# Sudoku Project Documentation

## Specification

My project idea was to create the game Sudoku in C++, running in the console window. To make this project, I had to pick out the key elements of the game, and composite them in order to understand how I could structure my programming. Sudoku is a well known game, with fairly simple set rules, and it is an easy game for anyone to enjoy. I wanted to make my own version of Sudoku accessible, easy to play, and fun. Firstly, I thought about how I could display and have users interact with Sudoku. I knew I had to output a 9x9 grid, divided into nine 3x3 boxes, so I decided to use a coordinate system for the user to enter the row, then the column, and finally the value they wanted to enter into a specific cell. This method of interaction was simple, and it wouldn’t require constant console screen updates, like other methods I had considered (such as using the arrow keys).

After carefully considering how I could create this project, I created the specification:

**Main screen**

When the user loads up the Sudoku game, a console window will load up, welcoming the user. In this screen the user can choose their desired difficulty level. This will influence the Sudoku grid, with harder difficulties generating more empty cells.

The only user input the main screen should take is an integer input from 1-3 (Selecting difficulty), failure to enter a valid integer results in the user being prompted again, until they make a valid input. I will use error checking to ensure all invalid inputs are accounted for. Once the user enters a valid board difficulty, the program will move from the main screen into a randomised Sudoku board.

**Board**

I plan on storing various completed Sudoku boards in external .txt files. I will have a class ‘Sudoku’ that will handle the reading of these external text files, as well as displaying them to the user once they start a Sudoku game. When displaying the boards, the program must hide random cells from the user, the amount of cells hidden depends on the difficulty they have chosen in the main menu. The program will randomly select one of four boards every time the game is ran. I will use the ifstream functions from the #include <fstream> library to do this. If for some reason the .txt file is not found, the program will inform the user there has been an error starting the game, and the program should close.

The Sudoku class in my program will have a lot of functionality. It will be responsible for the entire execution of the Sudoku game, and will require various methods within the class to do this. The methods I have identified I will need are as follows:   
 -

* selectBoard
* showBoard
* randZeros (responsible for hiding random cells)
* guessNum (Takes user input, checks if it is valid row, column, and guess for specified cell)
* isGameFinished (checks to see if grid is completely filled out or not, ends game when complete)
* winScreen and loseScreen

There will likely be more methods, but I know I will need the ones stated above to make the base of my project.

When the user is playing Sudoku, the program will ask the user for their guess using three stages. Firstly, the program will ask the user for the row number, then the column number. After the program has verified that the coordinates match a valid space on the grid, and that the space is an empty cell, the user will then enter their guess for that cell. If the guess is correct (matching the filled grid from the .txt file), it will be added to the grid, and the user will move on to their next guess. If the guess is incorrect, one of the three user attempts will be taken, and they will be asked to try another guess. Using all 3 attempts will end the game.

Once a user has successfully completed a Sudoku grid, they are congratulated, and I want their time taken to be displayed. After that, the user should be able to exit the program, or play another game (by taking them back to the main page).

## Specification and requirements

For my project, there are various software elements I will need to create in order to create the Sudoku game I have in mind. Because my project has specific requirements such as using classes and external files, I have to keep these in mind when designing and structuring my project.

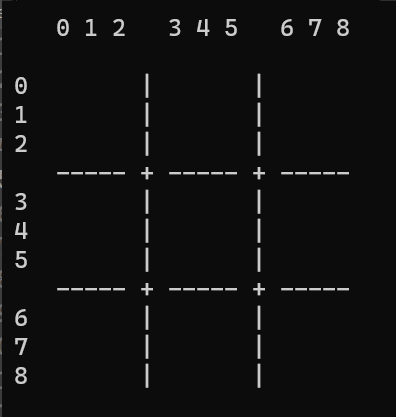


Figure 1 Sudoku grid

My first priority in my programming is to generate the user interface – the Sudoku board itself. I need to have a visual representation of the board, one that is easy to read and interact with. To do this I can use for-loop with nested if-statements that will use a combination of ‘cout’ statements to create a 9x9 grid, which separates each sub 3x3 grid by vertical and horizontal lines. *Figure 1* demonstrates the completed base board for the Sudoku product.

