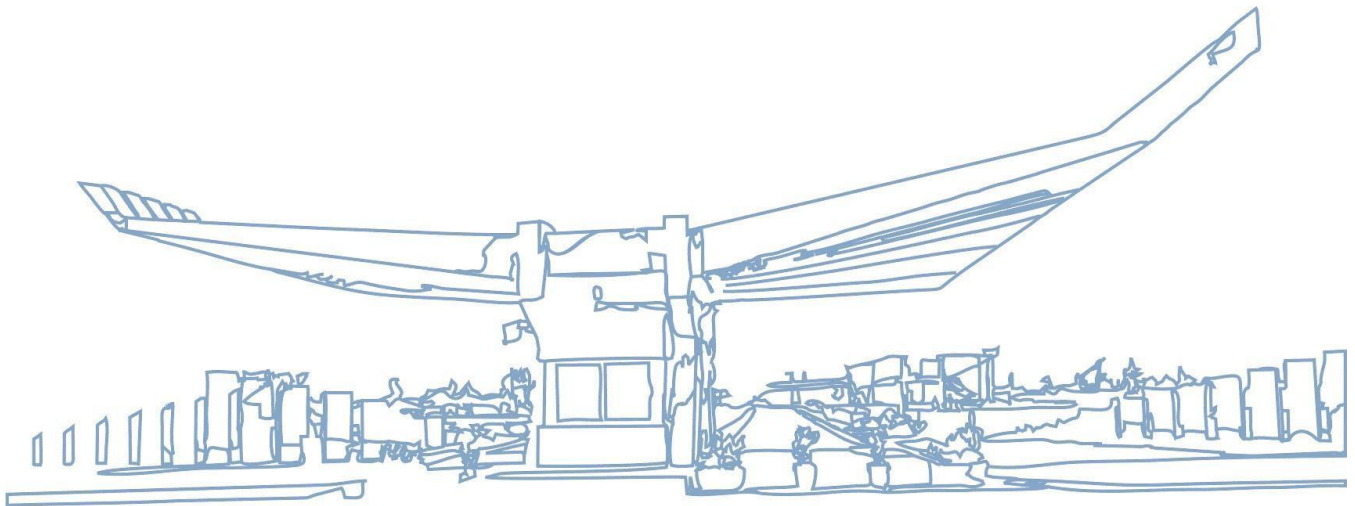


INTRODUCTION TO SOFTWARE ENGINEERING

Sudoku Solver



Team Members:

Aurora Zhibaj

Ema Kola

Keit Nika

Klea Faqolli

Table of Contents

1. EXECUTIVE SUMMARY	3
1.1 PROJECT OVERVIEW	3
1.2 PURPOSE AND SCOPE OF THIS SPECIFICATION	3
2. PRODUCT/SERVICE DESCRIPTION	3
2.1 PRODUCT CONTEXT	3
2.2 USER CHARACTERISTICS	3
2.3 ASSUMPTIONS	3
2.4 CONSTRAINTS	4
2.5 DEPENDENCIES	4
3. REQUIREMENTS	4
3.1 FUNCTIONAL REQUIREMENTS	4
3.2 NON-FUNCTIONAL REQUIREMENTS	5
3.2.1 <i>User Interface Requirements</i>	6
3.2.2 <i>Usability</i>	6
3.2.3 <i>Performance</i>	6
3.2.4 <i>Manageability/Maintainability</i>	7
3.3 DOMAIN REQUIREMENTS	8
4. DESIGN THINKING METHODOLOGIES	8
4.1 Empathy	8
4.2 Define	8
4.3 Ideate	8
4.4 Test	8
4.5 GUI	9
5. SOFTWARE DESIGN	9
5.1 Sequence Diagram	9
5.2 Use Case	10
5.3 Class Diagram	10
APPENDIX	11
APPENDIX A. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS	11
APPENDIX B. REFERENCES	11

1. Executive Summary

1.1. Project Overview

The Sudoku Solver is an application that allows users to enter Sudoku puzzles, ranging from low to medium levels of difficulty, and returns the solution. The program includes a basic user interface; it takes the input from the terminal space, where it also displays the solved puzzle. Several testing features are built in, to ensure that the program runs efficiently and correctly.

