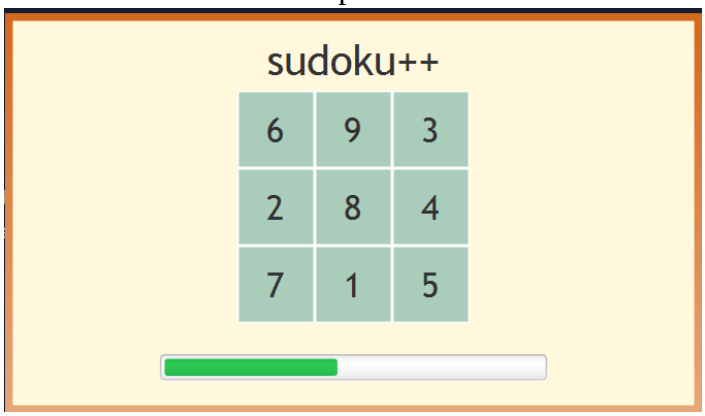
CS3233 Project Report
Chua Wee Chong M21302

Summary

Sudoku++ is an app that supports the creation and solving of Sudoku. The app allows users to construct several different types of Sudoku as well as save and open Sudokus in the app.

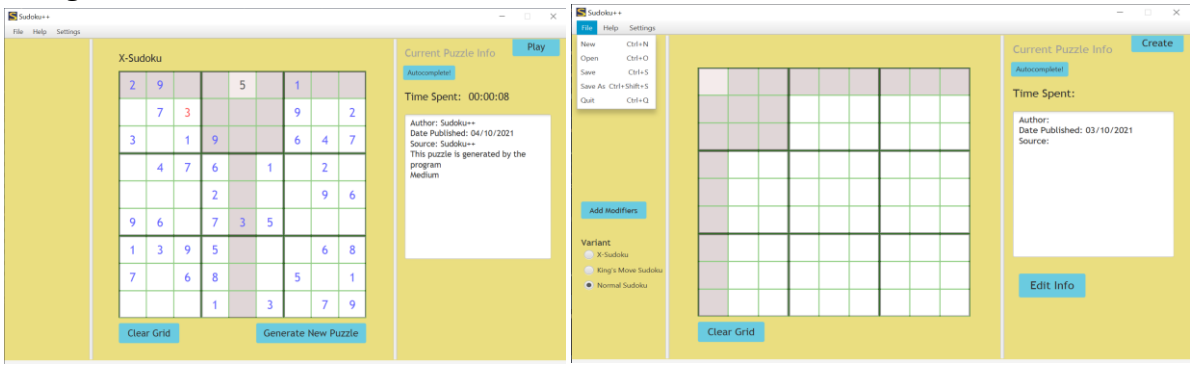
Splash Screen

The animated splash screen has a 3x3 Sudoku grid progressively filled then removed with numbers. A frame of the splash screen is shown as below.



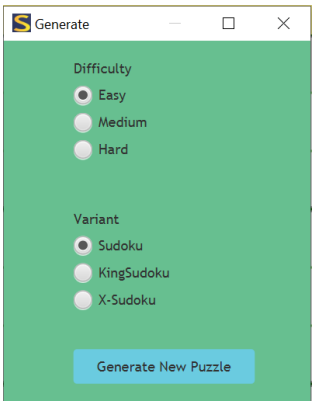
Main App

The centre Sudoku grid can be filled in with numbers, and only numbers. The Sudoku grid is implemented with 9 rows of 9 text fields in a GridPane. For normal numbers, If the number is conflicting with another number in the grid, it will be a different colour, by default red. There is a third colour for pre-set numbers. Where the pre-set number comes from will be discussed later. For pre-set numbers, they will not be coloured the conflicting number colour when it is conflicting, as they are the numbers from the original puzzle set. The back of the centre grid has an ImageView, which gives it the black grid lines. Additionally, there is another background for Killer boxes, for when the modifiers are added.



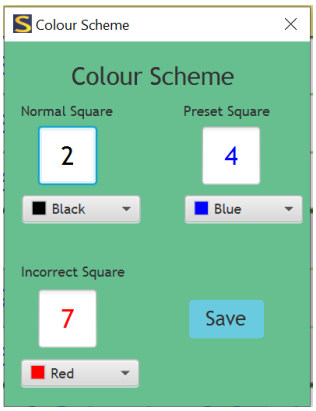
Generate Puzzles

There are three types of puzzles, normal, X-Sudoku, KingSudoku, with three difficulties, easy, medium, and hard. When GenerateNewPuzzle is hit, the puzzle will be created. In the puzzle created, the numbers will be loaded onto the grid, and given the preset property.



