# 

# END SEMESTER PROJECT

# SUDOKU

# A PROJECT BY MUHAMMAD HASHIR AND SHEIKH ANAS ABDULLAH SIDDIQUI

# SUBMITTED TO:

# MA’AM PAKEEZA AKRAM

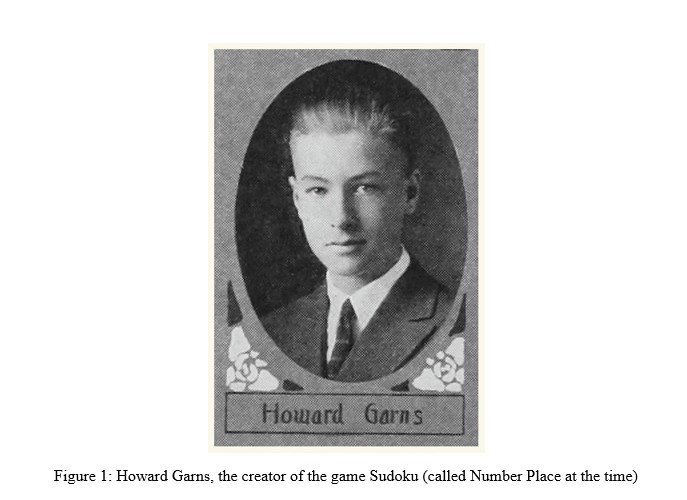
## ABSTRACT

The purpose of this report is to explain the workings of the Sudoku game that has been developed, explain the various logic that has been implemented and show what the code gives as output, and how the game is played.

## INTRODUCTION

### 2.1 Idea of The Game

While Sudoku has been around for a long time, dating as back as the late 19th century in French newspapers. However, the modern form of Sudoku is relatively modern, owing its creation to an American puzzle maker named Howard Garns. The basic idea of the game is that it consists of a 9x9 grid, with it being further divided into 3x3 blocks which make up the whole grid. The grid is filled with some numbers, while some boxes are empty and require the player to fill it in. However, there exists a condition for the filling of the boxes, which is actually the crux of the game. When the player fills the boxes, they have to keep in mind that the numbers in the boxes must not be repeated in the same row, column or the 3x3 block.



### Why Sudoku was Chosen

Sudoku has always been a fun-to-play game since childhood. Other than just being plain fun, it also presents many forms of benefit to the mind as well including improving memory as well as concentration power, learning to do things efficiently, and has even been shown to lower the risk of Alzheimer’s in people. Such a game is fun for all ages and provides them with the opportunity to widen the horizons of their minds and fully unlock its potential.

## REVIEW OF LITERATURE

Various concepts of Object-Oriented Programming were taught prior to starting this project. They have been vital in the completion of this project and has helped the group in making sleek and elegant code. They have allowed the code to be compact, reusable and has helped hide unnecessary code from the end user. They have also made a more elegant output, with a well-made user interface as well as providing the users with the ability to interact with the code in order to play the game. They are as follows:

### 3.1 Data Encapsulation

Data encapsulation is often defined as binding of data into one unit. This is usually done in order to hide the various variables or the data of the clas, so that other classes would not be able to access it. In order to access this, there would need to exist some form of function that allows changing of the data within the class. The purpose of this concept is to allow the programmers to hide their data so that each programmer only works on their own data without disrupting any other work done.

### 3.2 Inheritance

In this concept, there exists a form of a hierarchy between multiple classes. It can be one-to-one or one-to-many. While many-to-one relationship also exists, it cannot be used in Java. This works by a class adopting the characteristics (variables and methods) of another class. Due to this relationship’s similarity with parent and child, the class often providing the characteristics are called parent class while the class receiving them are called child class. The purpose of this is to allow multiple classes to have the same characteristics to improve code by making it less redundant.

### 3.3 Polymorphism

Polymorphism can literally be taken to mean “many forms” and its function is indeed very similar. With the use of polymorphism, one is able to make two methods of the same name and allow them to perform different functions. Polymorphism is further broken into two parts, which are as follows:

* **Compile-time Polymorphism -->** This is done by making methods of the same name but changing the parameters so that the computer is able to distinguish between the two. This is also called method overloading.
* **Runtime Polymorphism --------->** This is when a derived class has the exact same name as well as parameter. This overrides the method taking place in the parent class, hence the reason it is also called method overriding.

### 3.4 Exception Handling

A method by which the code handles runtime errors. These may include: ClassNotFoundException, SQLException, IOException and RemoteException, etc. The purpose of this is to allow proper flow of the code as exceptions usually disrupt the code and create problems for the coder.

