# Chapter 3. Software Design and Implementation

The Sudoku game is divided into 3 different components: Sudoku game interface, the Solver and the generator.

**1. Sudoku game interface**: This part of the Sudoku game comprises the stand-alone user interface implemented using sfml library with c++ code. It allows the user to select and solve new puzzle by only using keyboard, it also allows the user to select their level of difficulty and solve the puzzle. The interface is provided with simple menu bar and time label to show the time taken to solve the puzzle. The interface is well integrated with solver, generator, and efficiently handling the various events created by the different sfml components and inbuilt functions.

**2. Solver:** This part of the Sudoku game system comprises the logic and rules of the game thereby able to decide the possible next steps to find the right values to the individual cells. The solver uses the scanning technique to find the best solution; the final answer will be the combined result of different rules. The solver can only handling with simple Sudoku puzzles. The solver also checks different conditions while creating new puzzle.

**3. Generator:** This part of the Sudoku game system comprises the logic system to create new random puzzles. The system first creates the full board array with different numbers; the elements from the cells are then taken out randomly thereby creating a new puzzle. Then new puzzle is checked for solution against the solver by applying different rules. The generator is also being able to decide the difficulty ratings of the puzzle depending on the rules applied to obtain the final solution.

## 

## 2.1 Solver

This class mainly focuses on the way to find out the solution to the different Sudoku puzzles. Normally, the strategy of solving a Sudoku puzzle involves two parts: scanning and analyzing. In this work we just concentrate on scanning techniques.

1. void selectTile(){}

This function takes input and store it in entry.value. Beside this it also check if the number is valid in that block or not.

` if(entry.val>0 && entry.val<=size && valid)

{

map[index/size][index%size]=entry.val;

entry.valuee="";

}

After checking its validity and if it is true than [entry value is in between 0 and size i.e 6 or 9] than it palce it in that block. If the validity is false than it make the value of entry.value to 0. Also if false than it makes error++.

## 2.2 Generator

The Generator is for creating qualified Sudoku arrays. Combining with the Solver, we could start to make our own Sudoku puzzles. The main technique in this algorithm is using permutations and checking every possibilities and generate qualified Sudoku arrays. Finally, put the qualified Sudoku array into a text file.

1. void fillinDiagonal(int \*\*m, int size)

* It is the function which check the numbers for diagonal blocks.at first it simply create a random numbers and save it to aux variable. Thus task is done by:

Code: int aux=(rand()%size);

* After creating random numbers it calls 5 different functions:

Code: valid = valiDate(m,size,i,val[aux]) && validcolunm(m,size,j,val[aux]) && validTitle(m,size,i,j,val[aux]) && validDiagonal(m,size,i,j,val[aux]);

* If valid is true than it palce the number in block m[i][j].if it is false than it reset it to 0.

Likewise it check all the diagonal blocks.

2. void fillInmap(int \*\*m, int &size, RenderWindow &window, Data &media, bool &diag)

* This function checks the number in diagonal block as well as in other blocks.Like in fillindiagonal function it also create a random number.

Code: int aux=(rand()%size);

* After getting random number at first it calls 4 functions to check its validity.

Code: valid=validate (m,size,i,val[aux]) && validcolunm (m,size,j,val[aux]) && validTitle (m,size,i,j,val[aux]) && validDiagonal (m,size,i,j,val[aux]);

* This function again check for diagonal elements if the codition in above program is true else it check if number is valid in other block or not.

Code: valid=valiDate(m,size,i,val[aux])&& validcolunm(m,size,j,val[aux]) && validTitle(m,size,i,j,val[aux]);

* After complete filling it makes some block equal to 0 according to difficulty level.

Code: m [i2] [j2] = 0;

