CS 443 – SPRING 2021

Sergio Augusto De Sa

<https://github.com/S2R3/Sudoku.git>

**Final Project Report**

**1- Project Statement**

The purpose of his project is to implement an android app for a sudoku game.

The app will self-generate a random sudoku solution.

The main reason for the choice to develop this application is due to the opportunity to build a useful tool that applies some of the topics learned in CS 443. At the same time, it offers challenges, that are useful to solidify the concepts learned in class.

The app will benefit any user interested in solving sudoku.

There are several similar apps available in the Google Play store. The final version of this app has comparable capabilities.

Users can play the Sudoku game using this app without restrictions.

The app enables the user to:

- Choose inputs to add to the grid.

- Delete their input in sequence. Last in first out.

- Delete any user input, in any order.

- Hint button, show the users where they have errors.

- save the state of the grid when desired, and make it possible to access later.

- change the color of all values in the grid that match the input of choice to add to the grid.

- display counter time when the game start.

- display user choices different from the solution in red.

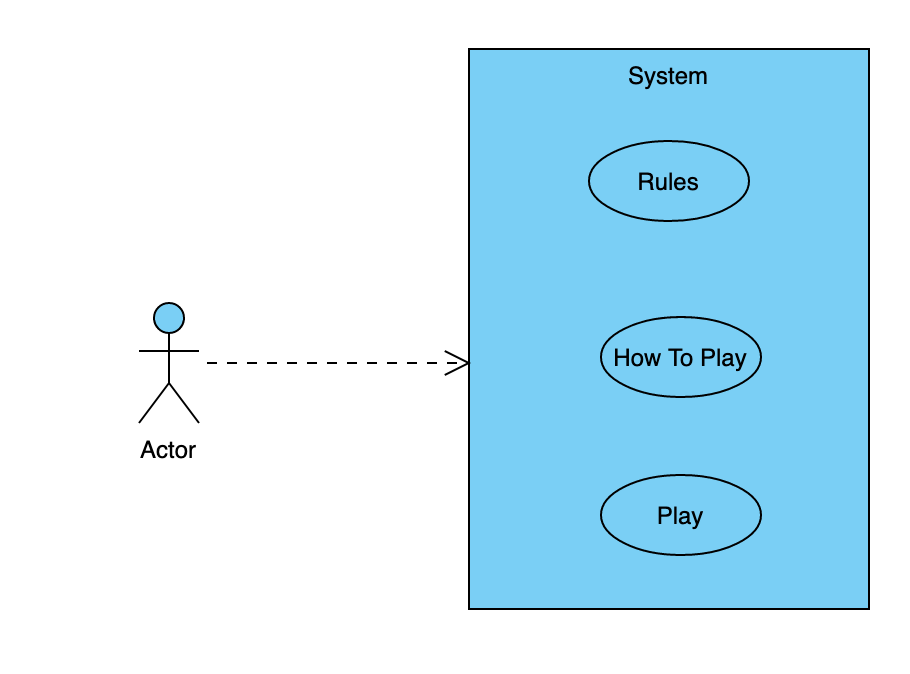
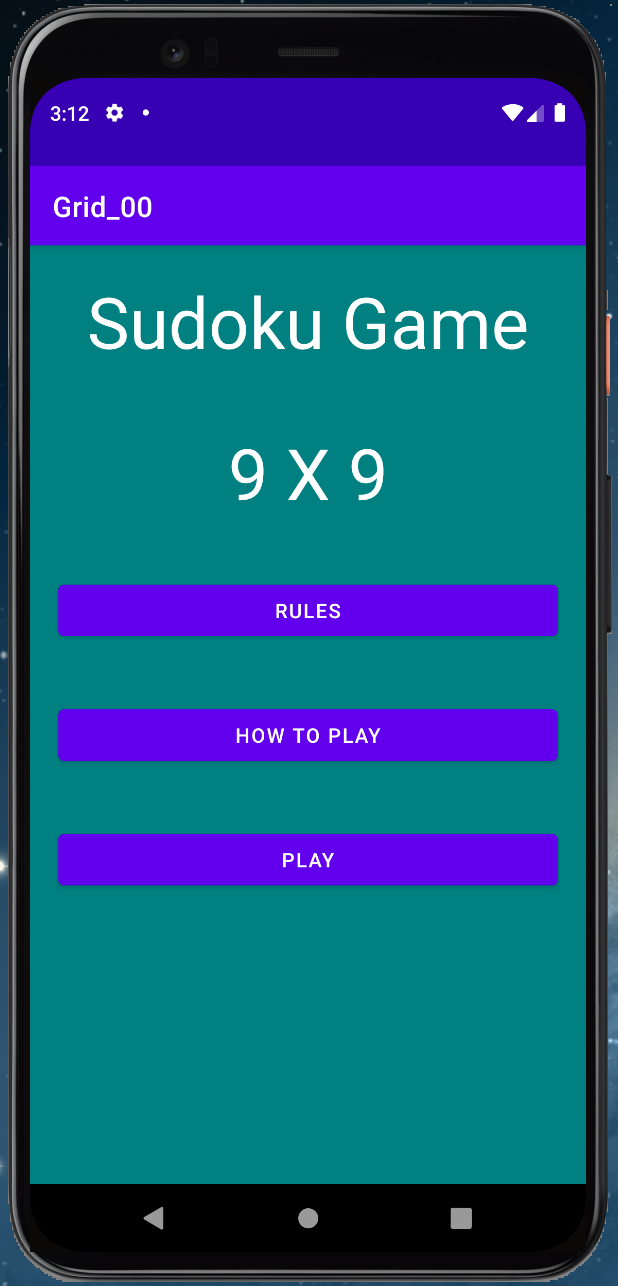
- display the time it took to finish the game.