1.2. Purpose and Scope of this Specification

This specification intends to provide a detailed description of the requirements for the Sudoku Solver. The scope of this specification includes user interface, input validation, algorithm design, error handling, testing, and documentation.

2. Product/Service Description

The Sudoku Solver is a program that solves Sudoku Puzzles, sizes 9x9 and 16x16. In the terminal space, the user is prompted to specify the size of the puzzle and enter an unsolved puzzle, in which the empty cells are represented by 0. The program includes specific functions and algorithms to solve puzzles accurately. It's also made of testing components that ensure a valid and correct execution.

2.1. Product Context

The Sudoku Solver is an independent software, intended to run in the terminal space of IDE-s in mobile and desktop platforms.

2.2. User Characteristics

The Sudoku Solver is designed for users of all skill levels.

2.3. Assumptions

The user must already have an IDE(Xcode or CodeBlocks) installed. It is assumed that users will have a basic understanding of Sudoku puzzles and will be able to input puzzles into the application.

2.4. Constraints

The Sudoku Solver is designed to solve standard 9x9 and 16x16 puzzles. Any other size is not supported by this program. Since the input and output will be handled in the terminal space of the IDE, the design possibilities are quite limited to text-based formatting of the puzzle and basic keyboard input.

2.5. Dependencies

It can only be directly accessed via Xcode and CodeBlocks programs.

3. Requirements

3.1. Functional Requirements

Req#	Requirement	Comments	Priority	Date Reviewed
BR-01	The code should be able to solve puzzles of varying difficulty levels.	The program as displayed may limit the user to 2 sizes only, 9 and 16 squares, however the code itself can manage to solve puzzles of other sizes.	1	20/05/23
BR-02	The code uses backtracking to solve the puzzles entered by the user.	Algorithms other than backtracking, such as pruning, may be faster when solving the puzzles.	2	24/05/23
BR-03	The program uses a matrix as a data structure, where it stores, modifies and then displays the solved puzzle.	The program uses an algorithm which is not very efficient when it comes to the 16 by 16 grid; it shows difficulty when solving it.	3	28/05/23

BR-04	The code allows the user to interact with the program by giving them an option to choose the size of the puzzle, to terminate the code and to exit it.	In the terminal, the user will be met with a size constraint. All input is collected as characters, therefore the user has the option to enter a letter to exit the code.	1	28/05/23
BR-05	The program provides a mechanism to display or output the solved Sudoku Puzzle.	The program displays the unsolved puzzle first, then after checking it, it shows the user the solved one. The display grid is fairly basic, with the top and bottom margins adapted for each puzzle size.	1	29/05/23
BR-06	The program provides a mechanism for users to input the initial state of the Sudoku puzzle.	Empty spaces are represented by 0 in the Sudoku Puzzle.	2	29/05/23

3.2. Non-Functional Requirements

Product Requirements

- **Accuracy:** The code should provide correct and accurate solutions for Sudoku puzzles.
- **Scalability:** The code should be able to handle Sudoku puzzles of varying sizes, from standard 9x9 puzzles to larger puzzles like 16x16 or even custom sizes.

Organizational Requirements

- **Maintainability:** The codebase should be well-structured and maintainable, making it easy to enhance, modify, and fix issues in the future.
- **Extensibility:** The code should be designed to allow for future enhancements or modifications, such as adding new solving techniques or puzzle variations.

External Requirements

- **Compatibility:** The code should be compatible with different input formats or puzzle representations commonly used in the Sudoku community.
- **Security:** The code should ensure the confidentiality and integrity of user data, especially when handling puzzles submitted by users.

3.2.1. *User Interface Requirements*

The Sudoku Solver has a user friendly interface which allows the user to enter a Sudoku Puzzle and view its solution. It will provide clear instructions on how to input the puzzle, initiate the solving process, and view the solution. If the inputted puzzle is not solvable or if there is a mistake in the input, the user will be notified by an error message that will indicate the problem, which will be displayed on the screen. Once the Sudoku Solver has found a solution, the solved puzzle will be displayed in a clear and understandable format for every user.

The menu gives users two options :

- *Choose a 9 by 9 grid*
- *Choose a 16 by 16 grid*

The options will be chosen by either entering 1 or 2 (*any other input will immediately terminate the program*).

To exit the program, the user will enter the letter 'e'.

3.2.2. *Usability*

We will provide the user with a user manual, guiding him through every step of the program.

[USER GUIDE](#)

3.2.3. *Performance*

1. *The code supports one user at a time to enter puzzles in the program.*
2. Time complexity: For a puzzle of size 9×9 , for every unassigned index, there are 9 possible options so the time complexity is $O(9^{(n*n)})$. Size 16 yields a time complexity of $O(16^{(n*n)})$.

3.2.4. *Manageability/Maintainability*

3.2.1.1 *Monitoring*

Error detection is ensured through intricate algorithms and functions that overlook every single character the user input, and checks whether they are valid (letters and numbers outside the scope of 0-N, where N represents the size of the puzzle, are invalid). The verification step is carried on by displaying the puzzle as received through the input and double-checking whether the valid number inputs are correct. The program makes sure to retain the numbers from 1-N in their designated positions, as per user input, replacing only the 0-s with the appropriate solution (number from 1-N).

3.2.1.2 *Maintenance*

Various functions are implemented and integrated within the system, ensuring a functional run of the program. Functions vary from data input, validity check, solution with backtracking, to displaying the puzzle output, with and without the solution.

3.2.1.3 *Operations*

No special operations are needed to guarantee the functionality of this software.

3.3. *Domain Requirements*

Valid Puzzle Representation:

- ⇒The code should accept valid Sudoku puzzles represented as a grid of numbers, where empty cells are represented by zeros.
- ⇒The puzzle grid should adhere to the rules of Sudoku, including having unique numbers in each row, column, and 3x3 or 4x4 subgrid.

Puzzle Solving Techniques:

- ⇒The solver should employ appropriate techniques to solve Sudoku puzzles, such as backtracking, and logical reasoning.
- ⇒It should handle unique solving scenarios, as well as puzzles with multiple possible solutions.

Puzzle Input and Output:

- ⇒The code should provide means to input Sudoku puzzles through the terminal space.
- ⇒It should output the solved puzzle grid visually, maintaining the standard Sudoku grid structure.

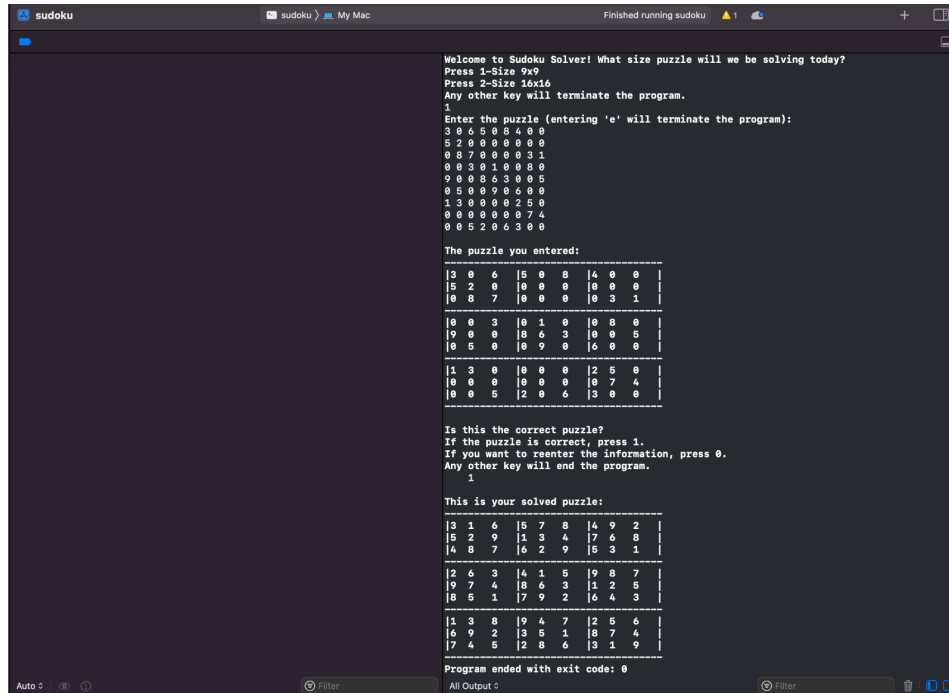
Error Handling and Validation:

- ⇒The code should detect and handle invalid or unsolvable puzzles, providing appropriate error messages to the user.
- ⇒It should validate the input puzzle for correctness and adherence to Sudoku rules before attempting to solve.