### 3.5 Graphical User Interface

Graphical User Interface(or GUI in short) is making the code in a way that it seems attractive to the user. For this purpose, there are two methods to make GUI’. One of them is Swing and the other is JavaFX. Swing is slightly older while JavaFX is the modern version of Java GUI. In addition to this, with the use of FXML’s, one can make GUI’s even easier.

## METHODOLOGY

In the construction of this code, a number of classes have been constructed. Some of them are their own classes while some are connected to one another through hierarchy. However, there are also some FXML files built from Scene Builder which help provide the user with the proper tools to construct their scene.

Now we will continue by discussing each of the classes in detail, by briefly summarizing the contents as well as the logic behind each method in the class.

### 4.1 Board.java

The Board.java class plays the role of generating the Sudoku grid for the complete code. In all of these methods, in addition to the grid that the user observes, two other grids are also filled. One of them is the solution Grid, which shows the correct answers to every box and is used to test the grid, and the reference Grid,

### 4.1.1 Constructor method

The constructor method (or Board () ) is used to implement all the other methods that are present within the class that generate the grid. It consists of a number of method-calls which include the filling of diagonal matrices first, then the non-diagonal ones, and then finally emptying the boxes. The logic behind filling the box in such a manner will be explained in the upcoming methods that implement this logic.

### 4.1.2 public void fillDiagonal ()

The purpose of this method is to fill the diagonal 3 by 3 matrices with numbers. The logic behind the generation of this grid is that first we fill the diagonal matrices in a specific way, and then fill the non-diagonal in another method.

In this method, we make sure that only the diagonal rows are filled, and for that we use the multiples of 3 in the for loops, and increment it at the end of the code. After we access the correct boxes of the grid using two nested loops, now comes the method by which we fill diagonal matrices. First, a random digit is generated, then using two methods in the class, checks if it has been repeated in the same row, columns or the same matrix. If it is so, it enters the number into that specific position in the grid. Otherwise, it just repeats itself until it finds a suitable number to be added into the grid. In this way, all the boxes in the diagonal matrices are filled.

### 4.1.3 public void fillNonDiagonal ()

This is used to fill the non-diagonal matrices after the diagonal matrices have been filled. Accessing these matrices require four different forms of nested for loops in order to cover all of them.

In all of these nested loops, the logic is the same. There is a third for loop in all of these loops in which a number starts from 1 and goes up to 9, and at each value, it is checked using the same two methods used in the diagonal matrices method and until a suitable number is found, keeps incrementing up till 9.

### 4.1.4 public Boolean check (int row, int column, int x)

This method is used to check for repetition of a number within both the rows and the columns. Using a for loop that goes from 0 to 8 (9 iterations), it first checks the rows and then the columns

### 4.1.5 public Boolean checkMatrix (int row, int column, int x)

This method is used to check for repetitio within the 3x3 matrix that the number is placed. Using a number of if-statements, it decides the loop according to the row as well as the column that it is accessing and checks for repetitions using that. It checks up through 3 in order to complete the matrix.

### 4.1.6 public void EmptyingBlocks ()

In order to remove random cells in the grid, this method is used. Firstly, a for loop is used to access each row, after which 4 iterations are processed. In each iteration, a random cell is accessed and filled with 0 until all 4 iterations are completed, after which it moves to the next row and continues the same process again.

### 4.1.7 public void setMoves (int x)

This is a setter function for number of moves

### 4.1.8 public int getMoves ()

This is a getter function for number of moves

### 4.1.9 public int countEmpty()

This method is used to count the number of empty boxes, in order to decide the number of moves, which is different for every difficulty.

### 4.1.10 public int [] [] getGrid ()

This is used to return the complete grid in order to make the puzzle for the player.

### 4.1.11 public void editGrid(int num, int row, int column)

This is used to edit a location on the grid in order to solve the puzzle

### 4.1.12 public void checkrefGrid()

Method to check user's answers. The whole of the sudoku grid is stored inside the 2D refGrid array. So, if the number at a particular cell matches the number at the same cell of the refGrid, it means that the number is correct. Therefore, every number entered by the user is checked against the number at its corresponding cell in the refGrid array. If the two numbers do not match then the number at that cell in the grid array is removed.

### 4.1.13 public void setChecks(int x)

This is the setter function of Checks.

### 4.1.14 public int getChecks()

This is the getter function of Checks.

