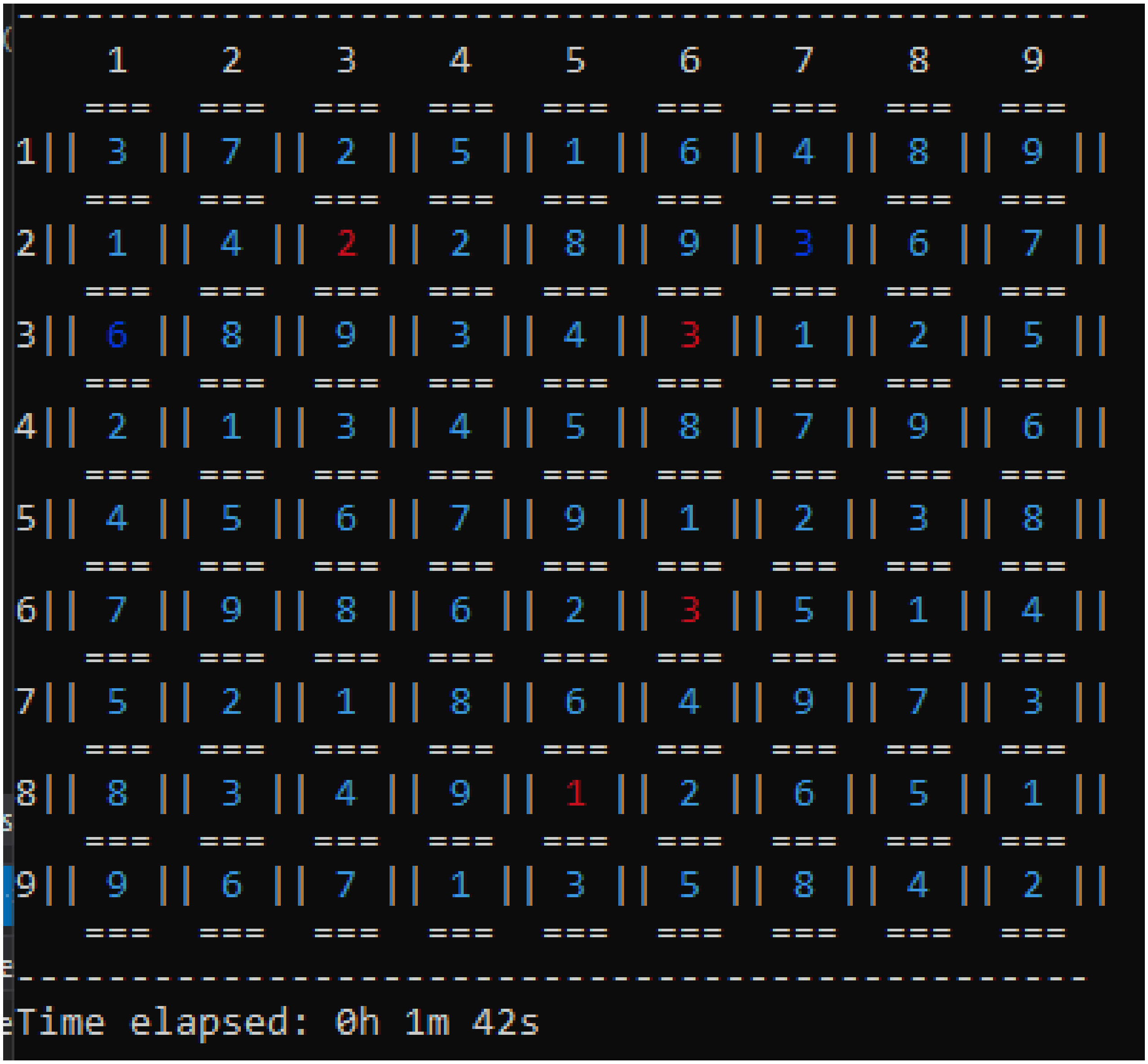
Sudoku

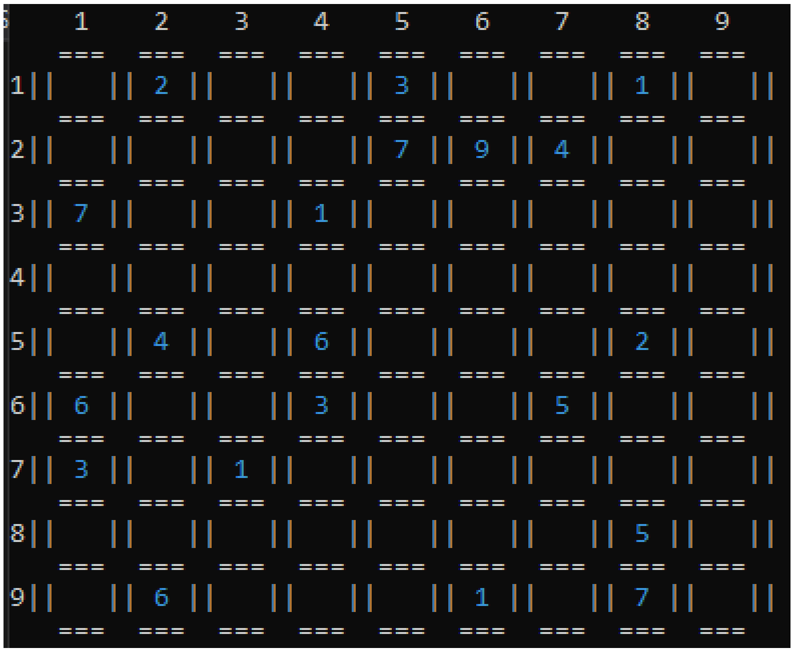
Introduction:



```
-----------------------------------------------
    1    2    3    4    5    6    7    8    9
   ===  ===  ===  ===  ===  ===  ===  ===  ===
1|| 3 || 7 || 2 || 5 || 1 || 6 || 4 || 8 || 9 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
2|| 1 || 4 || 2 || 2 || 8 || 9 || 3 || 6 || 7 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
3|| 6 || 8 || 9 || 3 || 4 || 3 || 1 || 2 || 5 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
4|| 2 || 1 || 3 || 4 || 5 || 8 || 7 || 9 || 6 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
5|| 4 || 5 || 6 || 7 || 9 || 1 || 2 || 3 || 8 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
6|| 7 || 9 || 8 || 6 || 2 || 3 || 5 || 1 || 4 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
7|| 5 || 2 || 1 || 8 || 6 || 4 || 9 || 7 || 3 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
8|| 8 || 3 || 4 || 9 || 1 || 2 || 6 || 5 || 1 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
9|| 9 || 6 || 7 || 1 || 3 || 5 || 8 || 4 || 2 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
-----------------------------------------------
Time elapsed: 0h 1m 42s
```

