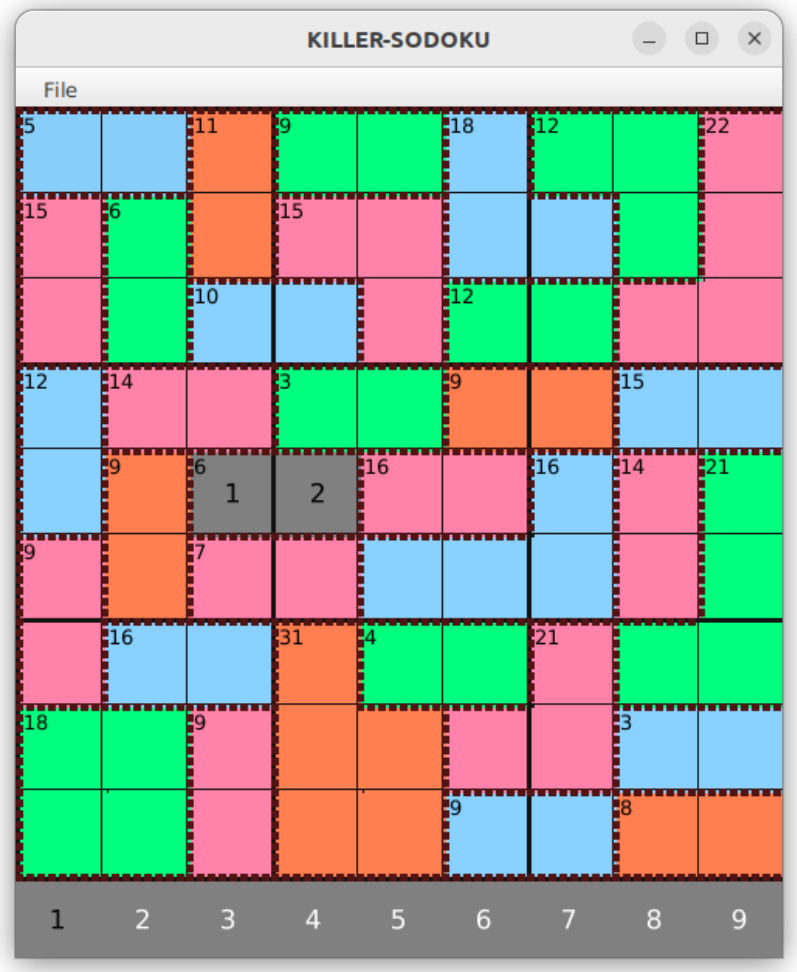
KILLER-SODOKU:



1.Personal information.

Student's Name : Quan Hoang
Student Number: 100566965
Degree program: Quantum Technology 2025.

2.General description:

Sudokus are popular reasoning tasks that are constantly published in many newspapers, magazines and in separate Sudoku magazines. The basic version of Sudoku has a 9*9 grid, in each square one of the numbers 1-9 must be placed so that a) in each row this number appears only once, b) correspondingly in each column the number appears only once and 3) in each 3*3 sub-grid number occurs only once.

Killer Sudoku differs from basic sudoku in that it does not give any ready-made numbers in the grid. Instead, the grid has different sub-areas formed by 2-4 squares and only the sum of the numbers in the area is given; in other words, a sub-area can contain several different combinations of numbers. Otherwise, the solution requirements for placing numbers in a 9*9 grids are the same as in basic sudoku. The shape and placement of the sub-areas in the grid is free.

Solving easy sudokus on paper by writing numbers directly in the squares is quite simple. However, killer sudokus require so much accounting that recording and updating them on paper is often tedious. The task is to build a program that makes it easier for the user to solve killer sudokus when accounting parts are automated. The program is also able to solve the puzzle immediately.

3.User Interface:

- The program offers a graphical user interface that shows the sudoku grid and sub-areas
separated by outlines, and in addition, each number 1-9 is listed below in separate boxes, which are here called candidate numbers.
- When the user moves the cursor over the open square of the grid, the program highlights the candidate numbers that can possibly fit into this square according to the basic sudoku rules, i.e. whether the number appears in the same row, column or part of the grid. The program does not do any more complex reasoning.
- In addition to the previous one, the program somehow shows the number combinations
that can currently be placed in the corresponding sub-area of the killer sudoku Corresponding.
- If the user clicks on square, he can also place one of the candidate numbers in this box. In this case, the number accounting of the remaining options in the sub-area is updated at the same time.
    - For example, if the sum of the numbers in the subarea of three squares is 15 and the user places the number 7 in one square, the remaining empty squares can have combinations (2+6 or 3+5).
- The user can remove a number from the grid if one realizes that there was a mistake, and the accounting will be updated accordingly.
- When the user moves the cursor over a candidate number without clicking it, the program highlights the squares in the grid (e.g. with a background color) that already have the number in question.
- When all numbers in a specific area are filled ,the color in these areas will be turned Gray.
- You can create a new sudoku problem by reading the sudoku problem definition from a File.
- The solution of a sudoku problem in progress can be saved to a file and read from there again.
- Represent the sub-areas with different colors so that no two adjacent sub-areas have the same color. However, angled areas can have the same color.
- The User can revert an action (undo) by pressing Z, or do the opposite by pressing Y.
- The program can instantly

4.Program structure:

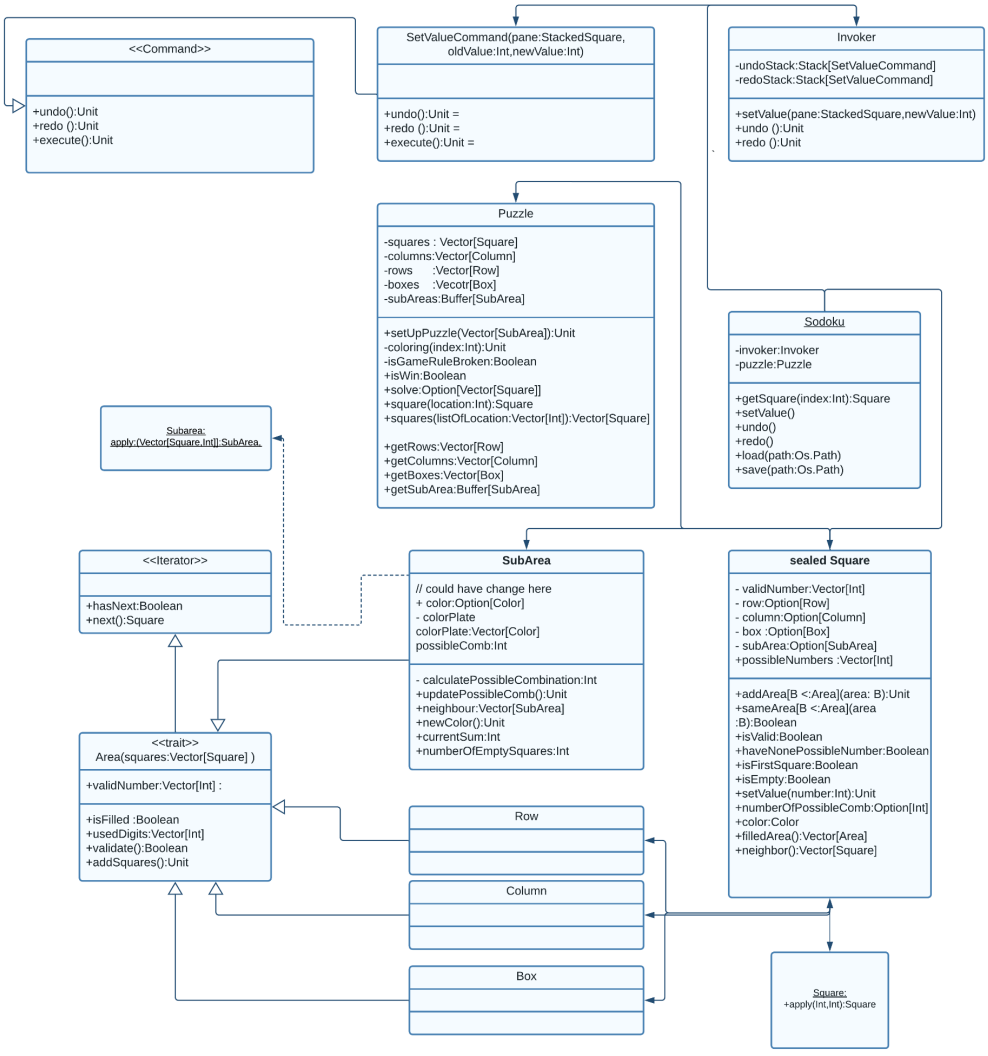
Based on the project specification, the problems domain of this project can be divided into two main sub-domain: designing user interface, and data management. Since sudoku is a

constraint problem, the program domain then can be extended into the domains of puzzle-solving algorithm, puzzle - generation, and optimization. Thus, after thoroughly examining the problems domain of the project as well as taking extensibility into consideration, I divide the program into three packages :logic, gui, and filemanagement.

4.1.logic

Suggested by its name, the logic package will represent the sudoku puzzle as well as control and advance the Game. Since the objectives for the logic package are clear, a top down implementation is suitable in this situation

Overall, there will be a group of classes representing a puzzle; a game object accompanied with a command invoker will then be used as a communication link between the and other parts of the program. (The UML diagram of the logic package can be found below)



Each puzzle class will represent a sudoku puzzle by dividing it into smaller <<Area>>(row ,boxes ,and subArea) and eventually to a Square which is the smallest element of a puzzle.

While A sudoku puzzle can be well defined by using only an Array of Integer with each number representing a square in the puzzle, this is obviously a bad idea since it profoundly reduces the reliability as well as extendability of the program. Thus, it is my initiative to use class Square instead of an integer to represent a square. The puzzle class then will use an immutable collection that holds a number of a square. Right now, the programme will use a magic number 81 to create this collection as it currently only supports 9x9 sudoku. However, they can be refactored with ease to extend into a generic sudoku. The puzzle class also has several variables to host the Area of a puzzle. The trait Area was used to help define row, column, boxes, and subArea since they have similar functionality especially for the first three classes (row, column, and boxes) which are interchangeable with only exception is subArea which require an companion object, and noticeably more methods, since its represent a cage in killer Sudoku-the defining characteristic of the game.

