Report on the Sudoku Game Implementation

Overview:

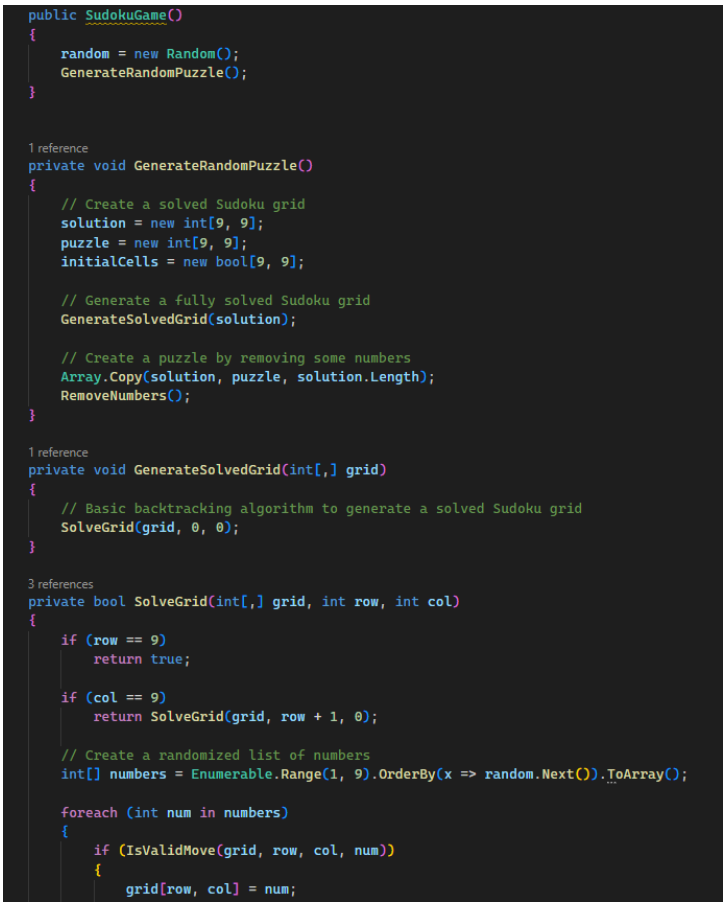
The provided code is a C# implementation of a command-line Sudoku game, designed for an engaging user experience. The core functionality involves generating a solvable 9x9 Sudoku puzzle and allowing the player to solve it by inputting numbers into the grid. The game provides immediate feedback on whether the player's entries are correct or incorrect and congratulates the user upon successfully solving the puzzle.

Key Features:

The game utilizes a backtracking algorithm to generate a fully solved Sudoku grid, which is then modified to create a puzzle by randomly removing a subset of numbers. The puzzle is displayed in a 9x9 grid format, and players are prompted to input values for specific cells. The system ensures that each move adheres to the rules of Sudoku by validating the entered number in terms of row, column, and 3x3 box constraints. Feedback is provided immediately, with correct answers displayed in green and incorrect ones in red, offering a clear indication of progress.

Puzzle Generation and Gameplay:

To create the puzzle, the game first generates a completely solved Sudoku grid using a recursive backtracking algorithm. After the grid is solved, a certain number of cells (between 40 to 60) are randomly cleared to form the puzzle. This randomness ensures that each game is unique and provides an appropriate level of challenge. The player is tasked with filling the empty cells while adhering to Sudoku's rules: no repeating numbers in any row, column, or 3x3 subgrid. The game provides user input prompts for row, column, and value, validating each entry before accepting it.