To fill in the Sudoku grid, I will need to have an array which contains values for the board. I want to use external files to hold these values, and have them read using the *ifstream* function, and then store them into an array. Now when printing the board out, it will also need to output the values of the array into the grid, whilst maintaining the grid layout from *figure 1*.

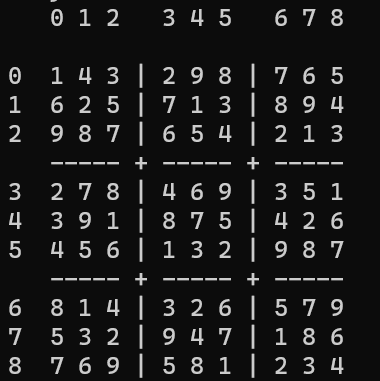


Figure 3 Sudoku grid now outputting with a complete board

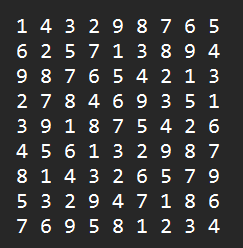


Figure 2 board2.txt

*Figure 2* demonstrates the content of one of the four .txt files used in my program. It is a 9x9 .txt file of numbers, with 1-9 on each and every row and column. *Figure 3* demonstrates the content of *figure 2* being used in the code, outputted into the grid.

After successfully outputting the content of a text file into the console Sudoku grid, I needed to hide random values in the grid. To do this, I needed to create a method ‘randZeros’, which generates random coordinates (row, column), for a certain amount of times. As mentioned previously in my documentation, I want there to be different levels of difficulty for my game, so I set *const int* variables for the difficulty setting. Each variable ‘easy’, ’medium’, ‘hard’, has their own value, corresponding to how many empty spaces the grid will have. Instead of outputting the value of the grid from the text file, the method outputs a ‘ cout << “ “; ‘. *Figure 4* is the result of this:

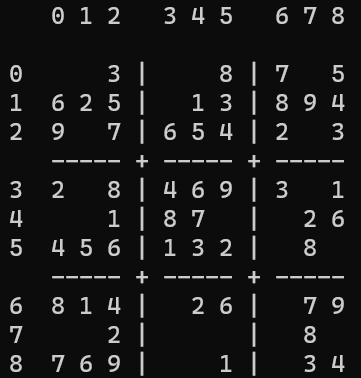
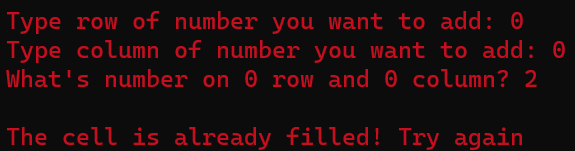
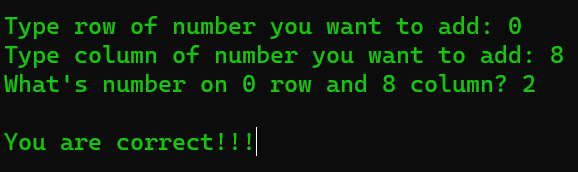
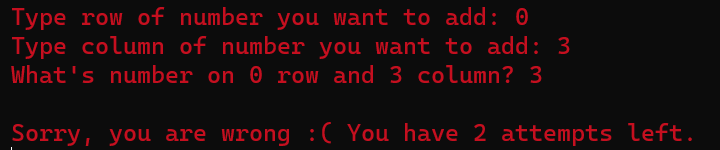


Figure 4 Sudoku grid, now with hidden cells

Now I have a working Sudoku grid, that outputs randomly selected boards from .txt files, and hidden cells in the grid, I needed my program to take a user input, to allow user interaction with the system. As stated earlier, I decided to use a coordinate system for my board, where the user types in the row and the column they wish to put their guess in to. In my program, I did took this input in three steps, to make it as simple as possible for the user.

When the user guesses correctly, the console text colour changes to green, informing them they are correct. It will then update the board with their guess, and ask the user for another guess.



When the guess is in a filled cell, the system rejects the guess, and asks the user to retry their guess. The user is informed their input was invalid.