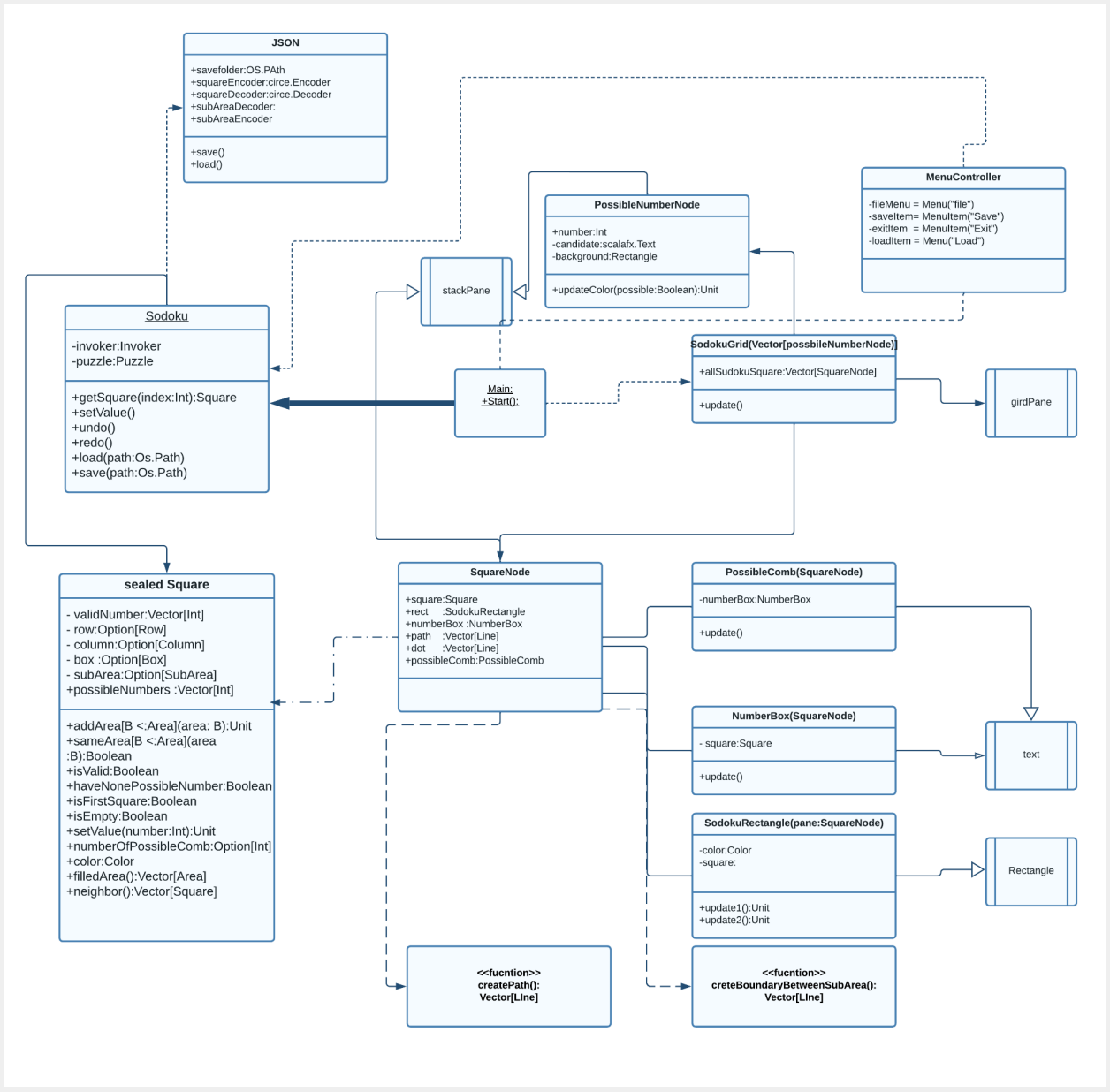
(Note: Trait Area will be extended with Interator, which currency provide no substantial extra benefit (currently it only provide syntactic benefit), but could be useful when trying to expanding the program to other domain)

After defining the most important part of the program, a game object Sudoku and its companion object invoker will be used to control the program. As mentioned before, the GUI and logic of the program will communicate via these two objects. The GUI should change the state of the program using methods of Sudoku and Invoker. The implementation of Sudoku is trivial since all the hard work is done in the implementation of Puzzle and related classes. To implement Invoker, the command design pattern is employed to carry out the undo and redo interface. Class SetValue Command is also implemented to achieve this functionality.

4.2.gui

Top down implementation is once again employed to create the Graphical User Interface of the program. This project will use the Scalafx library which is intuitive to work with and has many development tools that are suitable for the project.

In detail, a 9x11 grid pane will be used to represent a sudoku puzzle. Dua to some limitations with the base layout classes of the Scalafx, they were extended to better fit with the program logic, and dynamic property of the GUI. The structure of the GUI can be briefly summarize below:



Each instance of the SquareNode class ,which is an extension of a stackpane, will be linked to a Square Object. A Square node will have several components stacked together, each of them represents a functionality in the user interface:

- SudokuRectangle(pane:SquareNode) extends Rectangle :
  A rectangle whose base color is calculated via coloring and will update color base on the program state .That is ,when the player hovers into its parent SquareNode , the program will call method update2() and darken the color. Moreover , the all square in the same Area isFilled , the color will be turned to Gray
- NumberBox (pane:SquareNode) extends Text:
  The NumberBox will hander events from keyBoard.Given the input is valid the program will call the update() methods in this class

- PossibleComb(pane :SquareNode) extends Text:
  Represent number of Possible Combinations for the Subarea this square belongs to.
- <<Function>> : createBoundaryBetweenSubArea , and createPath : always return 4 lines ,whose visibility and other property(StrokeWidth,.....) depends on the subArea of the square ,and its location.

While implementing these classes, several options were considered. Since most of the classes have update(), It sounds reasonable to group them into a trait or an abstract class. However, dua to the great distinction between the functionalities of these classes, such "Updateable trait" is redundant. Separating Sudoku Rectangle into smaller part also result in higher cohesion to the program, and this explains why createPath() , createBoundaryBetweenSubArea, NumberBox ,and PossibleComb aren't combined together even though they have many similarities.

After having created the component, we need to combine them together and the ScalaFx library provides two suitable tools for this purpose: StackPane, and Canvas. Although they are both viable choices, the former was chosen dua to its simplicity.

Another kind of Node is PossibleNumberNode which is also extended from stackpane. Using similar ideas with SudokuNode, each PossibleNumberNode represents a digit (from 1-9),and uses a rectangle as background. The color of the node will change as specified in the user interface.

The class SudokuGrid is used to host all of the above nodes. This class has methods update(), which will erase all the current nodes and create new ones that align with current data.

