OCR A Level NEA 2022

Computer Science

Sudoku Solver

U.Arjun

4 July 2022

Contents

1 Analysis

1.1 Problem Definition

My project is focused around sudokus and it will be able to recognize a sudoku puzzle from an image, and then solve the puzzle using a backtracking algorithm. There are many programs which can take in string inputs to solve the sudoku but I will be developing upon this by incorporating a handwriting recognition AI to recognise the sudoku from an image. This prevents the need to input numbers into the relevant row and column. There are various algorithms to solve sudokus however, backtracking has been found to be the most effective in different studies and so I will be using it.

1.1.1 Scope

There are a number of limiting factors that may affect the performance of this project and I have outlined the key factors below:

- Time: As ideal it may be refine the program with time, there's a time constraint on the development of this program. This may result in certain features not being implemented or as polished as possible such as the neural network which will be used to train the handwriting recognition AI.
- Skills: Whilst the project could be developed in a more efficient way by others, I have a limited scope of skills which will dictate the effectiveness of this project. Moreover, I am using a basic neural network to develop my AI and there are alternatives that may be more accurate such as a convoluted neural network but require more expertise to implement.

1.1.2 Purpose

The reasoning behind this project is to facilitate the process of finding a solution for a sudoku. Generally, it is very easy to find a sudoku to complete but producing a solution for it is much more complex. As such, this program will allow users to upload an image of an uncompleted sudoku and then receive a solution for it. This would save time for the user as they wouldn't have to spend time inputting values manually whilst also being able to check their sudoku.

1.1.3 Stakeholders

Following the development of mobile phones, sudokus have lost popularity significantly over the years. Nevertheless, there are still numerous enthusiasts who continue to complete them as a hobby. Due to its potential to stimulate brain activity, many people attempt to complete at least one sudoku each day and for all those people, this program would save them time significantly because a solution will be produced in less than a minute. Whilst there are programs which can solve sudokus based on a string input, there could be human error in copying over the problem and so, using a camera to capture the problem would be more efficient. It won't be flawless but the user will be presented with the option to verify the computer's interpretation.

1.2 Background information

Simple rules of sudoku

Before continuing, it would be helpful to understand the basic rules of sudoku. The game is usually played on a 9 by 9 grid filled with numbers in random locations. Each sub grid of 3 by 3 has to contain all 9 numbers from 1-9. This is the same for each column and each row. So, the aim is to complete the square by placing number appropriately to achieve that outcome. Each sudoku only has one solution and if not, then its likely that the player made a mistake and has duplicate somewhere or the sudoku is poorly designed. There have been numerous studies about the minimum number of clues necessary to produce a valid sudoku and the University College Dublin have proposed a widely accepted theory which I will be referencing later.

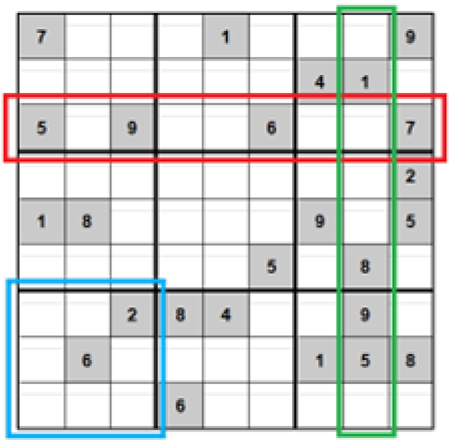


Figure 1: In the picture, all 3 outlined boxes should contain the numbers 1-9.

Basic terms in sudoku involve:
Candidate: an empty square may have a certain set of numbers that will fulfil the square without conflicting with other squares in the same column, row, or box
Box: the smaller 3 by 3 grids marked by the darker lines
Region: a column, row or box
Clue: refers to the numbers that were given by the board or the "starting numbers"

1.3 Model stakeholder

My model stakeholder is a student at the same school as me who enjoys doing sudokus everyday. His name is Alfred and because he wants to pursue a career in mathematics, he says that doing a daily sudoku helps him relax, while warming up his brain for the day.

I will interview Alfred to see what he thinks about my program and certain features that will be available. This will allow me to learn how this program may be used and which features are more important for users. Moreover, it gives me user feedback about my program which will aid me in providing updates to make the program more accessible.