- **Game:**

Sudoku is a puzzle in which players insert the numbers one to nine into a grid consisting of nine squares subdivided into a further nine smaller squares in such a way that every number appears once in each horizontal line, vertical line, and square.



```
    1    2    3    4    5    6    7    8    9
   ===  ===  ===  ===  ===  ===  ===  ===  ===
1||   || 2 ||   ||   || 3 ||   ||   || 1 ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
2||   ||   ||   ||   || 7 || 9 || 4 ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
3|| 7 ||   ||   || 1 ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
4||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
5||   || 4 ||   || 6 ||   ||   ||   || 2 ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
6|| 6 ||   ||   || 3 ||   ||   || 5 ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
7|| 3 ||   || 1 ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
8||   ||   ||   ||   ||   ||   ||   || 5 ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
9||   || 6 ||   ||   ||   || 1 ||   || 7 ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
```

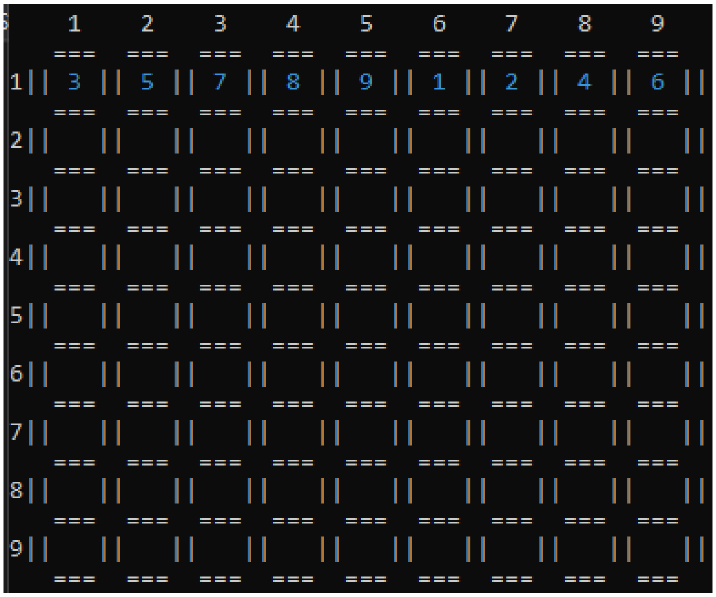
The basic setting for a Sudoku is a 9x9 grid with some of the hints given. Then the user is asked to choose a vertex (x row, y col) of an empty cell and a number (1-9)to go inside the empty cell.

There are nine rows, nine cols, nine 3x3 grids, and 81 cells.

- **Rules:**

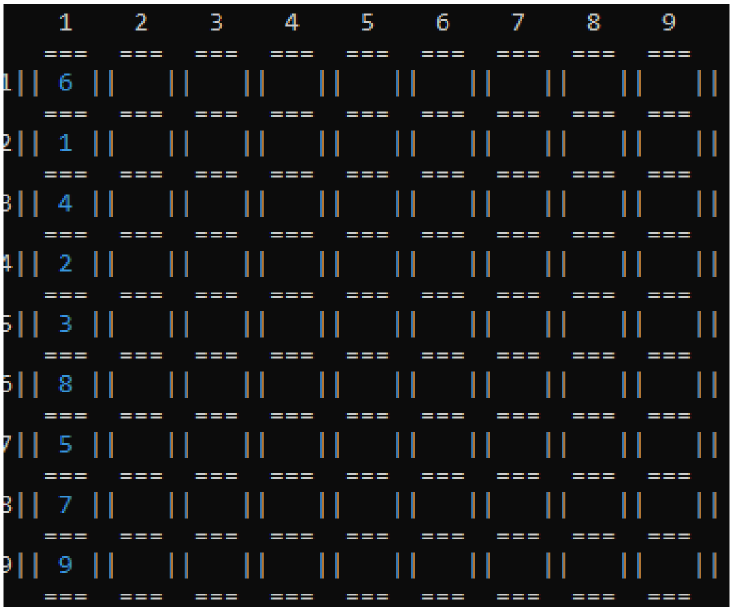
  There are three basic rules in Sudoku.

  1-No repetition in a row.



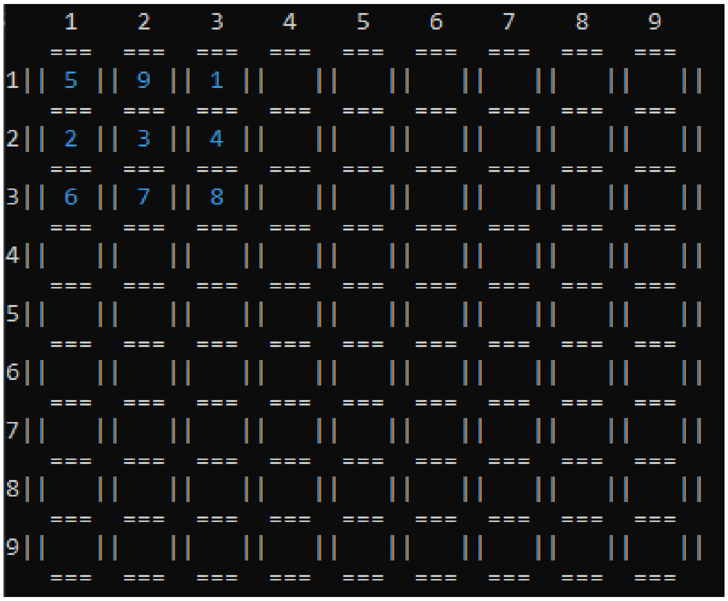
```
    1    2    3    4    5    6    7    8    9
   ===  ===  ===  ===  ===  ===  ===  ===  ===
1|| 3 || 5 || 7 || 8 || 9 || 1 || 2 || 4 || 6 ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
2||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
3||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
4||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
5||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
6||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
7||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
8||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
9||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
```

  2-no repetition in a column.



```
    1    2    3    4    5    6    7    8    9
   ===  ===  ===  ===  ===  ===  ===  ===  ===
1|| 6 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
2|| 1 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
3|| 4 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
4|| 2 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
5|| 3 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
6|| 8 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
7|| 5 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
8|| 7 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
9|| 9 ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
```

3-no repetition in a 3x3 block.



```
    1    2    3    4    5    6    7    8    9
   ===  ===  ===  ===  ===  ===  ===  ===  ===
1|| 5 || 9 || 1 ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
2|| 2 || 3 || 4 ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
3|| 6 || 7 || 8 ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
4||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
5||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
6||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
7||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
8||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
9||   ||   ||   ||   ||   ||   ||   ||   ||   ||
   ===  ===  ===  ===  ===  ===  ===  ===  ===
```