The MenuController will handle all functionality concerning the file. The class will communicate with the JSON object(will soon be discussed) via Sudoku Object, and have menu elements representing aforementioned functionality (save, load ……)

4.3.IO.

Os.lib and circe will be used to implement JSON objects which encode and decode the program. Circe is a modern library that can automatically parse and encode json files, and together with Os.lib , works with json files can be done with ease.However , there is one condition for circe to automatically do its job and that class which we want to parse or encode need a companion object. Thus two objects with the apply method Square and SubArea were created.

5.Algorithms.

We will mainly deal with the Constraint satisfaction problems (CSPs) which can be effectively solved using a form search algorithm .Taking the number of constraints and problem size into account ,I will use brute force algorithm will be used to calculate the number of possible combinations ,and backtracking will be used to solve the puzzle as well as mapping to color for the GUI.

When trying to calculate possible combinations for each sub area ,because the problem size is small (subarea typically have 2-4 squares) and the constraint is simple (total sum is equal to a number), brute force is as effective as other algorithms in terms of speed . Combining this with the fact that brute force is simple to implement , Brute force is a decent choice in this situation.

On the other hand ,since number constraints and the problem size of puzzle solver and mapping coloring is significantly higher than when trying to calculate possible combinations ,

brute force is no longer applicable. Backtracking is suitable for these two problems for two reasons. Firstly, not only does backtracking have decent runtime, but it is also easy to implement and fit well with the defined class structure. Secondly, deterministic property of backtracking algorithm allows the program to detect corrupted data files as well as extend the solving algorithm to a error detector (the program can return incorrect square ,if the solver don't return any value)

In detail, when trying to mapping coloring, the program will start with a SubArea and choose a color. After that it will remove the previously chosen color out of the color palette of the neighboring area. The program will recursively do this process, and if it reaches an area with no color left in the palate, it will go back to the last area and choose a new color.

Similarly, when trying to solve the sudoku, the program will greedily set a new value of a square, and will go back if it reaches a square which can't fit any number. Since the problem size is quite large, the algorithm will be aided with several auxiliary methods defined in the related classes.

6.DataStructure:

On the first glance mutable collections are needed to implement Puzzle class because the program needs some means to update data based on the input of the users. However, dua to the fact that the number of row, column, box , and square objects won't change during the execution of the program and they have their own means to update data, Puzzle will use Vector to store instances of these classes. In contrast, a buffer is used to host all SubAreas.

Concerning Area trait which extends from Iterator, each of them will take a Vector[Square] as parameter which will be used to define the iterator methods. Combining the Square iterator with A Vector that stores Integer value from zero to nine, the area has the data it needs to to implement methods such as validate() which detects duplicate numbers .

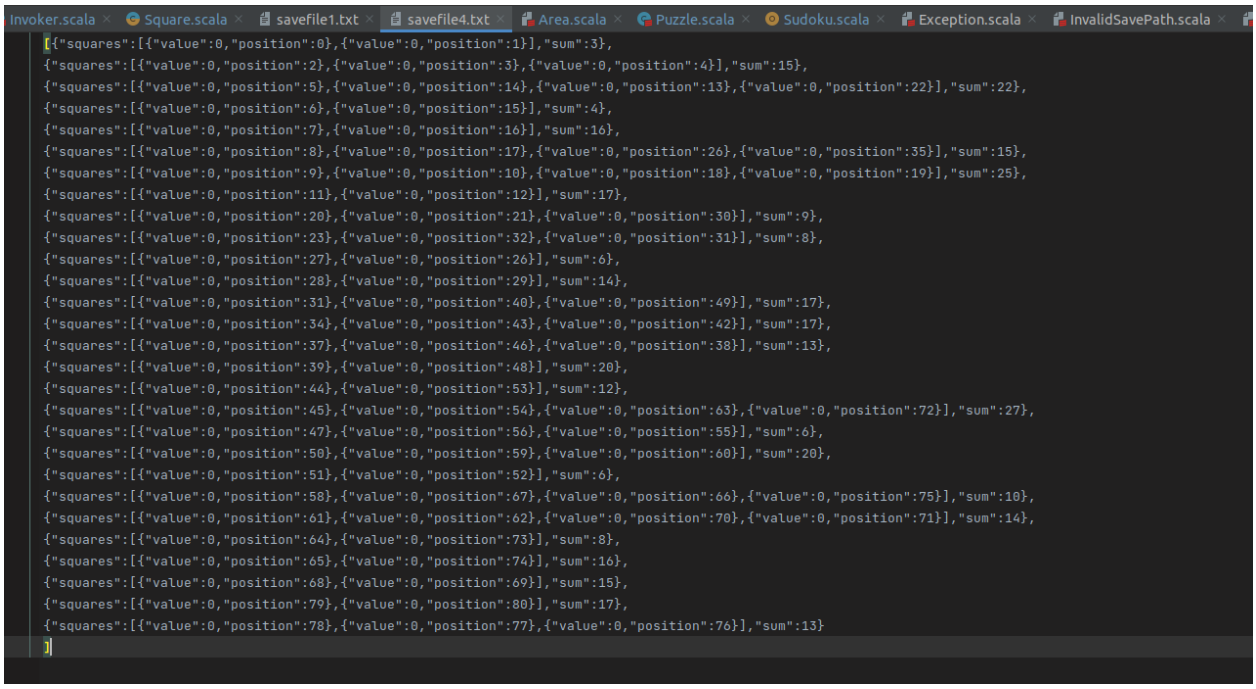
The creation of Row, Column, and Box is trivial since they don't have any extra functionality. SubArea, on the other hand, requires a variable to the color palette as well as another "temporary" variable that holds possible color according to the mapping rule .Because the size of the former is fixed and the latter is a temporary variable, we will use Vector in this situation.