1.4 How is the problem computationally solvable?

A problem becomes computationally solvable if it has a finite number of steps and is solvable by a series of mathematical steps, such as an algorithm. In particular, the Sudoku problem is known as an NP-complete problem. NP is a theory in computer science which refers to all decision problems - questions that can be answered with a yes or no - and for those problems, the answer can be verified in polynomial time of $O(n^k)$ where n is the problem size and k is a constant. []. In this regard, NP-complete is a subsection of NP and it describes the hardest set of these NP problems for which finding a problem is very difficult but verifying a solution is simple. As such, Sudoku has been investigated by numerous mathematicians and computer scientists who have been attempting to solve these NP-Complete problems. This problem uses a camera to recognize the digits, and this would need a computer to be feasible. As such, to be able to solve these sudokus efficiently, a computer will be needed to execute the backtracking algorithm in a minimal amount of time. However, it is worth highlighting that the computer may not always be faster than a human because a deterministic algorithm such as a rule-based algorithm may save time by confirming as many numbers as possible before iterating through each possibility. Ultimately, it varies between problems but in most cases, a computer can solve the problem faster than humans by iterating through all the possibilities.

1.4.1 Thinking Abstractly

To simplify the problem, I will be using abstraction to focus on the necessary details to make the design more accessible. When identifying the sudoku from the image, abstraction will be used to ignore all detail in the image except the sudoku grid and its numbers.

1.4.2 Thinking Ahead

Thinking ahead will allow me to effectively plan out the different components of my program and this will give me an insight of how to begin coding it.

- I will be using python to create this program and numpy to create a 2d array
- Then, I will be importing the tkinter library to allow the user to interact with the graphical user interface
- I'll train an AI to recognise written numbers
- The input would be numbers from the keyboard
- The output is a completed 2d array of the Sudoku - the clues could be in a different colour to distinguish the answers

