

1.0 Introduction

Decision-making systems together with mutual games require computational plans as their underlying structure. Artificial intelligence together with automation and recreational games benefit from these plans which produce logic-based results. The research examines significant computational methods through Python implementation of a Rock Paper Scissors game. This research investigates how program features like randomization and conditional logic along with user actions create an exciting and operable application through detailed study of this case..

2.0 Fundamental Computational Strategies

In the **Rock, Paper, Scissors** game, several vital calculation strategies work together to create a smooth and entertaining experience for the user. These strategies include:

- **Randomization**: To keep the game unpredictable, the computer's move is randomly chosen. This certify that the game remains fair and engaging by remove any patterns or biases.
- **Decision Trees**: The core rules of the game are apply through conditional statements, forming a simple decision tree. This structure helps the program compare the user's and the computer's choices to decide the winner based on the settled rules.
- **User Input Handling**: The program ably processes the player's input by capturing it and converting it into a format that the game can understand. Proper input handling ensures that the user's choices are correctly explain and the game runs smoothly.
- **Iteration & Loops**: The game is designed to allow multiple rounds of play. Using loops, players can continue playing without needing to restart the program each time, improving the user experience and engagement.

3.0 Implementation Methodology

Creating a functional **Rock, Paper, Scissors** game involves several steps to ensure smooth gameplay and interaction. The methodology follows these key phases:

- **User Input Handling**: First, the program captures the player's choice (rock, paper, or scissors). This step ensures the player's input is processed correctly, enabling the game to function as expected.
- **Computer Move Generation**: The computer's move is selected casually to ensure fairness and unpredictability. By using a random selection function, the game avoids any partial outcomes and mimics real-life randomness.
- **Move Comparison**: Once both the player and the computer have made their choices, the program compares the two moves using conditional statements. This step resolve the winner based on the accepted rules of the game.
- **Multiple Rounds**: To improve user engagement, the game offers the option to play multiple rounds. A loop allows the game to continue until the player decides to stop, making it more communal and enjoyable.

- **Python Implementation (Snippet)**

```
import random

def get_computer_choice():
    return random.choice(["rock", "paper", "scissors"])

def determine_winner(user, computer):
    if user == computer:
        return "It's a tie!"
    elif (user == "rock" and computer == "scissors") or \
         (user == "paper" and computer == "rock") or \
         (user == "scissors" and computer == "paper"):
        return "You win!"
    else:
        return "Computer wins!"

def play_game():
    print("Welcome to Rock-Paper-Scissors!")
    choices = ["rock", "paper", "scissors"]

    user_choice = input("Enter rock, paper, or scissors: ").lower()
    while user_choice not in choices:
        user_choice = input("Invalid choice. Enter rock, paper, or scissors:
").lower()

    computer_choice = get_computer_choice()

    print(f"Computer chose: {computer_choice}")
    print(determine_winner(user_choice, computer_choice))

```

```
if _name_ == "_main_":
    play_game()
```

4.0 Experimental Results

Performance Analysis

To evaluate the functionality and accuracy of the game, a series of test cases were conducted. These tests help ensure the program act as expected across a variety of layout:

Test Case	User Choice	Computer Choice	Outcome
1	Rock	Scissors	User Wins
2	Paper	Rock	User Wins
3	Scissors	Paper	User Wins
4	Rock	Rock	Tie
5	Paper	Scissors	Computer Wins

Findings

- The program successfully implements the game logic, consistently producing the correct results based on the players' moves.
- User input is handled efficiently, with built-in error checking to ensure smooth gameplay and prevent invalid inputs.
- Randomization is effectively used to generate the computer's choice, maintaining fairness and unpredictability throughout the game.

5.0 Conclusion

This study emphasizes the vital role that core calculation plan—such as random, logical reasoning, and user input handling—play in the growth of connected games. By using the **Rock, Paper, Scissors** game as a case study, it becomes obvious how these techniques are successfully applied to create pleasant and functional game logic. The findings underscore the significance of these strategies in building more strong and active applications in programming.

6.0 References

- Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education.
- McDowell, G. L. (2018). *Cracking the Coding Interview: 189 Programming Questions and Solutions*. Career cup.
- Swinburne University Learning Resources. (2025). *Computational Thinking and Problem Solving*. Swinburne University of Technology.

Appendix A: Process Documentation

- **Peer Discussions:** Discussed game logic implementation with classmates.
- **Library Research:** Reviewed programming techniques for decision-making algorithms.
- **Coding Process:** Developed, tested, and refined the Python implementation.
- **Search Queries:** "Python random choice function," "Rock Paper Scissors game logic."

Appendix B: Python Source Code

Full Python source code is included as an appendix or in a private GitHub repository.