**2 – Application Design**

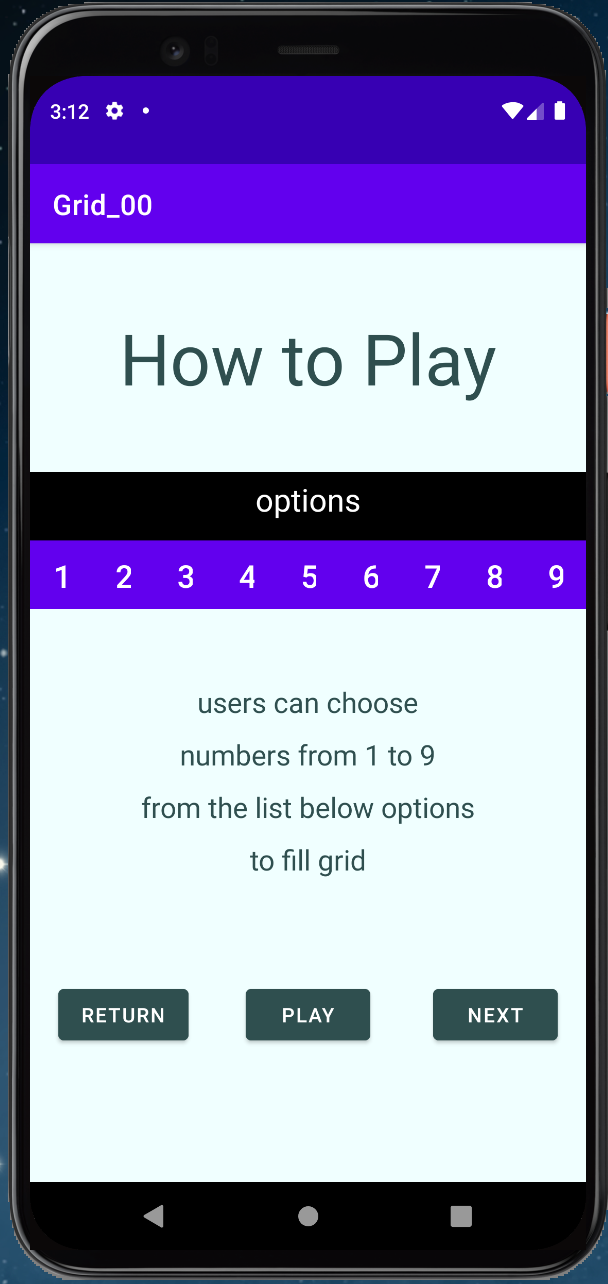
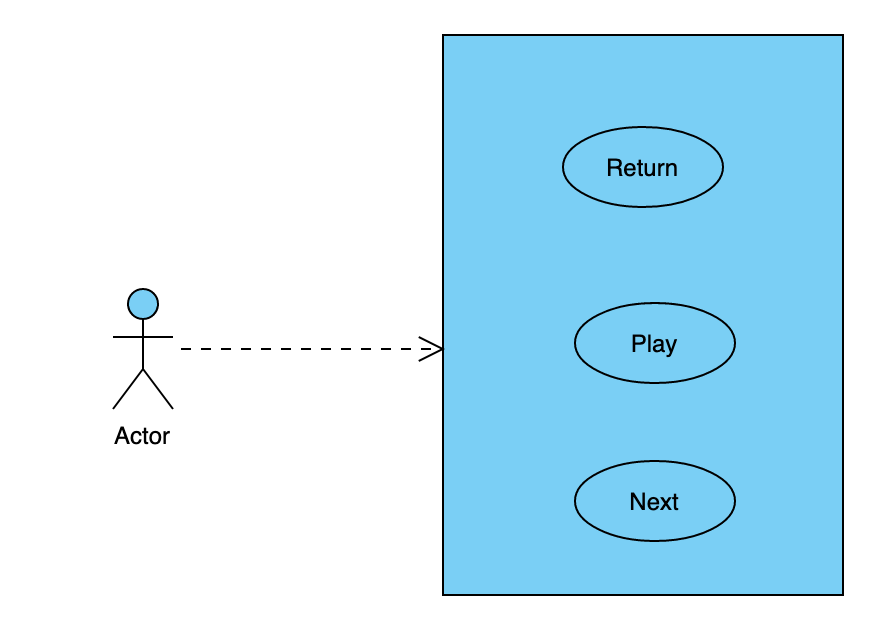
The Sudoku grid uses an Array to simulate a Matrix of nine rows and nine columns, with a total of 81 inputs.

