OOPS PROJECT ABSTRACT

“SUDOKU GAME IN C++”

NAME: VAISHNAVI SHARMA

ROLL NO.: 23179

BRANCH: CSE

SEMESTER: 4

SUBMITTED TO: Prof. PANKAJ KUMARI

Sudoku in C++

Sudoku is one of the most popular classic games. The playing grid of sudoku consists of the 9*9 squares space. The game initially has 9 squares combining rows and columns. Out of these 9 squares, every square is a separate grid of 3*3 square spaces. Every row and column has 9 squares. It can be either one player or a two-player game where the competitors try to fill the boxes faster than their opponents. The game's primary purpose is to fill boxes in every square with numbers ranging from 1 to 9. The exciting thing about the Sudoku game is that no column, row, or square allows repeating a number.

Moreover, the game initially fills some of the squares with random numbers from 1 to 9. The rest of the boxes or squares are left empty for the player to fill. The number of already filled boxes determines the complexity of the game. Following is a picture of a sample Sudoku game for better visualization.

Requirements and Conditions of the Game

In this Sudoku game, the user wants to develop a game where the player fills empty boxes with remaining numbers according to the game's conditions. Following are the game requirements and resultant conditions for this Sudoku game:

The game board should be of 9*9 squares space.

The game should display the grid with some of the boxes already filled with random numbers ranging from 1 to 9.

The game should allow the player to fill in empty boxes.

The game should not allow the player to repeat any number in any row, column, or square.
If the player fills all the boxes with correct numbers and without any repetition, he wins the game.

C++ Example of Sudoku

Following is a step by step example of sudoku code in c++:

1. Display Game Board

In this C++ example, we pre-defines a board with some initial values and store them in a 2D array. This two-dimensional array is the game board. We can then define the showGame() function to print the sudoku board on the console screen.

2. Checking for Number Repetition

·Checking Columns: Additionally, according to the game's requirements explained above, the game cannot allow the player to repeat any number in any row, column or square. Therefore, the checkColumn() function takes the column number and the input number or value as its arguments and check if the number already
exists in the specified column or not. After it has checked the column for repetitive values, it returns the boolean variables based on its findings.

· Checking Rows: Similarly, the checkRow() function matches the input number with every number in the specified row and returns true or false after verifying its presence.

· Checking Small Squares: Furthermore, another function for checking, the checkBox() function takes the starting number of rows and columns of a specific square, and the user input to match it with the 3 by 3 square to check if the number exists in the entire square or not.

3. Placing Values in Boxes

As all the above conditions need to be false, to place a number in a specific box, the function canEnter() takes the input and calls these above functions inside its body. It checks the output they return. For example, if all of them are false, it returns “true” to confirm that the user can write the input number in that box.

4. Making Final Decision

Moreover, the findBox() function looks for an empty box on the entire game board to see if the user has completed the game or not. After that, it can make a winning decision based on this analysis. In this example, we do not give the input, but the game generates the input itself in the function finalSolution(), the game’s primary function. The function first checks that the board has any empty boxes or not. If it returns true otherwise, it moves on to the “for” loop, which generates the values of numbers from 1 to 9 and calls the above functions to check if it can place these values in the grid or not. Additionally, the findBox() function calls itself recursively until it fills all the positions or else returns false to confirm that no solution exists for that particular given sudoku board.

Conclusion

The users can use the above-explained rules and the coding example to program sudoku for real-world users in which the actual users can input values instead of the computer. This way, the developer can achieve real user interaction instead of automating the whole game. In that scenario, the program can ask for the exact row number, column number, and the input value from the user for every input and use the functions such as checkRow(), checkColumn(), checkBox(), and canEnter() to enter the inputted values. After every input by the user, the program can display the updated grid by using the showGame() function.