Square is the smallest component in the class structure, and it needs four Option containers to store the information about the area it belongs to. Option is suitable for this purpose because a corrupted save file might result in a null variable which might crash the GUI.

7.File and internet access.

The program doesn't have access to the internet and mainly deals with json files, which is not only human-readable but also conveniently work with modern libraries such as circe.

A save file, of course, flows the rule of typical json file, and it will look like this:



In detail, we will use two automatic json encoders provided by circe to encode the date. The first encoder turns a square object to a json string with the information about its current value of position:

`{"value":0,"position":78}`

The second encoder will turn instances of SubArea into a json string. In detail it will use the first encode to turn the Vector of Square into a json string, for instance:

`"squares":[{"value":0,"position":78},{"value":0,"position":77},{"value":0,"position":76}]`

After that it only combine this with variable sum which result in:

`{"squares":[{"value":0,"position":78},{"value":0,"position":77},{"value":0,"position":76}],"sum":13}`

To make the creation of a custom file more convenient, the JSON object will print out its error message to help the user detect the error in the file.

8.Testing plan:

Taking Teaching assistant's advice seriously, the program is tested regularly after a new feature is implemented. In most cases the program is tested via both print function and graphic user interface.

However this strategy soon becomes less effective as more and more features are added to the program which make dependency between classes unavoidable. A bug in one component could radically affect the overall functionality. Thus, I have created a test for the data structure to detect such changes.

In detail, the test class will first include test case auxiliary methods or variables that are used to build other important ones. If these methods or variables doesn't work as intended ,there is no point to look for bugs in other part of the program:

- A Square should belong to 4 types of Area: Row, Columns, Box, and SubArea.
- SetValue, and Update possible value must work as intended otherwise the GUI and the solve methods won't work.
- Number of possible Combinations should return the correct number according to the verified data.

After checking the auxiliary methods works correctly, I add some println commands into the key methods and open the GUI. Using this strategy I will not only test both of them at the same time but also enable me to detect the situation where the GUI fails to fetch data from the class structure. With this strategy, I have tested all the functionality and the related methods and they all yield positive results.

To test the file management scheme, I first wrote all small test to check if the program can handle error with "invalidsavefile.txt". After that, I wrote more tests to make sure that the program can deal with the situation where the user tries to save into source code or cancel the selection process. Finally, I test if the program can actually save and load a file by setting some squares into new values and saving before loading to another file and loading back. Nothing unusual happened, and I conclude the file management works as intended.

9.Known bugs and missing feature:

Although there are no known bugs or missing features, there is a small modification so that instead of changing color when all instances of number are filled into the area then put all squares passive, the program will change color when an area is filled and pop a message when the player wins.

10. Three best sides and three weaknesses.

Not sure about this, but I think the project has a good program structure. The project was divided into different packages and they communicate well together with a well defined interface: Sudoku Object. The Classes and its Methods are created in the ways that not only they result in a decent coherent but make room for meaningful abstraction, and Polymorphism (The methods solve and Puzzle class could be an example for this). Although I am not sure about how success my attempts to encapsulate the internal behavior in classes are, they aren't made without rationality (You can see my explanation in code comment)

Having a good program structure enables me to add more meaningful functionality to the program. Beside the requested feature, the project is extended to include a sudoku solver ,which could be extended to aid the user to detect errors. Moverover ,the user experience is also improved by the undo and redo methods. Both of two features above are implemented without any scarification which act as a proof for good extensibility of the class structure.

The last thing that I consider to be a good point is the Data Structure and Algorithm. In Terms of runtime, the chosen Algorithm is fast enough to do needed computation ,and this can be shown during the execution of the program. The Data Structure is also chosen with rationale ,and It helps facilitate the implementation of needed algorithms.

On the other hand, I think I should have done better with the testing plan. Not many proper test units are implemented, as most of them are tested via println and the GUI. Given the

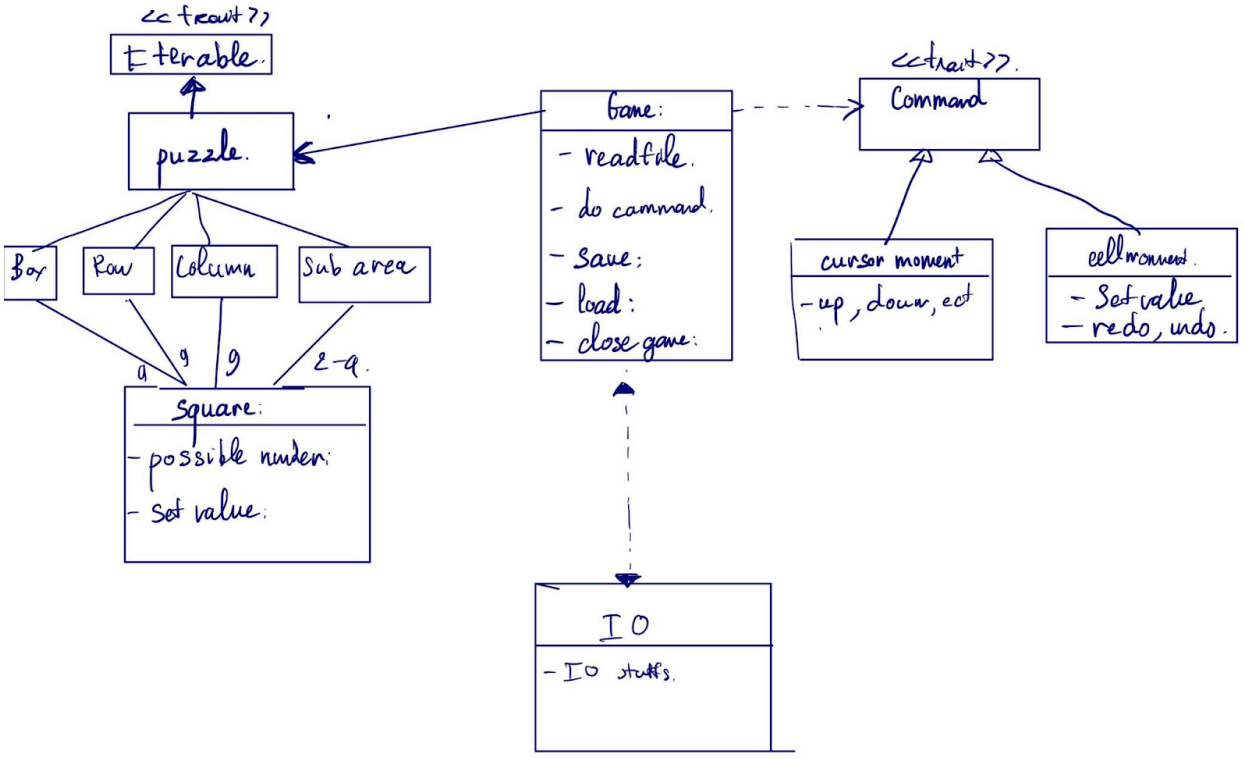
opportunity to do it again, I would definitely put more effort and focus to test out things rather than rely on the “as long as it works” mindset. I learned this during the implementation of the calculation number of combinations of a subArea. If I had a proper test for the GUI, I could have known better the problem is not about the methods in the class structure but in the GUI itself.