- **Features:**

1-colored representation

2-save/load option

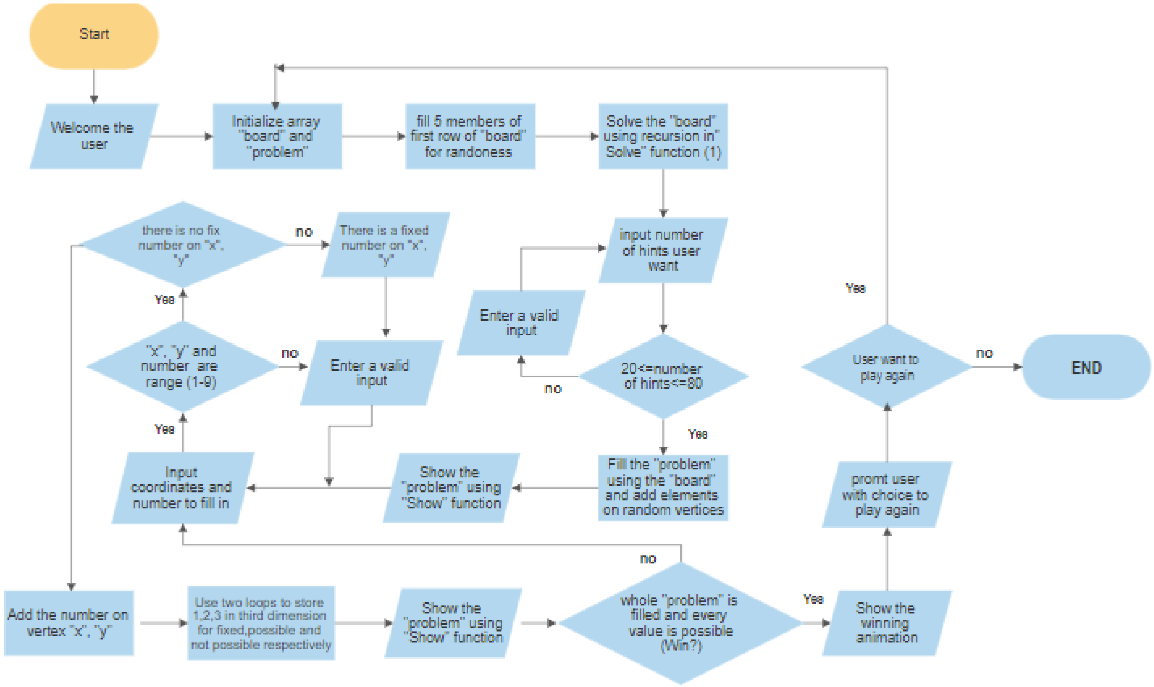
3-User friendly

4-Time and #of steps

5-Winning Animation

Description:

- **Planning and Flow chart:**


Start
Welcome the user
Initialize array "board" and "problem"
fill 5 members of first row of "board" for randoness
Solve the "board" using recursion in" Solve" function (1)
there is no fix number on "x", "y"
no
There is a fixed number on "x", "y"
input number of hints user want
Yes
Enter a valid input
"x", "y" and number are range (1-9)
no
Enter a valid input
20<=number of hints<=80
no
User want to play again
no
END
Yes
Yes
Input coordinates and number to fill in
Show the "problem" using "Show" function
Fill the "problem" using the "board" and add elements on random vertices
promt user with choice to play again
no
Add the number on vertex "x", "y"
Use two loops to store 1,2,3 in third dimension for fixed,possible and not possible respectively
Show the "problem" using "Show" function
whole "problem" is filled and every value is possible (Win?)
Yes
Show the winning animation

Following are the steps we would use to solve make the game.

1- Define two arrays “board” and “problem”. “board” for making and storing the solution in and “problem” to let the user solve.
2- Solve the “board” array with a random solution using the recursive function “solve ()”.
   (Step 2 is ignored in case of loading from a saved game)
3- Fill up the “problem” board with some of the hints (specified by the user). We will fill it with “board”

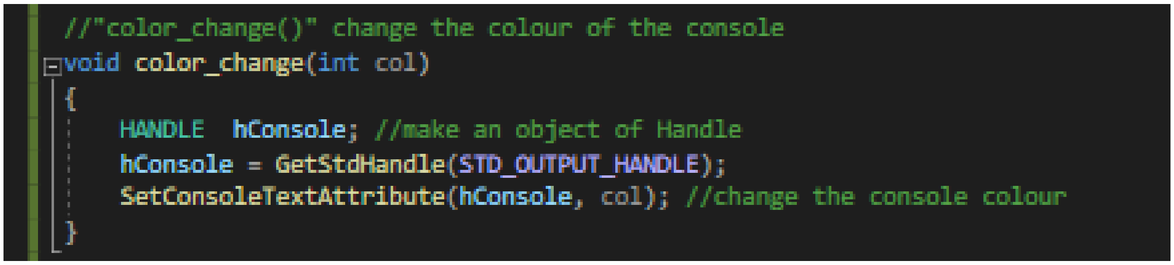
array. Or we will load the “problem” from the saved file.

4- Ask the user to enter numbers in the “problem” array.

5- When the game is over, show an animation to the user and ask him if he/she wants to play again.

● **Functions:**

A. **Color_change() :-**

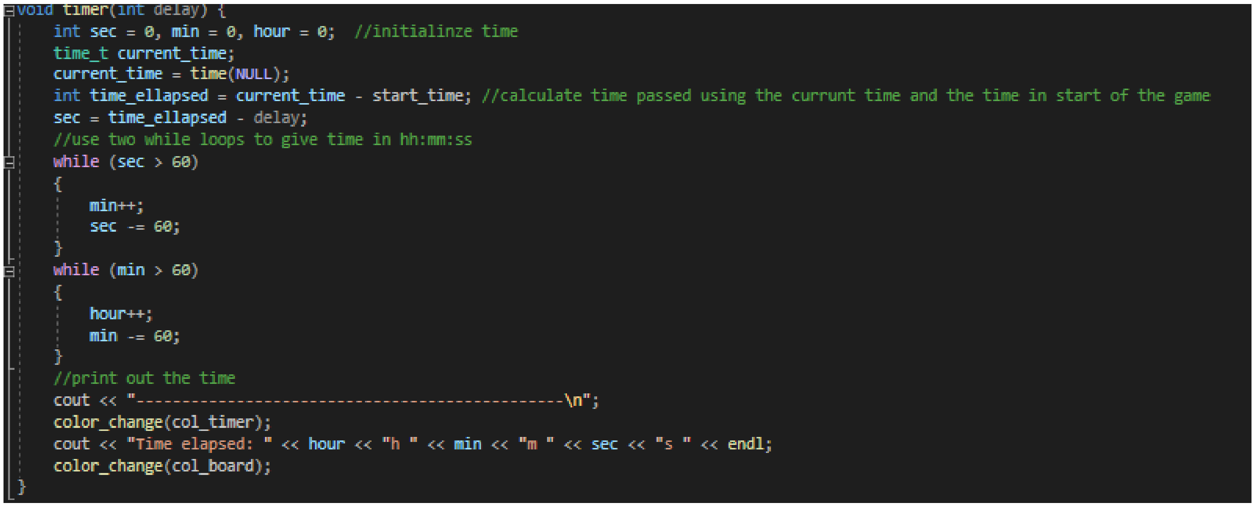


```
//"color_change()" change the colour of the console
void color_change(int col)
{
    HANDLE  hConsole; //make an object of Handle
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, col); //change the console colour
}
```

