Project Plan

Analysing the problem

In our pursuit to aid individuals grappling with Sudoku challenges, we envision a versatile program capable of generating and solving Sudoku puzzles. Our objective is to craft an interactive assistant program, capable of providing users with step-by-step hints for navigating the intricacies of Sudoku problem-solving.

Accessible to Sudoku enthusiasts and those new to the sport, our program will cater to varying difficulty levels, ensuring a gratifying experience for all. Users confronted with Sudoku puzzles from magazines or puzzle books will be able to upload their puzzles, offering assistance on their problem-solving journey even outside the program.

Ethically unrestricted, our program focuses solely on puzzle resolution. Time permitting, we aspire to introduce various user-friendly additions, such as a night mode. Moreover, our program could go beyond merely assisting users for traditional sudoku's and assist with bigger and more complicated variants, such as diagonal and mega sudoku's.

Drawing inspiration from 'Sudoku Wiki,' a similar online platform, we aim to distinguish our approach by providing more comprehensive explanations for each step, accommodating both seasoned players and newcomers. For our program, we plan to translate existing Sudoku-solving algorithms into Python code, relying on insights from 'Sudoku Wiki' and other related online articles.

The final goal for our program is in ways comparable to hint-providing tools already existing within various sudoku apps. Such existing tools often provide answers for certain cells or highlight which rows or columns to focus on, when the user feels stuck. Our program will separate itself from such tools as it will provide more of an educational experience. Teaching the user what algorithms to utilize in certain situations.

Requirements

Naming convention requirements:
All requirements given below have a unique id-code. The first part 'REQ' signals to the developer that this is a requirement id. The next number ranging from 01 to 04 signifies the importance of the requirement within the MoSCoW method, 01 indicating a must have requirement, 04 indicating a won't have requirement. The id-code is completed with a float number. Two requirements with the same number before the decimal point are strongly related to each other, given that the requirements have the same importance category. For example REQ-02-2.2 is a should have requirement which is strongly related to REQ-02-2.1.

In the requirements the term ‘legal sudoku’ refers to a sudoku that does not contain any mistakes according to the ruleset of that type of sudoku and a ‘legal sudoku’ must have a single solution.

Must haves:

REQ-01-1.1 The program must be able to calculate the solution of a 9x9 square sudoku if the sudoku has one single solution.
REQ-01-1.2 The solution of the sudoku is clearly shown in the UI by pressing a button in the UI.

REQ-01-2.1 In the UI an initially empty 9x9 sudoku is shown.
REQ-01-2.2 The user must be able to fill cells in the sudoku displayed in the UI with numbers that are within the allowable number range for a 9x9 sudoku.
REQ-01-2.3 After filling in the desired cells the user must be able to press a button upon which the sudoku is uploaded to the program.
REQ-01-2.4 When a sudoku is uploaded the user can use the functionality of the solution, mistake and hint buttons (REQ-01-1.2, REQ-01-3.2 and REQ-01-4.2 respectively).

REQ-01-3.1 The software must be able to locate any mistake in a partially finished sudoku.
REQ-01-3.2 In the UI the user can press a button which checks whether any mistakes have been made (REQ-01-3.1).
REQ-01-3.3 The amount of mistakes is clearly shown to the user upon pressing the button described in REQ-01-3.2.
REQ-01-3.4 The user is notified if any mistakes have been made if they click on the hint button (REQ-01-4.2) or the solution button (REQ-01-1.2).
REQ-01-3.5 When a mistake has been made and the user has pressed the hint button (REQ-01-4.2) the user will not be provided with a hint.
REQ-01-3.6 When a mistake has been made and the user has pressed the solution button (REQ-01-1.2) the solution will not be shown to the user
REQ-01-3.7 When a mistake has been made and the user has pressed the solution button (REQ-01-1.2) the user will be provided the option to show the solution anyway.
REQ-01-3.8 When the user is notified that one or several mistakes have been made (REQ-01-3.3) the user can press a button which clearly shows the cells of all mistakes.