4. *Design thinking methodologies*

- 4.1. **Empathy:** We as programmers have conducted various researches throughout the Internet and with other students in our class to finally be able to say that the most used Sudoku puzzle sizes are 9x9 and 16x16, and those are the sizes we have used in our code.
- 4.2. **Define:** The code will solve the puzzles of different sizes, exactly standard 9x9 and 16x16 efficiently and quickly. The input is taken one by one from the keyboard and when it is not valid the code prints a message .
- 4.3. **Ideate:** We have come up with different ideas to make our code more efficient and fast. Throughout the time we have developed our code we have encountered some difficulties including either the validation of the input or solving the puzzle (mostly the 16x16 puzzle). But we have worked together, each one of us giving unique solutions until the code was successful.
- 4.4. **Test:** To be sure that our code is efficient and displays the correct solution, we have done two unit tests. Each test is done by a function, where one test is for a 9x9 puzzle and the other is for a 16x16 puzzle.

4.5. GUI (Screenshots)



```

Welcome to Sudoku Solver! What size puzzle will we be solving today?
Press 1-Size 9x9
Press 2-Size 16x16
Any other key will terminate the program.
1
Enter the puzzle (entering 'e' will terminate the program):
3 6 5 0 8 4 0 0
5 2 0 0 0 0 0 0
0 8 7 0 0 0 3 1
0 0 3 0 1 0 8 0
9 0 8 6 3 0 0 5
0 5 0 0 9 0 6 0
1 3 0 0 0 2 5 0
0 0 0 0 0 0 7 4
0 0 5 2 0 6 3 0 0

The puzzle you entered:
3 0 6 | 5 0 8 | 4 0 0 |
5 2 0 | 0 0 0 | 0 0 0 |
0 8 7 | 0 0 0 | 0 3 1 |
-----
0 0 3 | 0 1 0 | 0 8 0 |
9 0 8 | 6 3 0 0 5 |
0 5 0 | 0 9 0 | 6 0 0 |
-----
1 3 0 | 0 0 0 | 2 5 0 |
0 0 0 | 0 0 0 | 0 7 4 |
0 0 5 | 2 0 6 | 3 0 0 |
-----

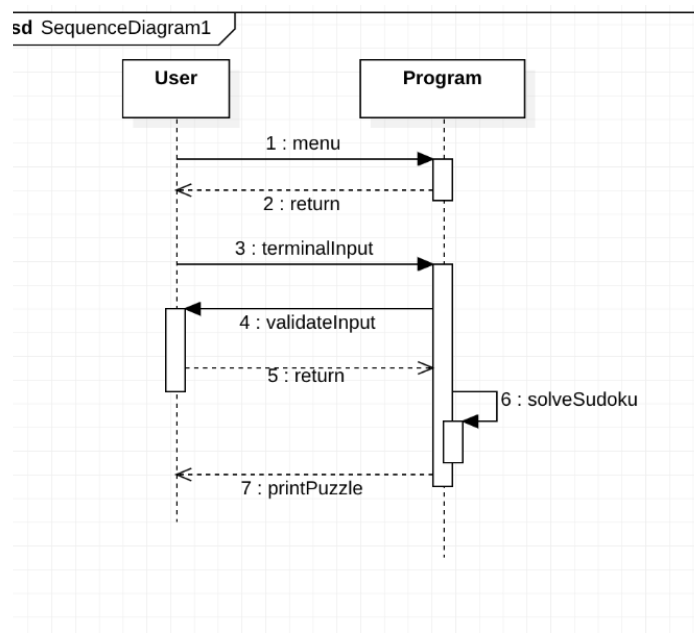
Is this the correct puzzle?
If the puzzle is correct, press 1.
If you want to reenter the information, press 0.
Any other key will end the program.
1

This is your solved puzzle:
3 1 6 | 5 7 8 | 4 9 2 |
5 2 9 | 1 3 4 | 7 6 8 |
4 8 7 | 6 2 9 | 5 3 1 |
-----
2 6 3 | 4 1 5 | 9 8 7 |
9 7 4 | 8 6 3 | 1 2 5 |
8 5 1 | 7 9 2 | 6 4 3 |
-----
1 3 8 | 9 4 7 | 2 5 6 |
6 9 2 | 3 5 1 | 8 7 4 |
7 4 5 | 2 8 6 | 3 1 9 |
-----

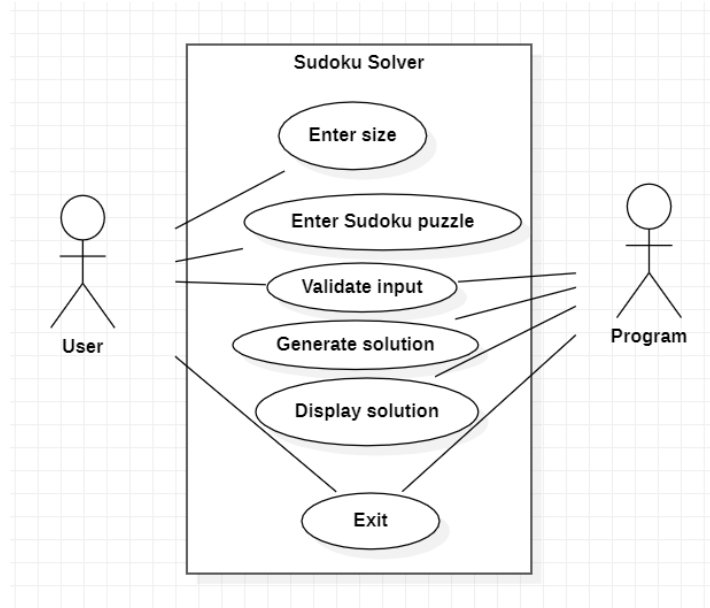
Program ended with exit code: 0
  