“Color_change ()” function takes an int input as a parameter and turns the console text color into a different color. The color is changed according to the given color list. It is just for better understanding and to make the code user friendly.

```
0 = Black
1 = Blue
2 = Green
3 = Aqua
4 = Red
5 = Purple
6 = Yellow
7 = White
8 = Gray
9 = Light Blue
```

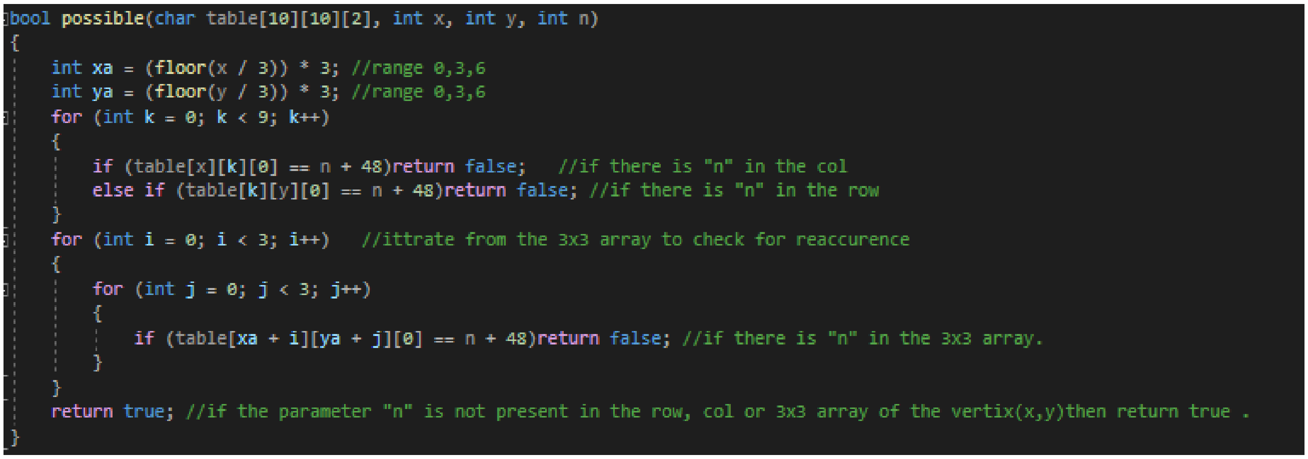
B. Timer() :-



```
void timer(int delay) {
    int sec = 0, min = 0, hour = 0;  //initialinze time
    time_t current_time;
    current_time = time(NULL);
    int time_ellapsed = current_time - start_time; //calculate time passed using the curunt time and the time in start of the game
    sec = time_ellapsed - delay;
    //use two while loops to give time in hh:mm:ss
    while (sec > 60)
    {
        min++;
        sec -= 60;
    }
    while (min > 60)
    {
        hour++;
        min -= 60;
    }
    //print out the time
    cout << "----------------------------------------------\n";
    color_change(col_timer);
    cout << "Time elapsed: " << hour << "h " << min << "m " << sec << "s " << endl;
    color_change(col_board);
}
```

"Timer ()" takes delay as a parameter and calculates the total time passed since the game is started. Then it removes the delay from the time and prints it out in a neat output.

C. Possible():-

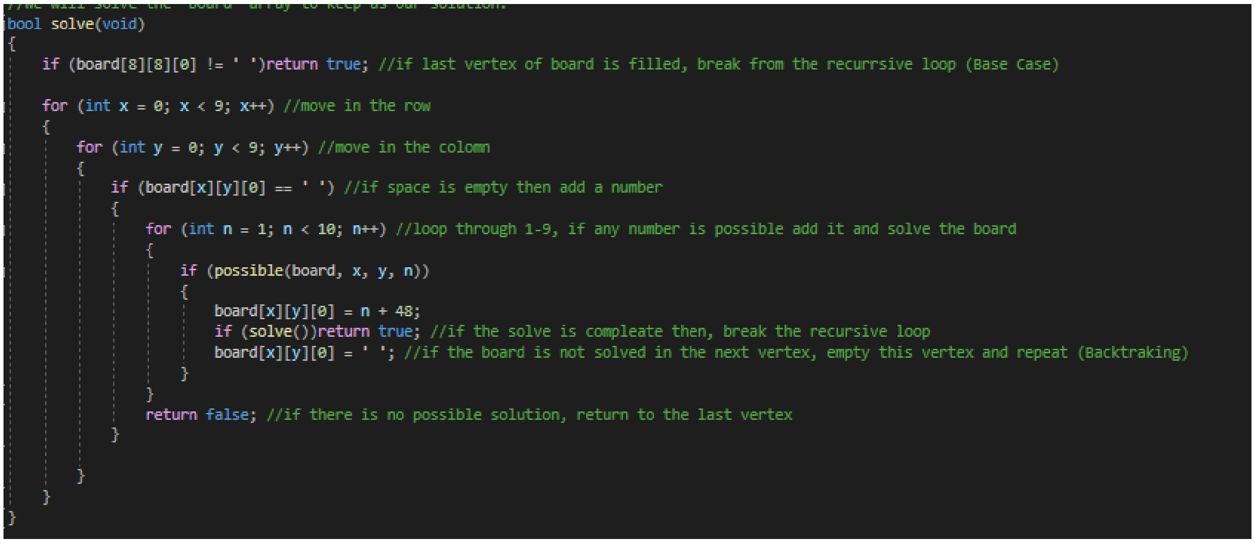


```
bool possible(char table[10][10][2], int x, int y, int n)
{
    int xa = (floor(x / 3)) * 3; //range 0,3,6
    int ya = (floor(y / 3)) * 3; //range 0,3,6
    for (int k = 0; k < 9; k++)
    {
        if (table[x][k][0] == n + 48)return false;   //if there is "n" in the col
        else if (table[k][y][0] == n + 48)return false; //if there is "n" in the row
    }
    for (int i = 0; i < 3; i++)   //ittrate from the 3x3 array to check for reaccurence
    {
        for (int j = 0; j < 3; j++)
        {
            if (table[xa + i][ya + j][0] == n + 48)return false; //if there is "n" in the 3x3 array.
        }
    }
    return true; //if the parameter "n" is not present in the row, col or 3x3 array of the vertix(x,y)then return true .
}
```

“Possible ()” function takes an array and vertex (x, y) and a number “n” as the parameters and returns a Boolean. If the number “n” is right to go in row “x” and col “y” in the array “table” then it returns true, otherwise it returns false.