REQ-01-4.1 Given an uploaded sudoku the program must be able to identify what cell can easily be filled in using the ‘last remaining cell in a box, row or column’-algorithms.
REQ-01-4.2 In the UI the user must be able to click on a button after which the user is given a hint as described in REQ-01-4.3.
REQ-01-4.3 The program must clearly provide either the row, column or box of the cell described in REQ-01-4.1 when the hint button (REQ-01-4.2) is pressed.
REQ-01-4.4 The user will be given the cell described in REQ-01-4.1 if they press the hint button (REQ-01-4.2) again.

REQ-01-5.1 The following algorithms for finding hints should be used to find a cell to fill in if the previously implemented algorithms fail to find a cell:

- **REQ-02-5.1.1** Naked Pairs/Triples
- **REQ-02-5.1.2** Hidden Pairs/Triples

REQ-01-5.1.3 Naked/Hidden Quads
REQ-01-5.1.4 Pointing Pairs
REQ-01-5.1.5 Box/Line Reduction

REQ-01-5.2 All algorithms mentioned in REQ-01-5.1 must work for all implemented sizes of square sudokus (9x9, 16x16, 4x4).
REQ-01-5.3 If any of the algorithms mentioned in REQ-01-5.1 find a cell to fill in the user must be notified what algorithm to use to find that cell.

Should haves:

REQ-02-1.1 The software must be able to solve square sudokus of sizes 16x16 and 4x4.
REQ-02-1.2 The software must be able to give the user hints (as described in REQ-01-4) for square sudokus of sizes 16x16 and 4x4.

REQ-02-2.1 When the user uploads a sudoku (REQ-01-2) the program automatically checks if there are any mistakes.
REQ-02-2.2 When the user uploads a sudoku (REQ-01-2) the program automatically checks if there are more than one solutions.
REQ-02-2.3 If the program finds a mistake (REQ-02-2.1) or if there are more than one solutions (REQ-02-2.2) the user is clearly notified which of the two conditions for the legality of the sudoku was violated.
REQ-02-2.4 The user can change and reupload the sudoku after trying to upload an illegal sudoku.

REQ-02-3.1 In the program the user must be able to select a button to upload a new sudoku of sizes 16x16 and 4x4 in the same way as described in REQ-01-2.
REQ-02-3.2 Once the button mentioned in REQ-02-3.1 is pressed the user is shown an empty grid of the selected size where any cells can be filled in with the allowed number range.
REQ-02-3.3 For all sizes the user can upload the sudoku to the program following the same procedure as in REQ-02-2

REQ-02-4.1 In the UI the user must be able to select a button which leads the user to a page displaying information on algorithms the user can use to systematically solve sudokus, containing at least all algorithms implemented for giving hints.

REQ-02-5.1 After a sudoku is uploaded the cells that were not filled in are still accessible to the user to fill in whereas any filled in cells cannot be changed anymore.
REQ-02-5.2 Any value filled in a cell after uploading can be changed at any point.
REQ-02-5.3 When the user presses the hint button (REQ-01-4.2) the program gives the user a hint based on the sudoku in the UI at any stage of completion (not only the uploaded sudoku).

REQ-02-6.1 The software must generate a 9x9 square sudoku which has only a single possible solution.
REQ-02-6.2: The software must be able to generate sudokus with three levels of difficulty, by adding more or less numbers to the beginning state of the sudoku.

REQ-02-6.3 The user must be able to select a difficulty in the UI and generate a sudoku with that difficulty.
REQ-02-6.4 The generated sudoku must be displayed in the UI

Could haves:

REQ-03-1.1 The user must be able to upload a diagonal sudoku of sizes 16x16, 9x9 and 4x4, starting with an empty grid as described in REQ-02-3.
REQ-03-1.2 The software must be able to solve the diagonal sudoku.
REQ-03-1.3 The user must be able to upload a chess sudoku of sizes 16x16, 9x9 and 4x4, starting with an empty grid as described in REQ-02-3.
REQ-03-1.4 The software must be able to solve the chess sudoku.
REQ-03-1.5 The software must be able to give hints, using the 'last remaining cell in a box, row or column'-algorithms, to the user as described in REQ-01-4.

REQ-03-2.1 The user must be able to upload a 9x9 irregular sudoku, starting with an empty grid as described in REQ-02-3.
REQ-03-2.2 The software must be able to solve the irregular sudoku.
REQ-03-2.3 The software must be able to give hints to the user about the next logical step in solving the irregular sudoku.