This app was designed with the idea of creating a Random solution to the grid. After the solution is generated, it gives some of the inputs to users and hides others. Therefore, users will have to find the missed inputs.

The initial UI gives users a choice to read rules, find out how to use the app, or play the game.

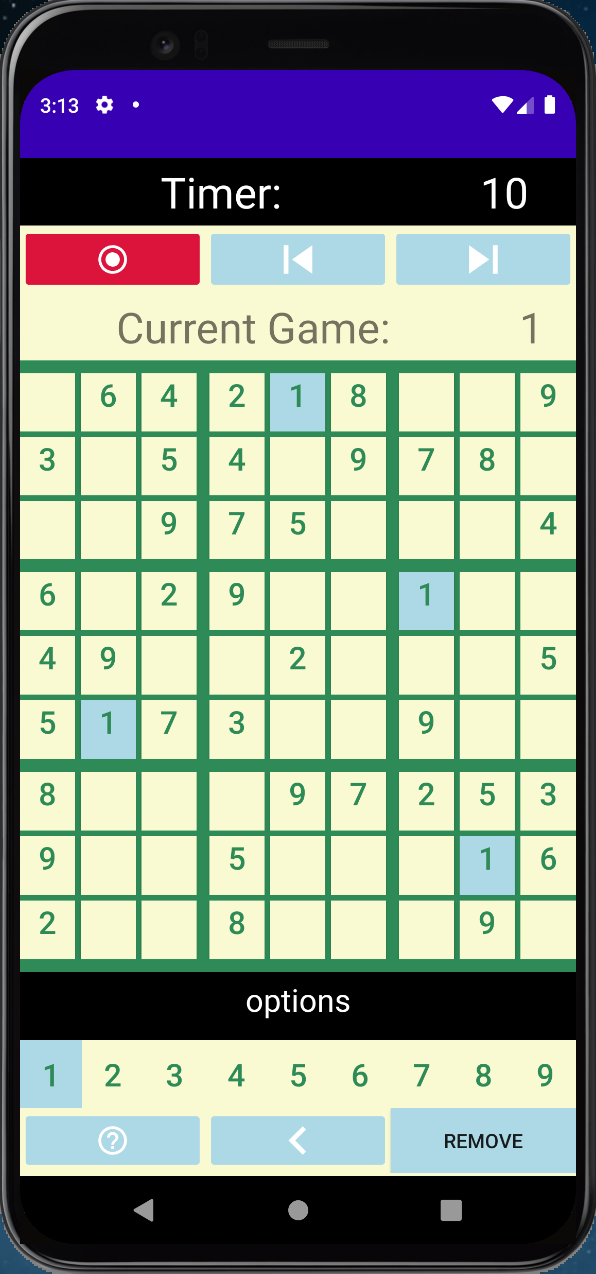
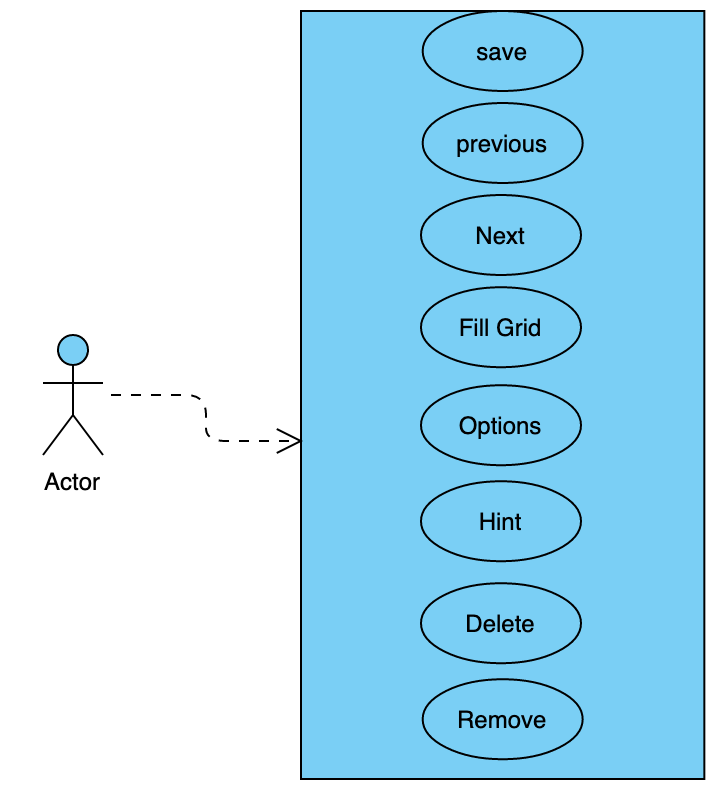
 

Users have several views to choose from. Each view is a pair of XML and java. The views give the user the choice to return, play the Sudoku game or advance to the next view.