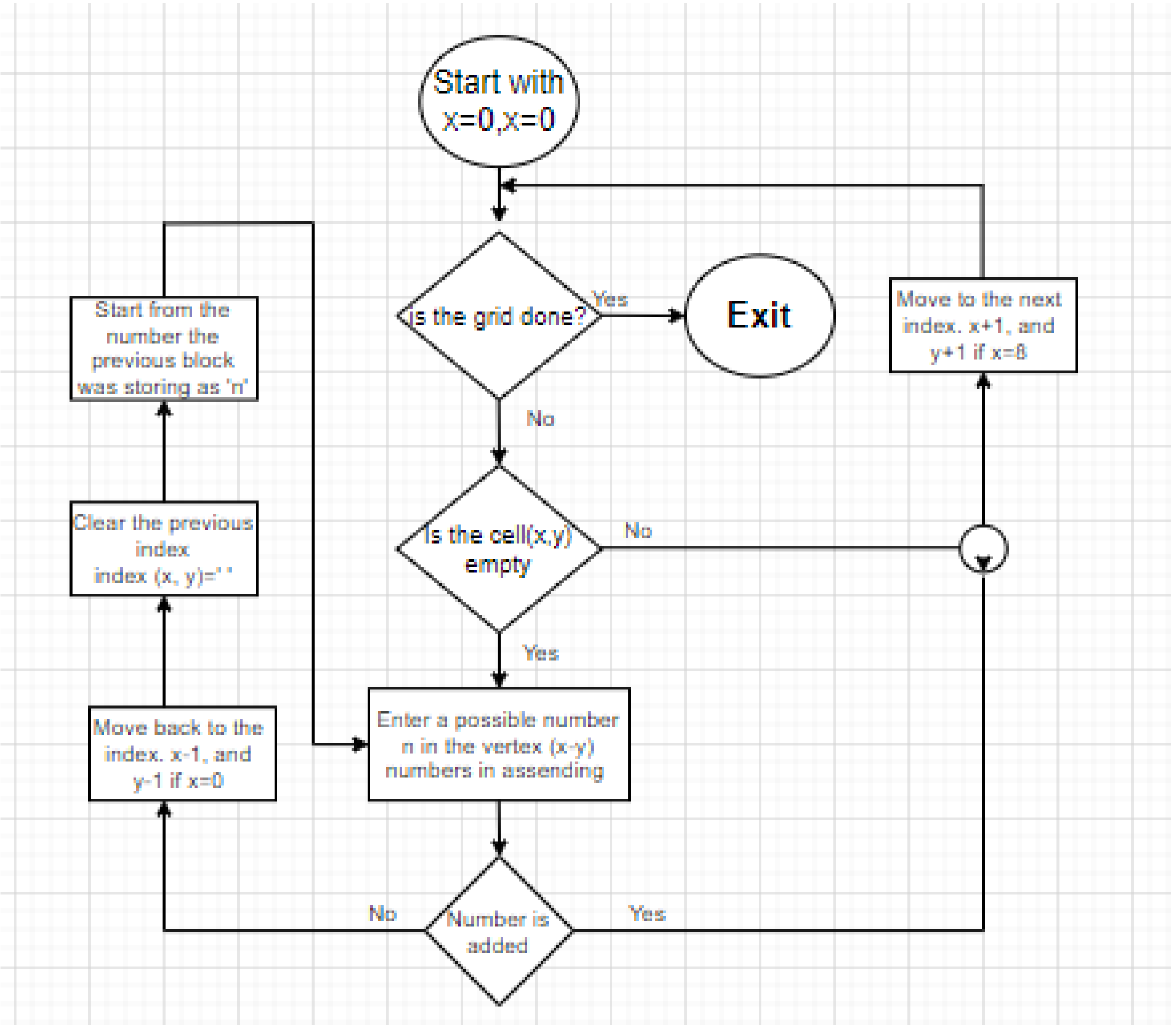
It checks the possibility of using iterative loops of the number according to the three rules discussed above.

D.Solve():-

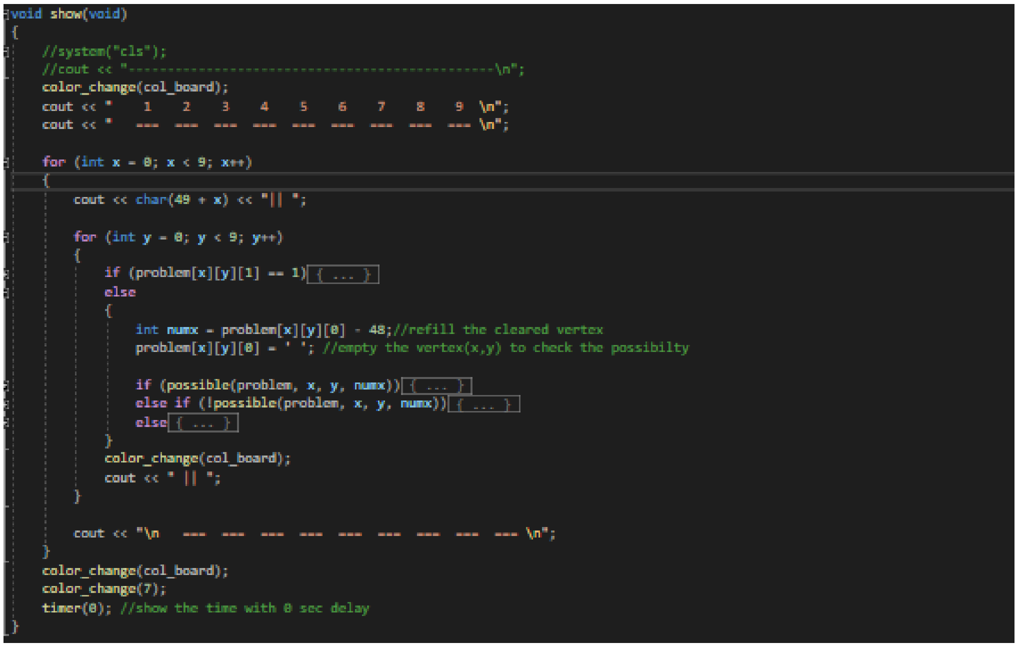


```
//We will solve the "board" array to keep as our solution.
bool solve(void)
{
    if (board[8][8][0] != ' ')return true; //if last vertex of board is filled, break from the recurrsive loop (Base Case)

    for (int x = 0; x < 9; x++) //move in the row
    {
        for (int y = 0; y < 9; y++) //move in the colomn
        {
            if (board[x][y][0] == ' ') //if space is empty then add a number
            {
                for (int n = 1; n < 10; n++) //loop through 1-9, if any number is possible add it and solve the board
                {
                    if (possible(board, x, y, n))
                    {
                        board[x][y][0] = n + 48;
                        if (solve())return true; //if the solve is compleate then, break the recursive loop
                        board[x][y][0] = ' '; //if the board is not solved in the next vertex, empty this vertex and repeat (Backtraking)
                    }
                }
                return false; //if there is no possible solution, return to the last vertex
            }

        }
    }
}
```

“Solve ()” function uses backtracking and Recursion to solve the first row filled board made by the ***“set_board()”*** function.