### 4.1.15 public boolean CheckIfWin()

This is used to check if the player has won the game or not. In order to do this, it compares all 81 entries of the grids according to a reference Grid. If all the number match, it shows that the game has been won, otherwise it continues the game until either the game is won or the players runs out of either checks or moves.

### 4.2 BoardEasy.java, BoardMedium.java and BoardHard.java

These are all child classes of the Board class. The purpose of these classes is to make different instances of the code depending upon the difficulty. In this program, the difference in difficulties was defined by the number of checks as well as moves provided to the player. This is done by using the parent class’s (i.e. the Board class) methods, namely setMoves() and setChecks() in order to set the difficulty.

For example, in the Easy class, moves are set to 15 plus the number of empty boxes while the number of checks has been set to 7, for Medium, it is 10 more than the boxes and 5 respectively and for Hard, moves equal to number of empty boxes and the number of checks have been set to 3.

### 4.3 GOController.java

This is the controller class for the GameOver FXML file. The purpose of a controller is to provide code for the various controls used in it, so that some action be completed when they run. In this controller class, there are two components. One is the Exit Game button, one is the ImageViewer which shows GameOver and there’s a label which shows the status of the game (Win/Lose).

The other component is the Exit Game button. When this button is used, it accesses the System.exit() function in order to terminate the complete code.

Since the image viewer and the label are created in the FXML using the scene builder, instead of coding in the controller class, the initialize function allows us to display image and text according to the situation, as soon as the FXML is loaded instead of displaying the same image and text, which is the case if the image and text are hard coded in the FXML.

### 4.4 Main.java

The purpose of this class is to initiate the code. It is the file that is executed in order to start the game. Hence, it may be called the entry class. It consist of a stage upon which the FXML Loader loads the MainMenu.fxml file.

In order to run the FXML loader, it has to be surrounded by a try block so that in case of any runtime error, some sort of exception is thrown so that the code would be able to stop.

### 4.5 MenuController.java

This is the controller class for the MainMenu.fxml file. It consists of three buttons that are mapped on the FXML file, namely Easy, Medium and Hard. The purpose for these are to select the difficulty. In addition to this, a Stage is also declared in the beginning. The purpose for this will be explained later.

There are four methods in this class. They all perform rather basic options just helping the code to proceed ahead.

### 4.5.1 setLevel(int x)

This sets a value to the variable lvl that is actually present in the SudokuController class and is used as a static variable, so that it can be changed in this class. The purpose for this is to help set a difficulty by putting an if condition in the SudokuController class so that the necessary operations take place.

### 4.5.2 EasyPressed, MediumPressed, HardPressed

All of these methods can easily be defined under a single heading as they all play a very similar role. All of them, when pressed, only load up the Main FXML file which the actual Sudoku grid. This is done with the help of the Stage previously declared. Loading the FXML file into a scene and then that scene into the Stage helps it to be run smoothly.

The only major difference is the difference in difficulty. With the press of each button, firstly a setLevel(x) method is called, however “x” is different for each one. Pressing Easy means that 1 is set in the method under discussion, 2 for Medium and 3 for Hard.

### 4.6 SudokuController.java

This is the controller class for the Main FXML file. It consists of 10 buttons, a canvas and 3 labels. Nine of those buttons are the 1 to 9 buttons, their function is to fill the selected box with the number specified in the buttons. In addition to this, there is a Check button which checks the grid. The entire Sudoku puzzle is cast on the canvas and is accessible by the player. The labels are there to show the player the number of moves and checks remaining for them in order to play the game, as well as an error label which shows the exception if the user tries to access the predefined values. The methods bound to the buttons are as follows:

### 4.6.1 public void initialize (URL arg0, ResourceBundle arg1)

This is the initialize function in order to run the FXML file along with the methods that are bound to the various controls.

The concept of polymorphism is applied here. We have declared an object of "Board" type above. The different levels of the games have different number of maximum allowed moves, and maximum allowed checks. The user must solve the grid before running out of the remaining moves, and before using all the checks. Therefore the three sub-classes BoardEasy, BoardMedium and BoardHard, have different number of moves and checks. Since all their other properties are same, we simply re-use the code, by using the concept of inheritance. Based on the level chosen by the user we have to create an object corresponding to that sub-class. To make this possible, we declared an object of the "Board" type, and inside the initialise function we decide its actual type, ensuring that a grid of the corresponding difficulty level is loaded, if the user selects Easy, lvl = 1, and hence we set the actual type of the object to BoardEasy(),actual type = BoardMedium() if lvl =2, actual type = BoardHard() if lvl = 3.