```

5. Software Design

5.1. Sequence Diagram



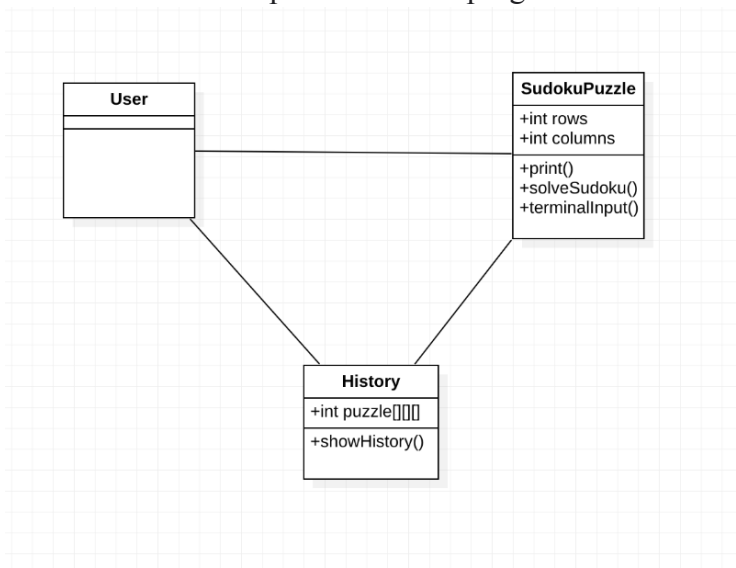
5.2. Use Case



5.3. Class Diagram

Our project does not involve object oriented programming, so translating it into a class diagram is not really appropriate. But when considering system evolution, eventually some new features could be added to the product, making it more user friendly.

We could add a history feature in order to store previous puzzles and display them to the user. Eventually, users could create accounts to keep track of their progress.



APPENDIX

Appendix A. Definitions, Acronyms, and Abbreviations

- **Algorithm** a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
- **Backtracking** an algorithmic technique whose goal is to use brute force to find all solutions to a problem.
- **Codebase** The complete body of source code for a software program, component or system.
- **Code Blocks** a free C/C++ and Fortran IDE built to meet the most demanding needs of its users.
- **GUI** A graphics-based operating system interface that uses icons, menus and a mouse (to click on the icon or pull down the menus) to manage interaction with the system.
- **IDE** A software application that helps programmers develop software code efficiently
- **O: Big O** a time complexity notation.
- **Terminal space** A text input and output environment.
- **Time Complexity** The amount of time taken by an algorithm to run, as a function of the length of the input.
- **User Interface** The point of human-computer interaction and communication in a device.
- **Xcode** is Apple's integrated development environment (IDE) for all Apple's platforms.

Appendix B. References

USER GUIDE pg 6.