Start with x=0,x=0
Is the grid done?
Yes
Exit
No
Move to the next index. x+1, and y+1 if x=8
Start from the number the previous block was storing as 'n'
Is the cell(x,y) empty
No
Clear the previous index index (x, y)=' '
Yes
Move back to the index. x-1, and y-1 if x=0
Enter a possible number n in the vertex (x-y) numbers in assending
No
Number is added
Yes

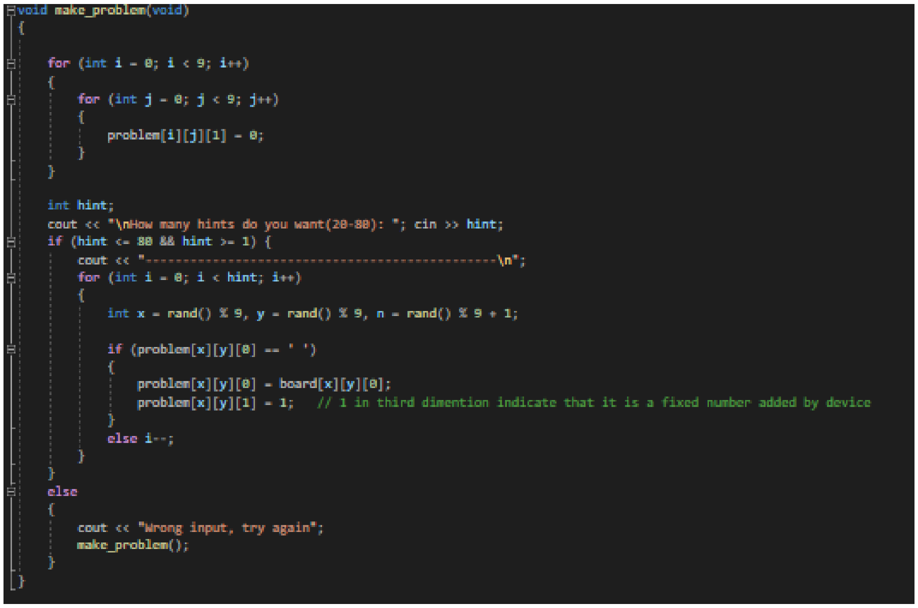
E. Show() :-



```
void show(void)
{
    //system("cls");
    //cout << "---------------------------------------------\n";
    color_change(col_board);
    cout << "    1    2    3    4    5    6    7    8    9  \n";
    cout << "   ---  ---  ---  ---  ---  ---  ---  ---  --- \n";

    for (int x = 0; x < 9; x++)
    {
        cout << char(49 + x) << "|| ";

        for (int y = 0; y < 9; y++)
        {
            if (problem[x][y][1] == 1){ ... }
            else
            {
                int numx = problem[x][y][0] - 48;//refill the cleared vertex
                problem[x][y][0] = ' '; //empty the vertex(x,y) to check the possibilty

                if (possible(problem, x, y, numx)){ ... }
                else if (!possible(problem, x, y, numx)){ ... }
                else{ ... }
            }
            color_change(col_board);
            cout << " || ";
        }

        cout << "\n   ---  ---  ---  ---  ---  ---  ---  ---  --- \n";
    }
    color_change(col_board);
    color_change(7);
    timer(0); //show the time with 0 sec delay
}
```

"Show ()" function uses iterative functions to print out the whole 9x9 grid in a user frame.

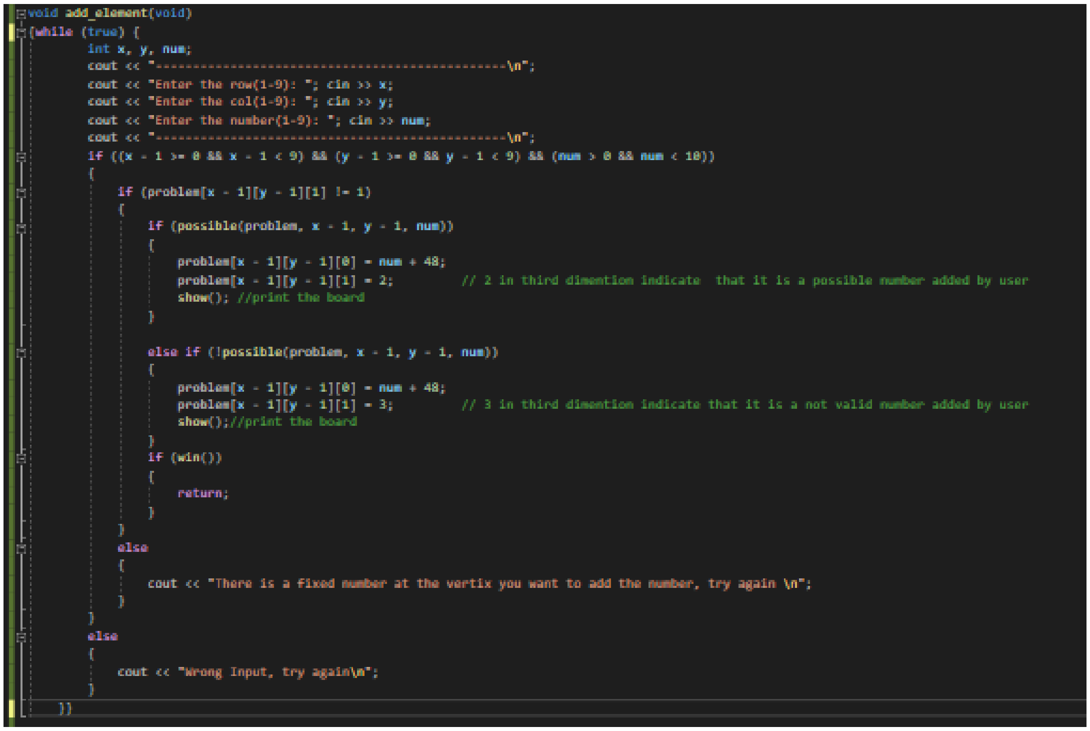
F. <u>Make_problem():-</u>



```
void make_problem(void)
{

    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            problem[i][j][1] = 0;
        }
    }

    int hint;
    cout << "\nHow many hints do you want(20-80): "; cin >> hint;
    if (hint <= 80 && hint >= 1) {
        cout << "---------------------------------------------\n";
        for (int i = 0; i < hint; i++)
        {
            int x = rand() % 9, y = rand() % 9, n = rand() % 9 + 1;

            if (problem[x][y][0] == ' ')
            {
                problem[x][y][0] = board[x][y][0];
                problem[x][y][1] = 1;   // 1 in third dimention indicate that it is a fixed number added by device
            }
            else i--;
        }
    }
    else
    {
        cout << "Wrong input, try again";
        make_problem();
    }
}
```

"Make_probem()" asks the user about the number of hints he/she wants and then chooses that number of hints from the “board” array and fills the “problem” array using that.