Menu System

New opens a new instance of the app and Quit closes the current instance. About Sudoku++ tells you about Sudoku++. Open opens up a Sudoku file(.sdk) and save saves the current Sudoku file. If the file has not been created, save as will be invoked instead. Lastly, the colour scheme allows the user to determine the colours of normal squares, conflicting squares, and wrong squares. The colour scheme function is implemented using ColorPicker.



Autocomplete

This solves the puzzle current puzzle. If no solutions are found, the app will tell you so. If there are multiple solutions, the "lexicographically smallest" sudoku will be returned. That is, the solution with the smallest number appearing in the earlier squares.

Variants

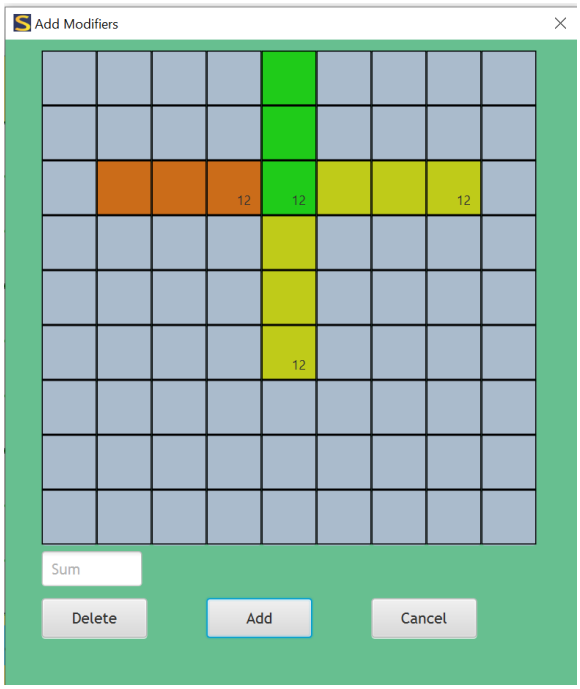
The mode of the grid are controlled by RadioButtons which are contained in a toggle group to allow the creator to change the rules of the puzzle.

Puzzle Info

Basic puzzle info are author, date, source, description, and comment. These fields can be edited when the button Edit Info is hit, a window with fields will pop-up for the user to use. When save is hit on that window, the info will be automatically updated on the screen.

Add Modifiers

Once clicked, it pops up a screen, with 9 rows of 9 buttons, much like the Sudoku grids. The buttons light up based on the current modifiers on the board, currently being only Killer boxes. Killer boxes group up certain squares, to sum up to a specified total. When a set of buttons are connected, the add button groups them up to form a sum, the delete button clears all the buttons from their groups, and the cancel button resets to the state after the last delete or add button was hit. A safeguard is in place to prevent the boxes from being "disconnected", that is the squares in the same Killer box are disjoint. When Killer boxes are added/deleted, it is reflected on the Sudoku grid, with the sum displayed on the bottom right of the group of boxes. However, if there is a Killer box in the grid, the autocomplete function will stop working, due to the algorithms not in place to solve it.