REQ-03-3.1 The user must be able to log into and account using a username and password.
REQ-03-3.2 The user must be able to sign up for a new account using a non-existent username and password.
REQ-03-3.3 All password information must be stored in a safe way where nobody has access to the exact password.

REQ-03-4.1 A user must be able to store a sudoku to their account at any stage of completion.
REQ-03-4.2 A stored sudoku must be accessible in two ways; the sudoku at the moment it was saved to the account and the beginning state of the sudoku (the uploaded sudoku).
REQ-03-4.3 If the same sudoku is stored twice at different levels of completion the user must have the option to store it as a separate sudoku or to override the old save.
REQ-03-4.4 In the UI the user must be able to access any previously attempted or solved sudoku they have saved and solve the sudoku either from the beginning or by continuing from the saved state.

REQ-03-5.1 When solving a sudoku the time the user takes must be tracked and displayed in the UI.
REQ-03-5.2 The time is stopped and displayed when the user solves the sudoku.
REQ-03-5.3 The amount of hints used to obtain the solution is displayed after solving the sudoku.
REQ-03-5.4 Based on a TBD scoring system - based on time taken, difficulty and hints needed - a score is calculated and displayed after solving the sudoku.
REQ-03-5.5 For each type and size of sudoku a high score list of time taken to solve the sudoku is accessible to the user in the UI.
REQ-03-5.6 A total high score list (of the score described in REQ-03-6.4) for all types and sizes together is accessible to the user in the UI.

REQ-03-6.1 The user must have the option to add notes to the cells for any type and size, with the range of numbers allowed for that type and size.
REQ-03-6.2 When a cell is filled with a number the notes must be removed, but if the filled in number is later removed by the user the notes must reappear.

REQ-03-7.1 There must be an button in the UI which turns the background to a dark color and the text a light color (night mode)

REQ-03-8.1 When the user uploads a sudoku (REQ-01-2) the program automatically checks if there are any mistakes.
REQ-03-8.2 When the user uploads a sudoku (REQ-01-2) the program automatically checks if there are more than one solutions.
REQ-03-8.3 If the program finds a mistake (REQ-02-2.1) or if there are more than one solutions (REQ-02-2.2) the user is clearly notified which of the two conditions for the legality of the sudoku was violated.
REQ-03-8.4 The user can change and reupload the sudoku after trying to upload an illegal sudoku.

For this project we are confident that we can implement the aforementioned must-haves and should-haves in the given timeframe. To achieve this, a couple python libraries will be essential. For the sudoku generation we will call upon the built-in random-module to create random starting points for newly generated sudoku's. As for solving sudoku's, we plan to use the dlx module. The dlx module contains the dancing links algorithm capable of efficiently solving sudoku's. The algorithm and module were developed by Donald Knuth. For the GUI we will be making use of the PySide6 module. We will be implementing a combination of the PySide6 designer tool and manual GUI programming. To ensure that code is readable and functioning as intended we will make use of MyPy and PyTest.

All our group members are familiar with the basic strategies of sudoku solving. Furthermore, some group members have delved deeper into the mechanics and solutions of various sudoku puzzles. This understanding provides our group with the ability to critically evaluate which solving algorithms to implement. And gives us a baseline for the human sudoku experience that we aim to assist with this sudoku tool. This boosts our confidence for successfully addressing the core set of must- and should-haves.

The most significant challenge we anticipate with implementing the must-haves revolves around creating the graphical user interface (GUI). None of our group members are particularly familiar with PySide6. Nevertheless, we are optimistic about our ability to overcome this hurdle. The designer tool in PySide6 is expected to provide a clear and real-time view of the UI, simplifying the process. Additionally, there's a wealth of accessible information online about using PySide6 and its counterpart PyQt5, which we plan to leverage as we progress through the project.

Technical approach

We will be using a few libraries to assist us in developing our sudoku helping software. The first one is Pyside6, Pyside6 is an easy to use GUI tool, we will be using this to display sudokus to the user and give access to the functions included in the program. The second

one is mypy, mypy is used for consistent use of programming conventions across all parts of the code and to ensure good programming practices. For testing we will be using pytest to ensure the complex logic of our code works, this can for example be something like the sudoku solving algorithm which is quite complicated.