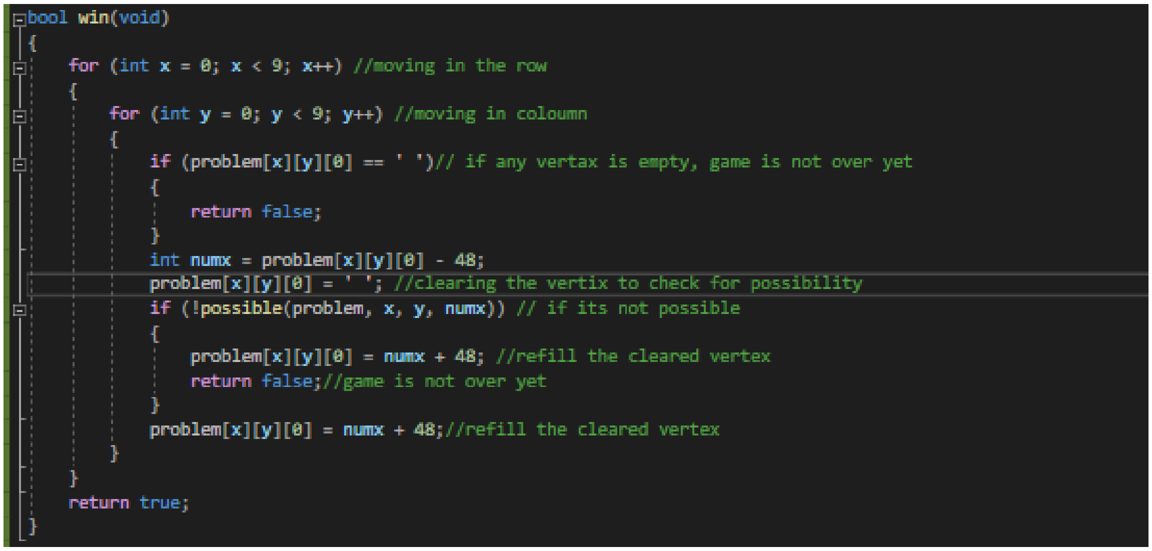
G. <u>add element():-</u>



```cpp
void add_element(void)
{while (true) {
        int x, y, num;
        cout << "-----------------------------------------------\n";
        cout << "Enter the row(1-9): "; cin >> x;
        cout << "Enter the col(1-9): "; cin >> y;
        cout << "Enter the number(1-9): "; cin >> num;
        cout << "-----------------------------------------------\n";
        if ((x - 1 >= 0 && x - 1 < 9) && (y - 1 >= 0 && y - 1 < 9) && (num > 0 && num < 10))
        {
            if (problem[x - 1][y - 1][1] != 1)
            {
                if (possible(problem, x - 1, y - 1, num))
                {
                    problem[x - 1][y - 1][0] = num + 48;
                    problem[x - 1][y - 1][1] = 2;        // 2 in third dimention indicate  that it is a possible number added by user
                    show(); //print the board
                }

                else if (!possible(problem, x - 1, y - 1, num))
                {
                    problem[x - 1][y - 1][0] = num + 48;
                    problem[x - 1][y - 1][1] = 3;        // 3 in third dimention indicate that it is a not valid number added by user
                    show();//print the board
                }
                if (win())
                {
                    return;
                }
            }
            else
            {
                cout << "There is a fixed number at the vertix you want to add the number, try again \n";
            }
        }
        else
        {
            cout << "Wrong Input, try again\n";
        }
    }}
```

“Add_element ()” function is to let the user enter a number (1-9) in the grid. And after entering the number into the user-specified vertex, it calls the ***“show”*** function.

H.<u>Win():-</u>



```
bool win(void)
{
    for (int x = 0; x < 9; x++) //moving in the row
    {
        for (int y = 0; y < 9; y++) //moving in coloumn
        {
            if (problem[x][y][0] == ' ')// if any vertax is empty, game is not over yet
            {
                return false;
            }
            int numx = problem[x][y][0] - 48;
            problem[x][y][0] = ' '; //clearing the vertix to check for possibility
            if (!possible(problem, x, y, numx)) // if its not possible
            {
                problem[x][y][0] = numx + 48; //refill the cleared vertex
                return false;//game is not over yet
            }
            problem[x][y][0] = numx + 48;//refill the cleared vertex
        }
    }
    return true;
}
```

“***Win()***” function checks the whole board and returns a boolean which is true if the game is over.

1) All spaces are filled.
2) All the numbers in the grain are possible

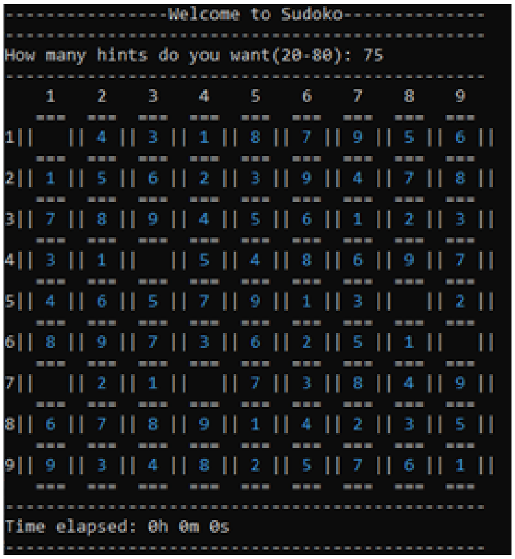
<u>Limitations:</u>

-User can Not Enter an input with different data type then the specified one, it causes a runtime error.

Results:

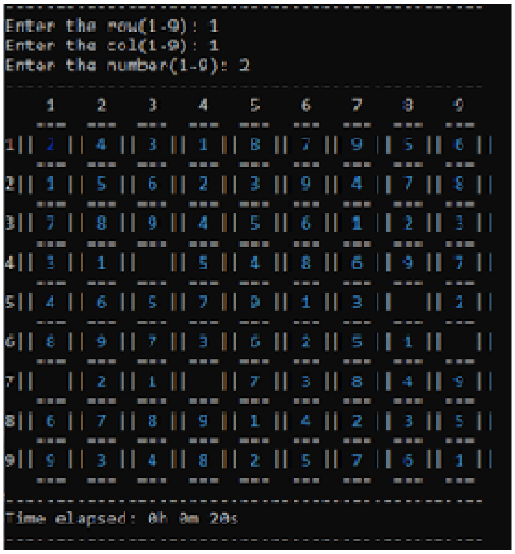
• Console Output:

Starting



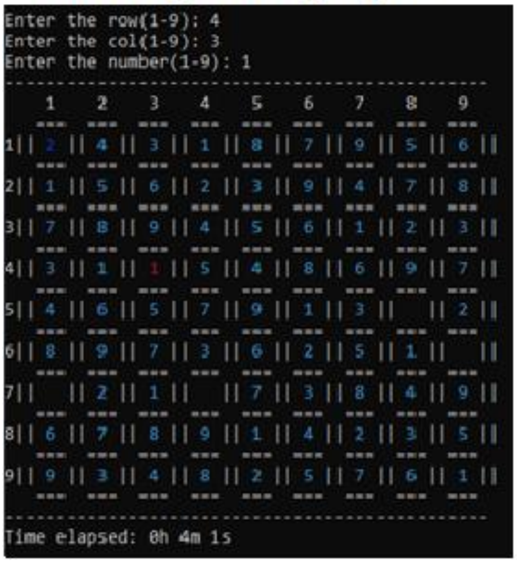
```
----------------Welcome to Sudoko----------------
-------------------------------------------------
How many hints do you want(20-80): 75
-------------------------------------------------
    1    2    3    4    5    6    7    8    9
   ---  ---  ---  ---  ---  ---  ---  ---  ---
1||   || 4 || 3 || 1 || 8 || 7 || 9 || 5 || 6 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
2|| 1 || 5 || 6 || 2 || 3 || 9 || 4 || 7 || 8 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
3|| 7 || 8 || 9 || 4 || 5 || 6 || 1 || 2 || 3 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
4|| 3 || 1 ||   || 5 || 4 || 8 || 6 || 9 || 7 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
5|| 4 || 6 || 5 || 7 || 9 || 1 || 3 ||   || 2 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
6|| 8 || 9 || 7 || 3 || 6 || 2 || 5 || 1 ||   ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
7||   || 2 || 1 ||   || 7 || 3 || 8 || 4 || 9 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
8|| 6 || 7 || 8 || 9 || 1 || 4 || 2 || 3 || 5 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
9|| 9 || 3 || 4 || 8 || 2 || 5 || 7 || 6 || 1 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
-------------------------------------------------
Time elapsed: 0h 0m 0s
-------------------------------------------------
```

Inputting a possible number



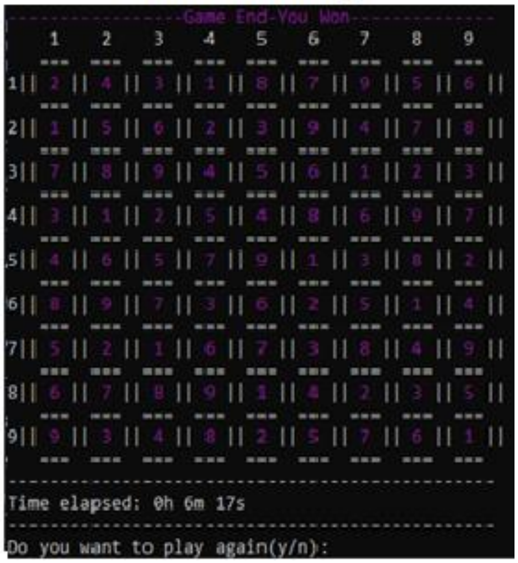
```
-------------------------------------------------
Enter the row(1-9): 1
Enter the col(1-9): 1
Enter the number(1-9): 2
-------------------------------------------------
    1    2    3    4    5    6    7    8    9
   ---  ---  ---  ---  ---  ---  ---  ---  ---
1|| 2 || 4 || 3 || 1 || 8 || 7 || 9 || 5 || 6 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
2|| 1 || 5 || 6 || 2 || 3 || 9 || 4 || 7 || 8 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
3|| 7 || 8 || 9 || 4 || 5 || 6 || 1 || 2 || 3 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
4|| 3 || 1 ||   || 5 || 4 || 8 || 6 || 9 || 7 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
5|| 4 || 6 || 5 || 7 || 9 || 1 || 3 ||   || 2 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
6|| 8 || 9 || 7 || 3 || 6 || 2 || 5 || 1 ||   ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
7||   || 2 || 1 ||   || 7 || 3 || 8 || 4 || 9 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
8|| 6 || 7 || 8 || 9 || 1 || 4 || 2 || 3 || 5 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
9|| 9 || 3 || 4 || 8 || 2 || 5 || 7 || 6 || 1 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
-------------------------------------------------
Time elapsed: 0h 0m 20s
-------------------------------------------------
```

Entering a wrong input



```
Enter the row(1-9): 4
Enter the col(1-9): 3
Enter the number(1-9): 1
-------------------------------------------------
    1    2    3    4    5    6    7    8    9
   ---  ---  ---  ---  ---  ---  ---  ---  ---
1|| 2 || 4 || 3 || 1 || 8 || 7 || 9 || 5 || 6 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
2|| 1 || 5 || 6 || 2 || 3 || 9 || 4 || 7 || 8 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
3|| 7 || 8 || 9 || 4 || 5 || 6 || 1 || 2 || 3 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
4|| 3 || 1 || 1 || 5 || 4 || 8 || 6 || 9 || 7 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
5|| 4 || 6 || 5 || 7 || 9 || 1 || 3 ||   || 2 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
6|| 8 || 9 || 7 || 3 || 6 || 2 || 5 || 1 ||   ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
7||   || 2 || 1 ||   || 7 || 3 || 8 || 4 || 9 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
8|| 6 || 7 || 8 || 9 || 1 || 4 || 2 || 3 || 5 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
9|| 9 || 3 || 4 || 8 || 2 || 5 || 7 || 6 || 1 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
-------------------------------------------------
Time elapsed: 0h 4m 1s
```

After winning



```
----------------Game End-You Won----------------
    1    2    3    4    5    6    7    8    9
   ---  ---  ---  ---  ---  ---  ---  ---  ---
1|| 2 || 4 || 3 || 1 || 8 || 7 || 9 || 5 || 6 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
2|| 1 || 5 || 6 || 2 || 3 || 9 || 4 || 7 || 8 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
3|| 7 || 8 || 9 || 4 || 5 || 6 || 1 || 2 || 3 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
4|| 3 || 1 || 2 || 5 || 4 || 8 || 6 || 9 || 7 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
5|| 4 || 6 || 5 || 7 || 9 || 1 || 3 || 8 || 2 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
6|| 8 || 9 || 7 || 3 || 6 || 2 || 5 || 1 || 4 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
7|| 5 || 2 || 1 || 6 || 7 || 3 || 8 || 4 || 9 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
8|| 6 || 7 || 8 || 9 || 1 || 4 || 2 || 3 || 5 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
9|| 9 || 3 || 4 || 8 || 2 || 5 || 7 || 6 || 1 ||
   ---  ---  ---  ---  ---  ---  ---  ---  ---
-------------------------------------------------
Time elapsed: 0h 6m 17s
-------------------------------------------------
Do you want to play again(y/n):
```

Conclusion:

Making a game like Sudoku was a really competitive and interesting task, we enjoy working on it and learn so much during the process. I hope you would like our attempt too. Thankyou.

END OF THE DOCUMENT