1.4.3 Thinking Procedurally

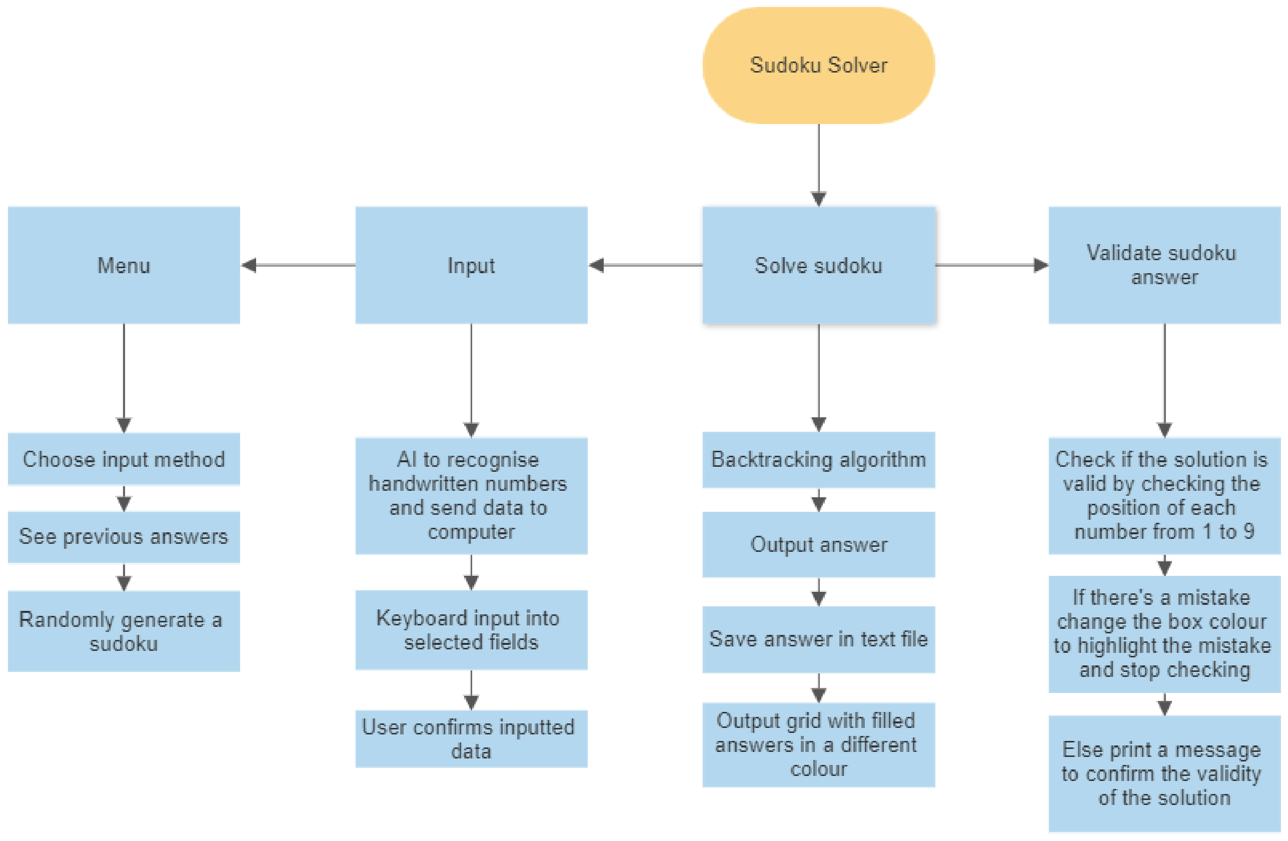

Sudoku Solver
Menu
Input
Solve sudoku
Validate sudoku answer
Choose input method
See previous answers
Randomly generate a sudoku
AI to recognise handwritten numbers and send data to computer
Keyboard input into selected fields
User confirms inputted data
Backtracking algorithm
Output answer
Save answer in text file
Output grid with filled answers in a different colour
Check if the solution is valid by checking the position of each number from 1 to 9
If there's a mistake change the box colour to highlight the mistake and stop checking
Else print a message to confirm the validity of the solution

Figure 2: Strucure Diagram

Although this demonstrates the same ideas as thinking ahead, the diagram helps me visualise the different sections of my program and the various subroutines which make up the section. As such, it simplifies the problem into smaller problems. The program has been broken down into smaller sections:

- Menu
  This allows the user to navigate through the program and access different features such as starting a generating a Sudoku, solving a new Sudoku, or seeing previous solutions to Sudokus.
- Input
  This allows the user to input their existing Sudoku problem onto the program via an image or inputting values manually.

- Solve Sudoku
  This section covers the coding necessary to solve the Sudoku using a backtracking algorithm.
- Validating Sudoku solutions
  This allows the program to validate a provided solution by checking each digit.

1.4.4 Conclusion

Therefore, this program is computationally solvable as it has a limited number of steps and finite number of options. Thus, a backtracking algorithm can be implemented to devise a solution to the sudoku. Moreover, thinking about the problem in different ways has allowed me to plan my solution and the steps I will need to achieve the desired solution.

1.5 Features of the proposed solution

My solution is a program which generates and solves sudokus for the user and can also check solutions. It can generate sudokus with a random number of clues given and it can solve any given sudoku using a backtracking algorithm. Furthermore, the user can input the sudoku in various ways including a photo input which will be read by Optical Character Recognition (OCR) AI or the user can manually type in the numbers in a string format. Then, the program will output the solution in a grid format to the user. If I have more time, then I will allow the user to view the answer to certain boxes as this will allow them to check the answer for certain boxes as they wish.