The second part of the libraries are more functional libraries and modules for our code. For generating random numbers, which is important for generating a sudoku, we will be using the built-in random module. For solving, apart from our own developed method which is brute force, we will be using the dlx library which implements the Dancing Link algorithm by Donald Knuth, this is an efficient way to solve exact cover problems like a sudoku. Having both algorithms can be useful for cross validation of solved sudokus and to see the time efficiency of our own solution. Lastly we will be using the os library in python, this way the code should run on most devices and not only Windows or MacOS which in turn improves user friendliness.

We will not be using numpy since the intent is to make a solver helping tool for humans and not necessarily the fastest sudoku solver. While numpy is quite a bit faster than standard Python lists and even more so than dictionary lookup this is not really perceivable by humans and this would make the code more complicated than necessary.

Apart from the libraries and methods we need to create this program, we also need the logic behind sudokus. On a website called ‘SudokuWiki.org’ the same program is created that we want to make. We will get our inspiration partly from here and it has a good explanation regarding the different techniques used to solve a sudoku. We will try to improve the flaws we see in this website and hope to create something better.

After setting the requirements using the MoSCoW-method, and thus defining what we would and should be able to accomplish within 10 weeks of developing our sudoku helping tool, it was decided to use the agile software development methodology for the project, which emphasizes collaboration and improvement of code. After this it was time to look at the program and how it should be functioning, this was mostly thinking about the inner workings of the program.

Our codebase consists of a few different modules, these can be separated into a few categories: GUI, using and importing sudokus, generating sudokus, and solving sudokus. These same categories are also implemented in the file structure in the project folder. This modularity gives us a lot of flexibility when coding which also ties into the agile development method. The same goes for testing the program, because of this modularity modules can easily be connected and disconnected and thus can be tested thoroughly. Because of the usage of mypy and following programming conventions the code should score quite high on code smell rating.