Another area for improvement is how the program handles errors .It might be not too late to rectify this as I think most of the critical situation where error might arrive is handler , but in many case errors are currently handled aren’t necessary meaningful .By the time ,I realize the problem, it requires substantially more effort than it could do if I have put more attention to errors during the development process.

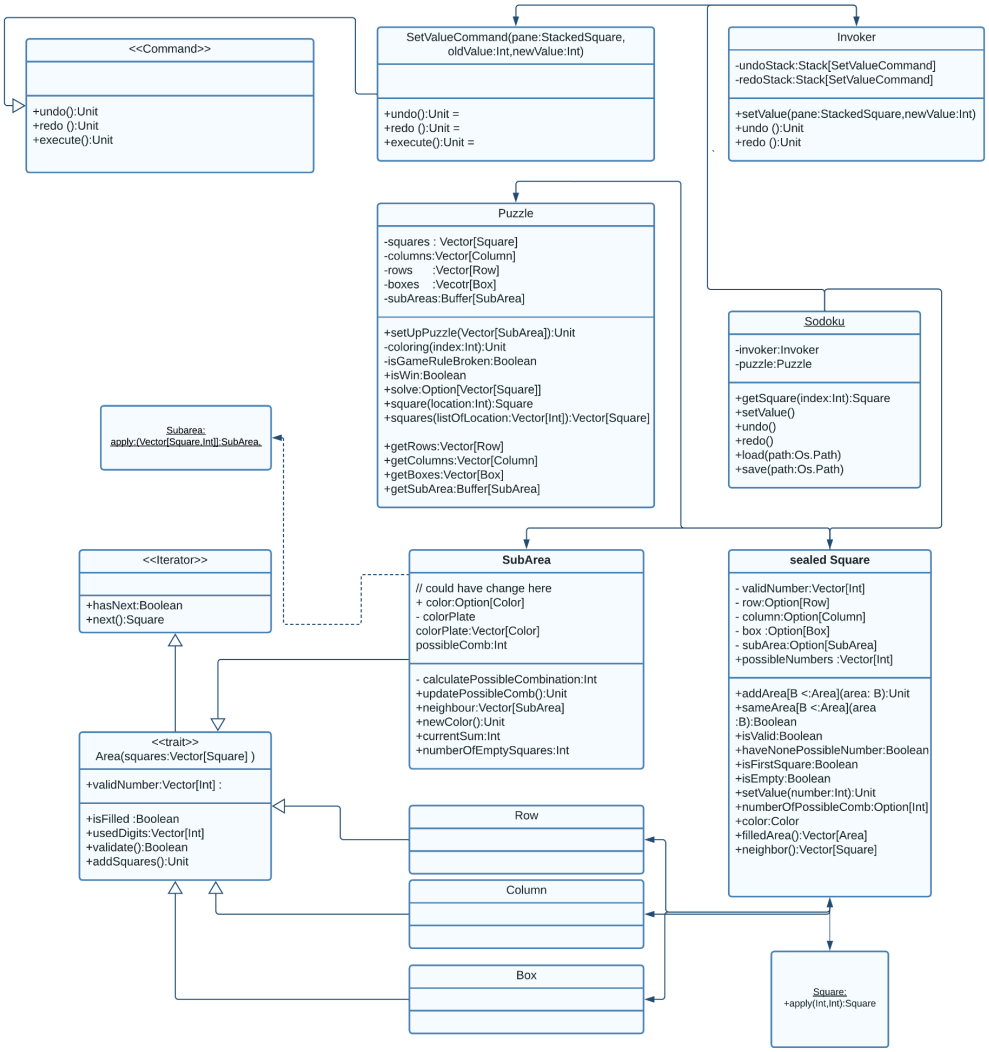
Finally, While the GUI generally functions well, I think it could achieve a better GUI with less effort and better quality of code. I know this sounds vague , but back then I didn't know “what could be done ?”, “What is possible ?”. For instance, I could integrate the pop-up feature to respond to errors which is actually a solution to the weakness I have just mentioned. In short, I should spend more time researching before beginning coding.