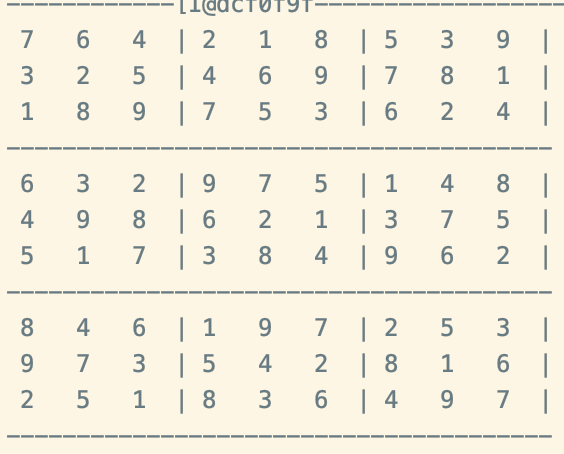
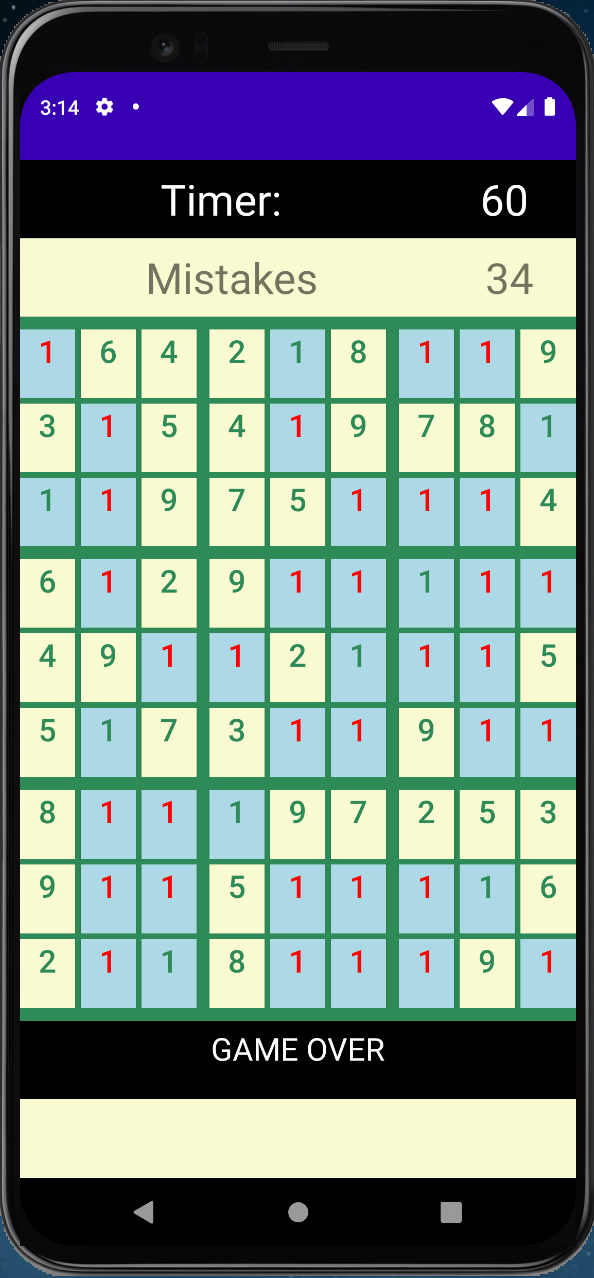
When the user clicks play, the game Sudoku start.

Once the user decides to play the game, the app will generate a random grid where users can play the game. Once the grid is generated, a timer displays the amount of time spent using the app.



After the user enters the last input available, the grid timer stops. The number of errors is displayed, and the incorrect inputs are changed to the color red.

The example below displays a solution and the final screen whit errors in red.

The XML provides the static interface for each view, while the java class is responsible for the dynamic interaction of UI.

The XML files use different types of layouts, Buttons, TextView, ImageButton.

The Java classes used for UI extend AppCompatActivity. There are other Java classes to support other operations required to use the app.

This app was created to use the phone as a target. Possible, several other kinds of devices can run the app with some reasonable useful UI.

**3 – Application Implementation and Evaluation**

There are several classes of Java and XML in this app.

**XML**:

*- Intro\_1.xml,*

It is the first UI available to users.

From this class, users can choose to play the game, read the rules, or visit a sequence of Views UI showing the capability of each operation available to users.

*- rules\_layout.xml*,

Show the user the rules of Sudoku and gives the user a choice of what to do next.

- *grid\_layout.xml*,

The most important of all classes under the layout. This class is responsible to display the grid, to accept user input, and to give user choices to play the game.

There are several combinations of layout under this class, but at the root is a LinearLayout. The other layouts are bounded to layout\_weight, which gives the app the ability to scale the UI on different devices.

- from: *intro\_2.xml* to: *intro\_9.xml*,

Those classes are mostly identical to each other.

Each class basically, gives some information about a specific game UI choice under the main grid.

The major difference among these classes is where they point to. Each class point to the previous one, the next one, or play the game. Users can choose where they want to go using one of the three-button available in each of these classes.

**JAVA**:

- From: *Intro\_1\_Activity.java* to *Intro\_9\_Activity.java* and *Rules\_Activity.java*

These classes give dynamic support to the UI from the XML files.

For the most part, the classes implement the listeners to answer user’s interaction.

- *Grid\_Activity.java*,

This is the most important class among the classes answering user interaction.

It is responsible to support all users’ choices to play the game. Every other Java class not listed so far was created to give support to Grid\_Activity.