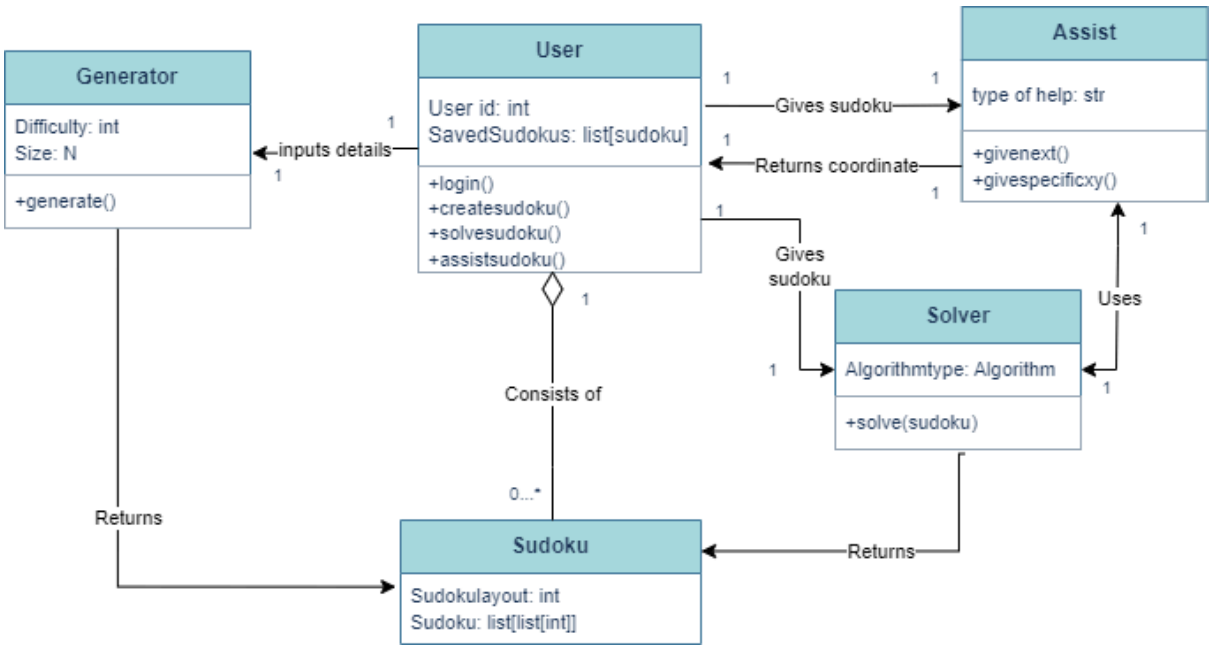

Figure 1: class diagram of the sudoku helping tool

The class structure is, as illustrated in the class diagram above, quite straightforward. The UI gives the user a few options of what to do, this can be generating or importing a sudoku. The user can then decide to solve manually, solve automatically, or ask for a logical hint or assist. This means we need a few classes: User, Sudoku, Solver, Generator, Assist. How these work can be seen in the class diagram. The User and Solver both solve sudokus, the Solver can check a Sudoku and give a solved sudoku to Assist. Assist then in turn can give a coordinate to the user. Important to note is that the user class is also partially a library of different saved sudokus.

Due to the modularity of the program it is possible for all the team members who have different skill levels in Python programming to contribute to the project in their own way. Since the software is developed using a non-resource intensive GUI and does not include complex algorithms it should be easy to run performance wise. As for scalability, because of the usage of the os library in Python it should run on most if not all modern computers.

Development methodology

The agile software development method ‘SCRUM’ was chosen for the project in order to ensure an effective working process. All requirements will be encapsulated by user stories which will be presented in the backlog, as well as indicate the prioritization of requirements over others. The SCRUM method was chosen over others due to its reflective nature within the sprint retrospective, which allows members to get a clear overview of what is going well and what is not. This enables us to easily detect and implement improvements in the work process or project features. It also offers a clear prioritization of what features are worked on during a sprint and by doing so ensures a focus on completing these features. The method is also still adaptable despite this, so any new features or changes in requirements can easily be added to the project, if these were to be found during the sprint retrospective. All team members are already familiar with this agile method, which makes it the easiest one to implement without having to spend too much precious time on learning it. In order to ensure

an effective process a variety of software tools were used. Pytest is used in order to orderly execute automated tests for all code written so code fails and where in the code they happen can be detected. Mypy is used as a type checker in order to improve the code smell of our code. GitLab is used in order to manage the repository and add new code, as well as review it and track issues. All the user stories created will be represented in GitLab issues so that they can be assigned to members, and the time tracking feature can be used to see how much everyone worked on each user story.

When interacting with the repository agreements were made in order to ensure a smooth operation. First of all, when developing a new feature this was done in a new branch in order to isolate it from the main branch which is reserved for complete work. The new branch is named accordingly after the issue that is worked on in the branch. By creating a separate branch for developing new features it ensures that the main branch always has a working and tested product. As mentioned before, all user stories are assigned to an issue on GitLab with the corresponding members that are going to work on it as well assigned to the issue. This is done in order to communicate clearly who works on which feature. When a member is done with their feature they can create a merge request and communicate this to other members, who then can write a code review for the merge requests. The new code must have tests written that provide adequate coverage and which all pass before an issue can be closed, as well as have classes and functions explained via well thought out comments and have arguments be typed. If no further improvements are detected during the code review by the other members the merge request can be approved and added to the main branch, afterwards the corresponding issue can be closed. If improvements are detected these first need to be implemented in the new code and a new merge request needs to be created with the improved code, so afterwards the reviewing and/or approving process can start again. When approved the developer that has written the new code then still needs to update the README if necessary.

Furthermore, a couple of agreements were made for handling group interactions. When meeting on Tuesday it was agreed that one person would be chosen beforehand to document the feedback of the TA meeting so this could then be used most effectively. These notes are also uploaded to the repository on GitLab after the TA meeting by the member who took them. On Tuesday the weekly SCRUM meeting would also take place since due to it being at the beginning of the week and within the course schedule which makes it the best time for it. During the meeting the task division for the current sprint is clearly communicated to all members. The SCRUM sprint will take 1 week, until the next Tuesday the team comes together. The Mattermost channel is looked at regularly too during the week by all group members for possible messages from the TA. Daily group communication regarding questions or encountered issues is done over whatsapp, as well as requesting code reviews on merge requests, in order to get the fastest response. Possible additional meetings during the week between group members are also agreed upon over whatsapp, and then held on campus or online via discord depending on the availability of group members.