11.Deviation from original plan ,realized process and schedule:

Because this project employs top down design strategy, there is evidently a lot of difference in the original UML diagram and the actual final UML even though the ideas remain unchanged.Below is the tentative UML in the project plan:



And here is the final UML diagram:



The biggest differences in file management schemes. Initially, the program is planned to work with its own defined file format. However, the availability of tools that work with popular file formats such that json convinced me to work with one of them instead.

12.Final evaluation:

I think my project is good enough for at least four. There is no missing feature.TA said that my program structure ,functionality .... are good.

However, If I have to do this again, I will keep the logic package and spend more time planning. The biggest lesson I have learned about this course is that it is all about the ideas and the math. Try to figure them out first before starting coding.

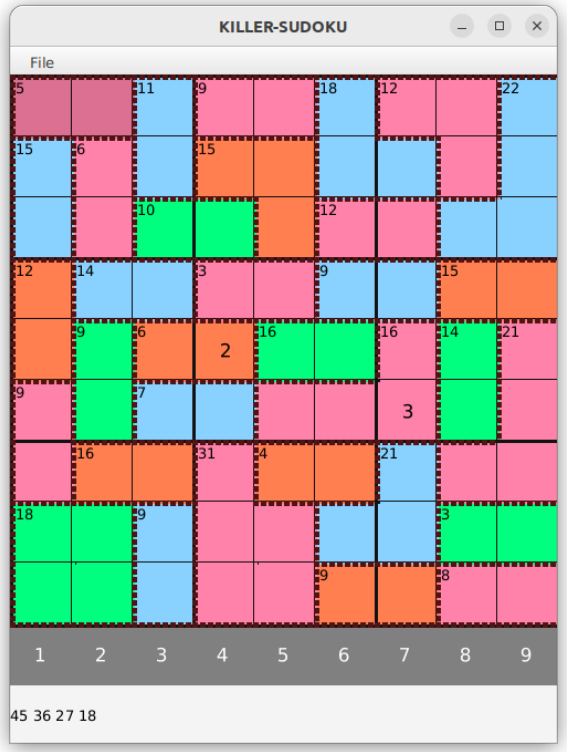
13.References:

1. Mark Lewis's series about Scalafx: ScalaFX First GUI
2. Lamda Town circe : Working with JSON in Scala, a Circe Crash Course
3. Os.lib API : https://github.com/com-lihaoyi/os-lib
4. Another blog about Os.lib : https://www.lihaoyi.com/post/HowtoworkwithFilesinScala.html
5. Introduction to Scalafx : https://www.scalafx.org/docs/home/

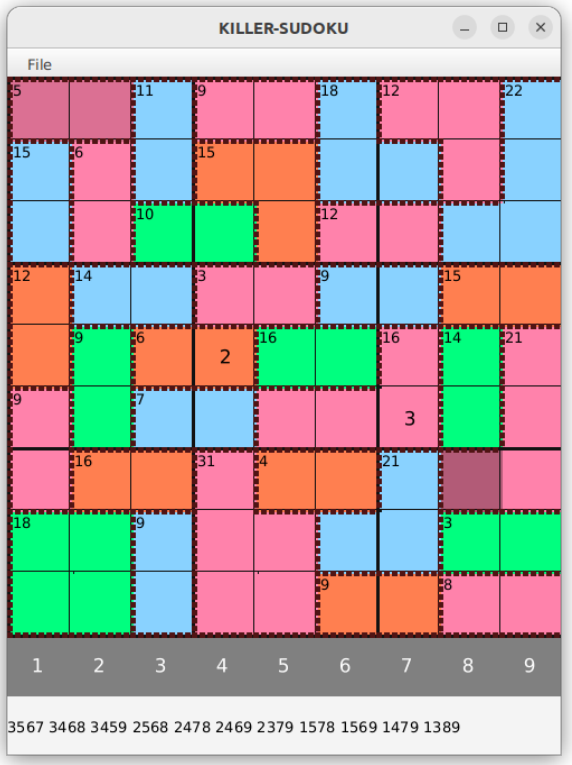
6. Scalafx API : https://javadoc.io/doc/org.scalafx/scalafx_2.13
7. JVM sudoku solver :https://www.youtube.com/watch?v=zBLCbqycVzw
8. Programming 2 Sudoku assignment : https://plus.cs.aalto.fi/cs-a1120/2023/recursion/recursion-sudoku/
9. Wiki page of killer Sudoku : https://en.wikipedia.org/wiki/Killer_sudoku
10. Wiki page for constraining satisfaction problem : https://en.wikipedia.org/wiki/Constraint_satisfaction_problem
11. Programing studio A course : https://plus.cs.aalto.fi/studio_2/k2023/
12. Page concerning variable's functionality :https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/The_Role_of_Variables
13. StackOverflow:https://stackoverflow.com/questions/46997267/how-do-i-insert-text-into-a-shape-in-javafx

14.Appendixes:

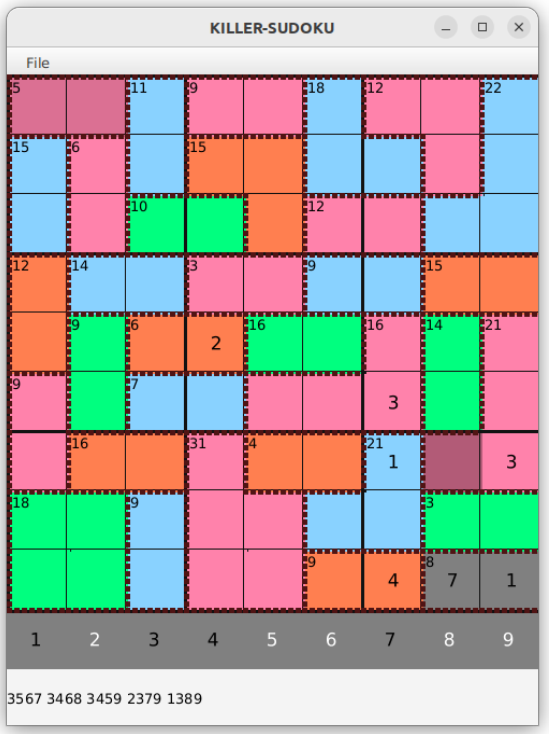
The GUI will look like this when you start the app:



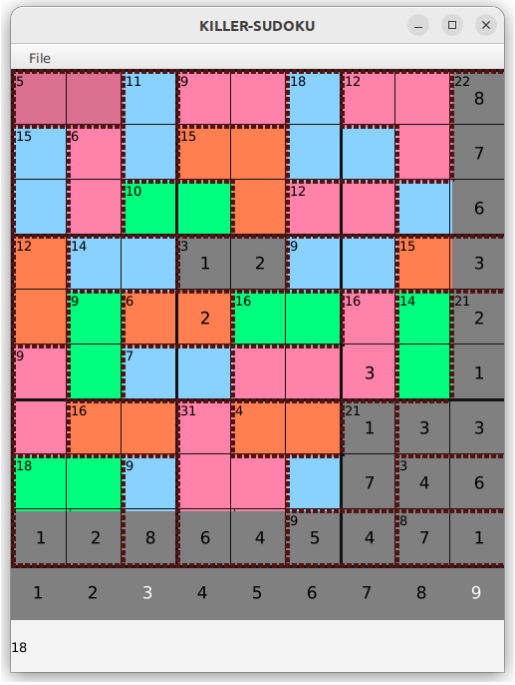
Hover to a square to see the possible combinations that can fit in this square. You can also notice that the number in the gray bar has two colors: black and white. The color white indicate that putting this number into the currently hovered square won't violate classical sudoku rules:



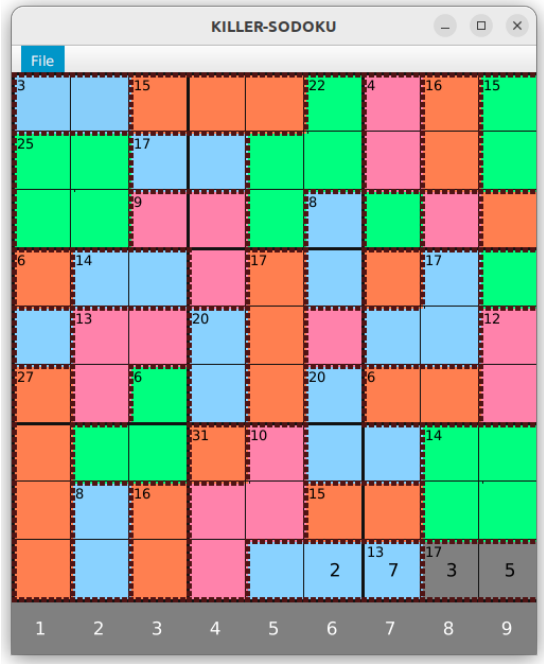
Let's try to put some numbers in nearby areas. Notice how the GUI have changed:

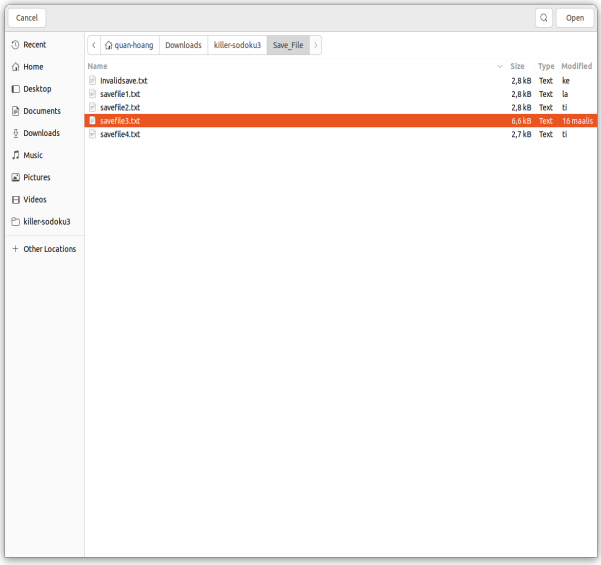


When you filled all squares in an area, that area will change color to gray:

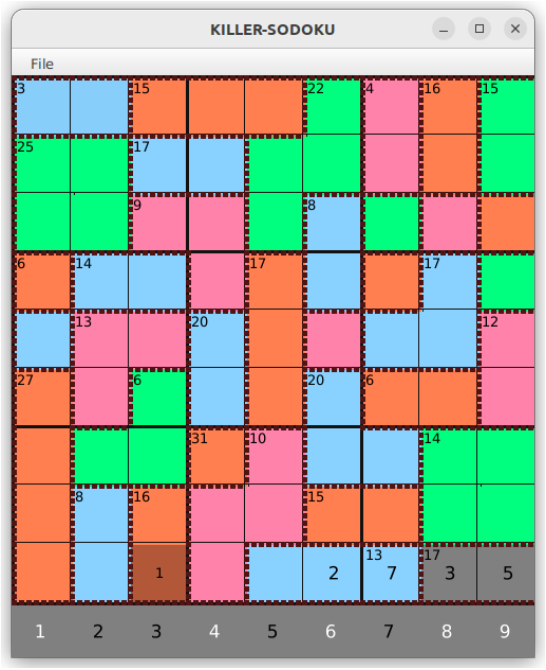


To load a new file ,click in the file menu and choose load.





This result in new non empty puzzle:



Finally you can solve the sudoku by click to solve in file menu: