Assignment 3

Assignment 3 is a follow-up task over assignments 1 and 2. While assignments 1 and 2 considered command-line versions of Sudoku, i.e., a game with a CLI (Command-Line Interface), assignment 3 will consist of a new version of your game, with a GUI (Graphical User Interface).

The implementation you will use as the starting point is your own implementation of *GTIIT Sudoku v2*, consisting of the four following classes:

- `SudokuCell`: it is the simplest of the classes. It models a cell of the Sudoku, with its possible states: fixed or not fixed, and its contents (value), with zero representing the cell being empty.
- `SudokuBoard`: it models the board through a list of rows, each row being a list of SudokuCell objects. It contains methods to create a board from a file, setting a value in a cell, checking if the board is solved, solving the game, etc.
- `SudokuInputReader`: it implements the reading of input commands from the user.
- `SudokuMain`: the main class of the game.

Your task consists of a single implementation stage:

The new project does not have your implementations in version 2. Incorporate your implementations of assignment 2, and make sure the project works as expected. You may perform minor changes to your implementation, e.g., if you found issues such as bug fixes or some refactoring. Do not remove any of the classes of assignment 2. In particular, the front-end classes (`SudokuInputReader` and `SudokuMain`) have to be preserved.

The actual new implementation starts here. You have to implement a graphical user interface for GTIIT Sudoku. The only constraints of the implementation are the following:

- Your implementation must contain a class named `SudokuGUIMain`, with a `main` method. This class and its main method will be the launching point of the graphical version of Sudoku, i.e., by executing the `main` method, the Sudoku game with its graphical user interface is launched.
- Your implementation must not further modify the backend classes, i.e., classes `SudokuCell` and `SudokuBoard`. The graphical user interface must delegate the logic of the game to these two classes.
- Your graphical user interface must allow one to interact with the game with the same functionalities that the existing command line interface offered: starting a game, setting a (non-fixed) cell, solving the game, and quitting.

Given these constraints, your main tasks will be to design the graphical user interface, and connect the graphical user interface components to the backend methods, e.g., connecting the button or menu item that starts a new game with the methods that create a board, etc. You may use any graphical user interface elements, the design is your choice. You may decide to interact via dialogs, buttons, menus, etc. You may decide to show the board as a grid of text fields, or a grid of buttons, or simply as a text area.

Submission will be via Github, as in assignment 2.

You may decide to add auxiliary classes or methods. You are not allowed to change the APIs of the already provided classes, i.e., not change method names, number of parameters or their types. You must not modify other existing methods in the implementation.

This assignment must be solved in groups of two. The correctness of your solution is important, as so is the quality of the code (clarity, readability, structure, etc.), and the quality of the graphical user interface. Ease of use as well as aesthetic aspects of the interface will be taken into account.