### 4.6.2 public void drawOnCanvas(GraphicsContext context)

This is the method to draw Sudoku board on the Canvas. First, we clear the whole area of the canvas of any drawing. Then, using nested loops we create a 9 by 9 grid of rectangles. Since the canvas is 450 by 450 and we need to make 9 rectangles in each row and 9 rectangles in each column it gives us a 50 by 50 area for each rectangle. So to get the vertical position of the rectangle, we multiply 50 by its row number. 2 is then added to add the offset. Similarly for horizontal position column number is multiplied by 50 and 2 is added. Then the rounded arc rectangle is made using these calculations. Then a code is made to draw the numbers that the users enter. The initial grid and the numbers that the user enters all are stored in the 2 dimensional array called "grid". In addition the computer generatednumbers of the grid are also stored in the 2D array "solGrid".

### 4.6.3 public void MouseClick() throws InvalidCellException

This is basically used upon the Canvas, allowing the user to click on the various boxes on the grid so that they could be accessed. In order to this, we have made the use of coordinates so that this task could be easily accomplished.

### 4.6.4 public void Button[n]Pressed() throws InvalidCellException

Here, n is the numbers 1 to 9. The purpose of these buttons is to replace the selected box with the number that is under consideration (equal to n). This is done with the help of Strings.

There is another concept applied in this, which is exception handling. If the user tries to fill one of the boxes that was already predefined by the game, it throws this exception in order to allow the natural flow of the code.

### 4.6.5 public void CheckButtonPressed() throws Exception

This is used to check the grid and make sure that the user entered the correct number in the grid by comparing it with the solGrid[][]. If it is so, the number stays in the grid. However, if it is not so,it is removed. In this way, the user can make sure not to make the same mistake again.

### 4.7 application.css

This is the .css file, using for styling of the Main FXML file in order to make it more attractive for the players. Using hexadecimal codes, the colors of various things such as the background, the buttons as well as their state when the mouse cursor is hovering over them, can be set as well as easily changed as Java recognizes the colors by their respective color codes.

In this file, the background has been set to a dark blue, the buttons to a light gray shade, and turn soft white when the button is selected.

### 4.8 GameOver.fxml

The FXML file for the GameOver screen, it consists of an AnchorPane. Within the AnchorPane, within which an ImageView is constructed in order to show the GameOver.jpg picture, which depicts a Game Over sprite. In addition to this, a button for Game Over exits for termination of code and a Label which shows the game of the status, if the player has won or lost the game.

### 4.9 Main.fxml

This is the FXML file for the Sudoku scene. Within its pane, it consists of a Canvas. On that canvas, there are 10 Buttons, from 1 to 9, each one of them allowing the player to populate the selected box with the text within the button as well as a check button to check the Sudoku grid

In addition to this, there are also a few Labels. Labels which shows the number of moves as well as checks, and a label which throws the excpetion if a predefined number is tried to be accessed

### 4.10 MainMenu.fxml

This is the FXML file for the Main Menu scene. It consists of AnchorPane, within which two label exists which shows us that we are playing Sudoku as well as instructing us to select a difficulty

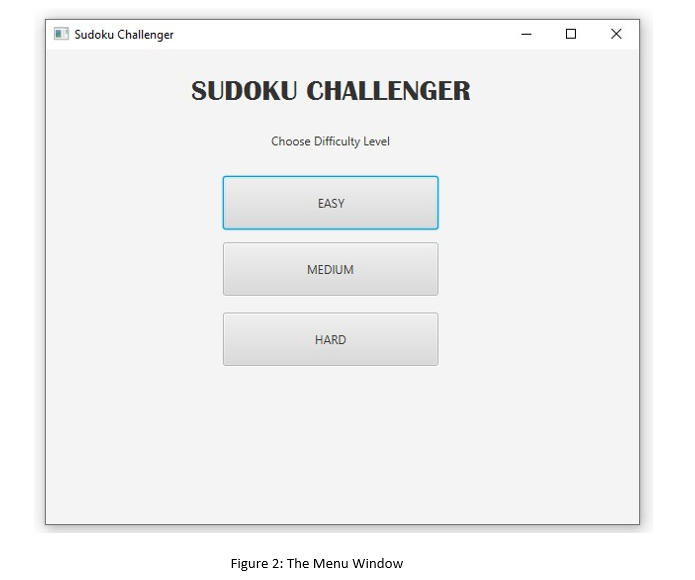
In addition to this, there also exists three buttons labeled Easy, Medium and Hard. These are the three difficulties that are available to the user.