-*Grid.java*,

The purpose of this class is to manage the UI grid.

This class instantiates a new instance of Grid.java.

The class will provide the operations required to update the user choice of input to fill or delete the Sudoku grid.

- *Control.java*,

The purpose of this class is to create a new instance of Game.java. In addition to that, the class will work as a manager of the game. Answering user interaction, and updating the game as necessary.

- *Game.java*,

The purpose of this class is to populate the grid.

The class uses an Array, ArrayList, and stack.

Game instantiate a new instance of Sudoku.java class. The grid solution is generated in Sudoku.java. An array is used to save the solution in Game.java. A second array is created with some of the inputs from the solution or zero. This array will update user’s input.

This class is responsible to save and recover the state of the game according to user’s interaction. To support this operation, a new instance of State.java is created for each new state.

There are several methods to support the saving and recovery of states.

- *Option.java*,

The purpose of this class is to save the buttons the user selected as a value to fill the remaining grid buttons.

The class instantiates an array to hold nine buttons. Each button is associate with a value ranging from one to nine. The values are the choice users have to fill the remaining grid values to solve the Sudoku.

Option class offers several functions to support user interactions.

- *State.java*,

The purpose of this class is to save the state of the game.

Each instance saves the user grid solution according to a user’s request.

There are several functions to support state recovery.

- *Sudoku.java*,

This class is responsible to create the solution for the Sudoku app.

There are four solutions saved under this class. The final solution is generated using a combination of random choices and the values of one of the four saved solutions.

To create a unique user solution at each new game, this class first draws the first row with random values from one to nine. The first row is completely random. The probability for the first row is 9 \* 8 \* 7 \* 6 \* 5 \* 2 \* 1 = 362880 different choices. The values of the first row will, again at random choice, choose one of the four pre-saved solutions. Each cell of the first row will map to the corresponded cell of the pre-saved solution first row. The value found at the cell of the pre-save first row will map to eight other cells with the same values on the other eight rows of the array. The final solution will use all the positions of those values and fill their value using the value from the first-row value. This process is repeated eight more times, and it results in a final solution completely different at each new game.

- *Print\_array.java, Print\_Activity.java*,

The purpose of both classes is for debuggers during app development.

Currently, those classes are not useful but were left here, for future improvements.

- Notes:

All the classes above were tested individually during development phase.

I used the Log class and some print statements placed in locations to confirm the expected values.

New features were added and tested several times after the game proves to work.

The app was tested using several different inputs. All the problems found were addressed before the final version.

A problem was found when displaying the game's final screen. The app was using an int to track the grid updates. The tracker was failing to update users' deletion. As consequence, on some occasions, the final screen was displaying before the user's final choice. The int was replaced with a boolean, and the app checks the array holding user input before display the final grid. No error was found by delivery time.

**4 – References**

There is no major use of outside work for this project.

The project was completed using the resources available in CS 443 support material, the class book, and Android API.

**5 – Experiences and Thoughts**

This project was more challenging than expected.

The initial idea was to create a complete random Sudoku solution. However, it was not possible to find a solution to this problem due to the time required to finish the project. After several tries, I concluded that it is not possible to have a random solution without deleting some of the last random choices found and choose new values. I believe I got close to finding a way to a random solution, but I decided to go in another direction for safety. Chances are, I would never find a completely random solution.

The grid to display the Sudoku game was challenging. I try the same gridView used in homework 1, but it was not displaying all the rows as I expected. To solve this problem, I used several layouts before deciding on the final solution.

I decided to use buttons to hold the values of Sudoku. The hardest part was to leave a gap between the buttons and keep all of them aligned. It was complicated because I wanted to have a wider gap every three cells to display the 3 by 3 square. The XML file is over one thousand lines, but mostly is a repetition of buttons. Certainly, other solutions are better to this problem, but my goal was to come with my original solution in time to finish the project under the deadline.

I had the intention to add a button “show the solution” after the game ends. In addition, another button “play again”. However, I could not find a way to do it before the deadline.

**6 – GitHub:**

The app is available on github.com at:

<https://github.com/S2R3/Sudoku.git>