When the user guesses an incorrect cell value in an empty cell, the system outputs a message telling them they are incorrect, as well as that they have used one of their three attempts.

If not one of the three possible cases above, such as when a user provides an invalid input, the system will reject their current input, and ask the user to retry the specific input which was incorrect. This could be a row, column, or a guess.

Text

Description automatically generated

The game of Sudoku ends in two possible ways:

* Winning the game, by completely filling out the board
* Losing the game, by using all 3 guess attempts

A picture containing text, electronics, keyboard

Description automatically generatedIn order for a player to win the game, my program needs to check the game is finished. To do this, I decided to use a method which checks the game board for zeros. Every time the user inputs a value to the board, the method *‘ifGameFinished()’* is called. The method works bychecking the array which stores the current board to see if each value in the array contains a ‘0’ (which represents an empty space in the integer array). If this method does not detect a zero, the game finishes.

If the user loses the game by using up all three attempts, the game will end. In either ending state of the game, the user has an option to either play again or to quit.

## Testing

|  |  |  |
| --- | --- | --- |
| Test name | Purpose of test | Expected outcome |
| **Main page:** |  |  |
| Displays options | When program loads, does it welcome the user and display three possible difficulty options, and the ability to select one. | Program is launched, console welcomes player and gives them three options to choose from |
| Difficulty selection | After displaying the options, can the player choose one of three options, and does it start a Sudoku game | When a correct option is selected, the console performs a system(‘cls’), and starts a game of Sudoku. |
| Difficulty selection error check | Error checking input for the difficulty selection on main page | When user inputs an invalid response to their difficulty selection (>1, <3, not int input, etc), program asks user to try again. |
| **Sudoku game:** |  |  |
| Display board | After selecting difficulty option, does the program exit the main screen and load up a new game of Sudoku | Outputs a Sudoku grid with the correct amount of empty cells (based on difficulty chosen). Board looks like a typical Sudoku game. |
| Display board fail | The program reads .txt files to access the board content. If the file is missing for whatever reason, the program will need to output an error message. | If the program is unable to find a sudoku board, the program will inform the user there has been an error, and the program will close as intended. |
| Taking user input | When playing the Sudoku game, the player will need to enter values in to firstly choose the row and column they want to guess a value for, and then the value they are guessing | Program will ask the user for their guess using 3 steps:  -enter row  -enter column  -enter guess |
| Taking invalid user input | The program must handle erroneous inputs, such as values outside of the grid, or wrong data types such as strings. | Upon entering an incorrect value, the program catches the invalid input, and asks the user to try again. This happens for the row, column, and guess inputs. |
| Correct guess | If the user correctly guesses a Sudoku cell, the program should congratulate the user, add their guess to the game board, and then move on to ask for a new guess. | When guessing correctly, the program outputs a congratulation message, updates the board to include their guess in the specified cell. Moves on to asking for next guess. |
| Incorrect guess | When guessing a value wrong, the program must take away one of the three user guess attempts, and inform the user they were incorrect. | The incorrect guess returns a message informing the user they were incorrect. One of three attempts are taken away, and the user is asked to try again. |
| **End screen** |  |  |
| Winning screen | Sudoku game has been completed by the user, the game should stop asking for guesses, and move to a victory screen. | Once the program has confirmed there are no empty cells in the Sudoku array, the game will stop taking user inputs. It will congratulate the user, output their time taken, and ask the user if they want to play again or quit. |
| Loser screen | Three attempts have been used up, use entered incorrect values three times. Game should end and inform them they are a loser. | The program will end, and provide the same options as the victory screen – for playing again or quitting. |
| End screen option input | User should be able to play Sudoku again, or quit the game | By entering [1], the user will be taken back to the main page, where they can enter their difficulty choice again, and start a new round.  By entering [0] the user will quit the game, and the program will close. |
| End screen input invalid | Entering an unexpected input, outside of the 0, 1 options, the program should catch the invalid input and inform the user to try again. | If the input is incorrect (>0, > 1), or an invalid input such as a string, null input, etc, the program will ask the user to try again until they enter a valid response. |