## RESULTS

The result in this particular may well be regarded as the output of the code, and as the idea behind this code was to construct a game of Sudoku. Therefore, running this code starts the game. There are mainly three windows(stages) that operate during runtime and every window closes itself upon the startup of the following window. These stages are as follows:

### 5.1 The Menu Window

The first window that opens when the game is started and greets the user with the title of the game. In addition to this, there are also three buttons present in this screen which allow users to play with the difficulty they want to. They are titled as “Easy”, “Medium” and “Hard”. Pressing each button starts up the next window, which is the actual game window where you can play Sudoku.



### The Main Window

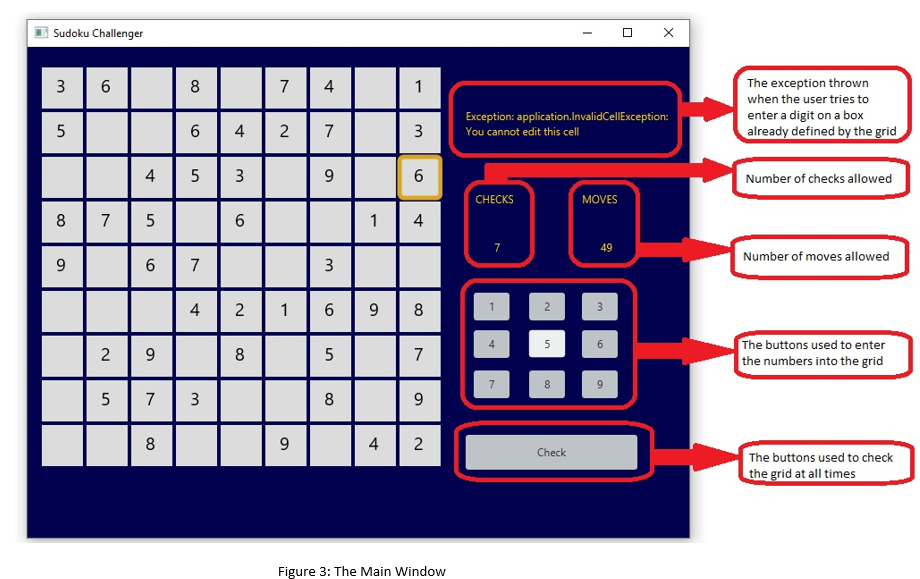
After the Menu Window, upon selection of a button, the next Window that opens is the Main Window, where the actual game is played. The selection of one of the three buttons actually influence the Main window as well, as the various choices of difficulty affect the number of moves as well as the number of checks of the game. Pressing Hard would allow users to have only 3 checks and moves equal to the number of empty boxes in the Sudoku. The Medium difficulty allows user to have 15 more tries then the number of empty boxes and the number of checks are a total of 5. Finally, the Easy difficulty gives the user 30 more tries in addition to tries equal to number of empty boxes while the numbers of check have been increased to 10.

In this window, exception handling has been used in order to make sure that while playing the game, the user only accesses the empty boxes and does not alter the predefined boxes. If the user tries to do that, an exception is thrown showing that the user cannot alter this box.

The game will end if one of these three conditions are fulfilled:

1. The number of checks decrement to 0.
2. The number of moves decrement to 0.
3. The user wins the game.

This will, in turn close the Main window and open the Game Over window.



### Game Over Window

The last window among the three is the Game Over window. This window plays a number of roles, even though it is the most basic of the three.

The first role this window plays is showing the status of the game. If the users had completed the game before the either the number of checks or moves had finished, the label present in the middle of the screen would show the text, “YOU WIN!” as shown below.

Otherwise, if either the number of checks or moves run out before the player is able to finish the game, the label would show “YOU LOSE!” as its text. This can be seen below.

The second role that this screen plays is the ability to exit the game by pressing the Exit Game button. This in turn, triggers the System.exit() function and allows the code to be terminated.



## CONCLUSION

To conclude, it can be said that this project required us to use our minds to its fullest, firstly making the logic of the Sudoku puzzle to allow it to become the way it is right now, and afterwards making a complete code of it, in addition to the work done behind learning how to use JavaFX in order to create the GUI. This game allowed us to use Java to its complete extent as well, using all the relevant concepts that make up Java as well as generally Object-Oriented Programming such as Inheritance, Polymorphism etc. This has allowed us to learn a lot about the language, as well as understand what it takes to become a successful computer scientist.

## AUTHORS

* **Muhammad Hashir-------------------BS-CS-9B----CMS ID: 282562**
* **Sheikh Anas Abdullah Siddiqui----BS-CS-9B----CMS ID: 305705**