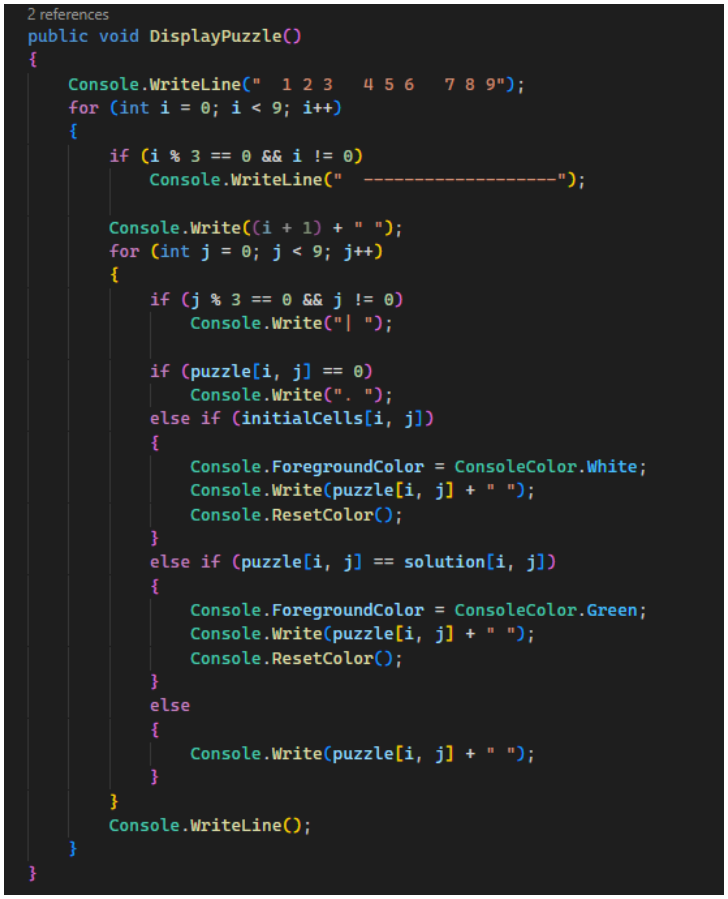


```
public SudokuGame()
{
    random = new Random();
    GenerateRandomPuzzle();
}


1 reference
private void GenerateRandomPuzzle()
{
    // Create a solved Sudoku grid
    solution = new int[9, 9];
    puzzle = new int[9, 9];
    initialCells = new bool[9, 9];

    // Generate a fully solved Sudoku grid
    GenerateSolvedGrid(solution);

    // Create a puzzle by removing some numbers
    Array.Copy(solution, puzzle, solution.Length);
    RemoveNumbers();
}

1 reference
private void GenerateSolvedGrid(int[,] grid)
{
    // Basic backtracking algorithm to generate a solved Sudoku grid
    SolveGrid(grid, 0, 0);
}

3 references
private bool SolveGrid(int[,] grid, int row, int col)
{
    if (row == 9)
        return true;

    if (col == 9)
        return SolveGrid(grid, row + 1, 0);

    // Create a randomized list of numbers
    int[] numbers = Enumerable.Range(1, 9).OrderBy(x => random.Next()).ToArray();

    foreach (int num in numbers)
    {
        if (IsValidMove(grid, row, col, num))
        {
            grid[row, col] = num;
```

```
3 references
private bool SolveGrid(int[,] grid, int row, int col)
{
    if (row == 9)
        return true;

    if (col == 9)
        return SolveGrid(grid, row + 1, 0);

    // Create a randomized list of numbers
    int[] numbers = Enumerable.Range(1, 9).OrderBy(x => random.Next()).ToArray();

    foreach (int num in numbers)
    {
        if (IsValidMove(grid, row, col, num))
        {
            grid[row, col] = num;

            if (SolveGrid(grid, row, col + 1))
                return true;

            grid[row, col] = 0;
        }
    }

    return false;
}
```

User Feedback and Visual Design:

The game makes use of colored text to enhance the user experience. Correct entries are displayed in green, while incorrect ones appear in red. This color-coding helps players easily identify their mistakes and successes. Additionally, initial cells, which are pre-filled and cannot be modified by the player, are displayed in white to differentiate them from editable cells. Upon completing the puzzle correctly, the player receives a random congratulatory message, further adding to the positive reinforcement and satisfaction of solving the puzzle.



```
2 references
public void DisplayPuzzle()
{
    Console.WriteLine("  1 2 3   4 5 6   7 8 9");
    for (int i = 0; i < 9; i++)
    {
        if (i % 3 == 0 && i != 0)
            Console.WriteLine("  ------------------");

        Console.Write((i + 1) + " ");
        for (int j = 0; j < 9; j++)
        {
            if (j % 3 == 0 && j != 0)
                Console.Write("| ");

            if (puzzle[i, j] == 0)
                Console.Write(". ");
            else if (initialCells[i, j])
            {
                Console.ForegroundColor = ConsoleColor.White;
                Console.Write(puzzle[i, j] + " ");
                Console.ResetColor();
            }
            else if (puzzle[i, j] == solution[i, j])
            {
                Console.ForegroundColor = ConsoleColor.Green;
                Console.Write(puzzle[i, j] + " ");
                Console.ResetColor();
            }
            else
            {
                Console.Write(puzzle[i, j] + " ");
            }
        }
        Console.WriteLine();
    }
}
```

```
1 reference
public bool PlaceNumber(int row, int col, int value)
{
    // Check if the cell is initially populated
    if (initialCells[row, col])
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Cannot modify initial cells!");
        Console.ResetColor();
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine(); // Wait for the user to press Enter
        return false;
    }

    // Check if the value is valid
    if (value < 1 || value > 9)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Value must be between 1 and 9!");
        Console.ResetColor();
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine(); // Wait for the user to press Enter
        return false;
    }

    // Check if the move matches the solution
    if (value == solution[row, col])
    {
        puzzle[row, col] = value;
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Correct number placed! 😊");
        Console.ResetColor();
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine(); // Wait for the user to press Enter
        return true;
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Incorrect number! 😞");
        Console.ResetColor();
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine(); // Wait for the user to press Enter
        return false;
    }
}
```

Game Flow and Interaction:

The game operates through a simple menu system that allows the player to start a new game, view instructions, or exit the program. When a new game is started, the puzzle is generated, and the player is prompted to input numbers into the grid. The game loops continuously, checking for valid moves and displaying the updated puzzle until the player either solves the puzzle or chooses to quit. Instructions are available for players unfamiliar with Sudoku, providing a brief overview of the game's objective and basic rules.

```
1 reference
static void ShowInstructions()
{
    Console.Clear();
    Console.WriteLine("===== SUDOKU INSTRUCTIONS =====");
    Console.WriteLine("1. The goal is to fill the 9x9 grid so that each column, row, and 3x3 box contains the digits 1-9.");
    Console.WriteLine("2. Use the menu to start a new game.");
    Console.WriteLine("3. Enter the row (1-9), column (1-9), and value (1-9) you want to place.");
    Console.WriteLine("4. Correct numbers will be displayed in green.");
    Console.WriteLine("5. Press 'q' during the game to return to the main menu.");
    Console.WriteLine("\nPress any key to return to the main menu.");
    Console.ReadKey();
}
}
```

```
static void Main()
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine("===== SUDOKU GAME =====");
        Console.WriteLine("1. New Game");
        Console.WriteLine("2. Instructions");
        Console.WriteLine("3. Exit");
        Console.Write("Choose an option: ");

        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                PlayGame();
                break;
            case "2":
                ShowInstructions();
                break;
            case "3":
                return;
            default:
                Console.WriteLine("Invalid option. Press any key to continue.");
                Console.ReadKey();
                break;
        }
    }
}
```

Potential Enhancements:

While the game is functional and engaging, there are several ways it could be enhanced. First, the difficulty level could be introduced by varying the number of cells removed, allowing players to choose between easy, medium, and hard puzzles. A time tracker could also be added to challenge players to complete the puzzle in less time. Additionally, a hint system could be implemented to offer players suggestions when they are stuck, improving the experience for beginners. Lastly, incorporating multiple puzzle generation algorithms would increase the game's replayability by offering a diverse range of puzzles each time the player starts a new game.

Conclusion:

This Sudoku game implementation is an enjoyable and well-structured experience that captures the essence of the traditional Sudoku puzzle. The backtracking algorithm ensures that each puzzle is solvable, while the interactive features such as real-time feedback and congratulatory messages enhance the player's engagement. The game is easy to play, yet

offers enough challenge to keep players entertained. With some added features, it could provide even greater replayability and variety for users of all skill levels.