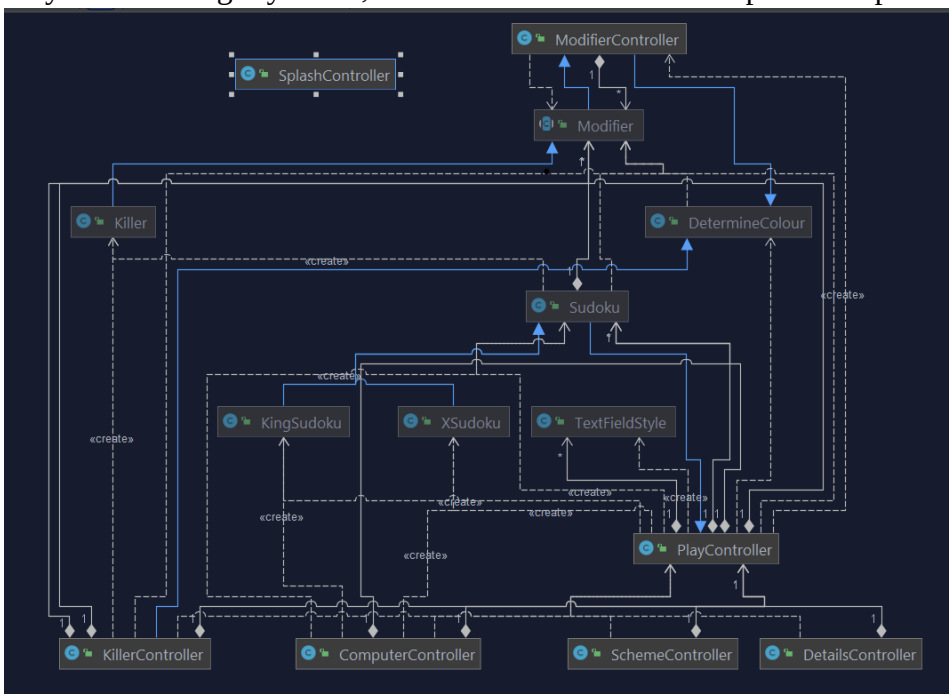
Key Code Features

Code Structure

MVC structure was used, where everything using JavaFX was placed in Controller, the inner code in Model and the FXML files in view. Even though some files in controller are not tied to FXML, however since they used the JavaFX API, therefore it was appropriate to place them in controller instead of Model. There are some CSS files, which have been placed in src folder so that they will work.

OOP Design

All Sudoku puzzles are modelled as objects and are the cornerstone of the project. Additionally, each Killer box is also modelled as an object. Sudoku variant puzzles extend from the parent Sudoku class and the Killer extends from the abstract modifiers class. This makes it easier for more such variants to be added in the future. Since from variant to variant only the rules slightly differ, thus inheritance was used to prevent repetition.



Key Algorithms

Some of the algorithms in Sudoku class use recursive backtracking to generate/solve the puzzle. They work by creating and pruning the possible solution tree, by immediately cutting all searches if the previous fill is invalid.

In colouring the killer boxes, a variant of depth-first search is used to determine the colour. For safety, six colours were reserved, however, in practice, the four-colour theorem should hold true, and only four colours should be required.

Variants

Variants extend from the original Sudoku class, with minor changes with their rules and toString, mostly inheriting from the parent class.

Modifiers

Modifiers extend from an abstact class, Modifier, so that all of them can be contained within a same list in a puzzle.

Testing

Mostly manual testing was used. For each function, different ways to try to break it were attempted. If any bug is found, the bug would be resolved and then tested again. Additionally, others were allowed to attempt to cause crashes and break the app. For some classes, tester classes were used, to specifically test out cornerstones of the project in isolation. For example, tester classes were used in testing the Sudoku class. These testers are removed in the final source code for easier reading. For testing the correctness of the app, some online

Sudokus were imported by hand, then solved by the program, and compared against the solutions given online.
For example, a test scenario is with the Sudoku grid in Main App. Due to the differences in triggers for onKeyPressed and onKeyTyped, the arrow keys break the program, and deletes the number, when it should not. To fix this bug, a catch was put in onKeyPressed, to prevent such operations in the future.
File tests were done, by sending the .sdk files over to friends, along with the JAR file, to see if the formatting works.

Reflections

Challenges

This project was technically challenging. Unlike other CS projects, there was no clear direction to proceed. In this respect, I was able to implement more complex algorithms, such as those in the sudoku methods. As a result of these algorithms, many bugs arose and it took some time to debug, as a result I learnt of the common pitfalls and not to make them again.
An obstacle faced was the kinks of JavaFX, the styling elements to make the Sudoku grid look like Sudoku grids. In the project there is a TextField grid and Button grid, both created with different css style elements.

Things Learnt

With this new knowledge, future such JavaFX projects involving these elements would be easier. As an example, crossword puzzles use similar functionalities and these skills can be applied. Previously, I implemented algorithms in C++, however I now know how to convert them to their Java counterparts.

Future Work

Crucially, the app is currently missing the usual function of online sudoku solvers, a pencil function. With more time, this feature could definitely be improved. Currently, I feel my code is rather unclean and will become hard to maintain in the future. So, the main thing I would do after this is do a code clean-up, to make the functions consistent. However, the structure of the code is made so that future implementations will not require a major overhaul. To me, this is a project that I will continue to work on in the future, possibly with a friend. In the future, I hope to send this out to friends for their usage.

Improvements to task

The project task set was enjoyable but could have been made better with more milestone checks, to check the progress of people.