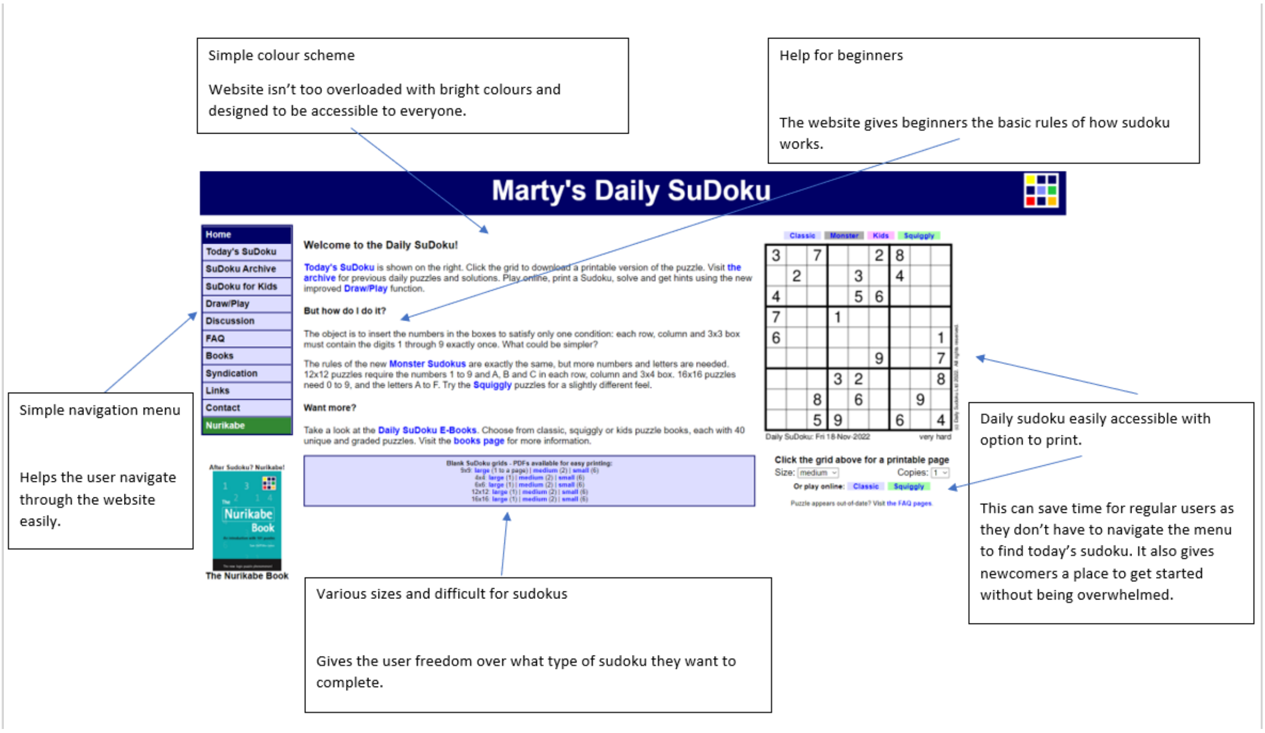
1.5.1 Limitations of this solution

The OCR AI will work best with neat handwriting where each digit is easy to understand. Also, the user will need a Python interpreter on their device but nowadays there is increased support for Python from various devices and operating systems. For example, QPython on Android; Pythonista on IOS and any IDE online can run Python.

1.6 Research

1.6.1 Existing Solutions

Marty's Daily SuDoku


Simple colour scheme

Website isn't too overloaded with bright colours and designed to be accessible to everyone.

Help for beginners

The website gives beginners the basic rules of how sudoku works.

Simple navigation menu

Helps the user navigate through the website easily.

Daily sudoku easily accessible with option to print.

This can save time for regular users as they don't have to navigate the menu to find today's sudoku. It also gives newcomers a place to get started without being overwhelmed.

Various sizes and difficult for sudokus

Gives the user freedom over what type of sudoku they want to complete.

Marty's Daily SuDoku

Welcome to the Daily SuDoku!

Today's SuDoku is shown on the right. Click the grid to download a printable version of the puzzle. Visit the archive for previous daily puzzles and solutions. Play online, print a Sudoku, solve and get hints using the new improved Draw/Play function.

But how do I do it?

The object is to insert the numbers in the boxes to satisfy only one condition: each row, column and 3x3 box must contain the digits 1 through 9 exactly once. What could be simpler?

The rules of the new Monster Sudokus are exactly the same, but more numbers and letters are needed. 12x12 puzzles require the numbers 1 to 9 and A, B and C in each row, column and 3x4 box. 16x16 puzzles need 0 to 9, and the letters A to F. Try the Squiggly puzzles for a slightly different feel.

Want more?

Take a look at the Daily SuDoku E-Books. Choose from classic, squiggly or kids puzzle books, each with 40 unique and graded puzzles. Visit the books page for more information.

Blank SuDoku grids - PDFs available for easy printing:
9x9: large (1 to a page) | medium (2) | small (6)
4x4: large (1) | medium (2) | small (6)
6x6: large (1) | medium (2) | small (6)
12x12: large (1) | medium (2) | small (6)
16x16: large (1) | medium (2) | small (6)

Daily SuDoku: Fri 18-Nov-2022 very hard

Click the grid above for a printable page
Size: medium Copies: 1
Or play online: Classic Squiggly

Puzzle appears out-of-date? Visit the FAQ pages.

Figure 3: Marty's Daily Sudoku

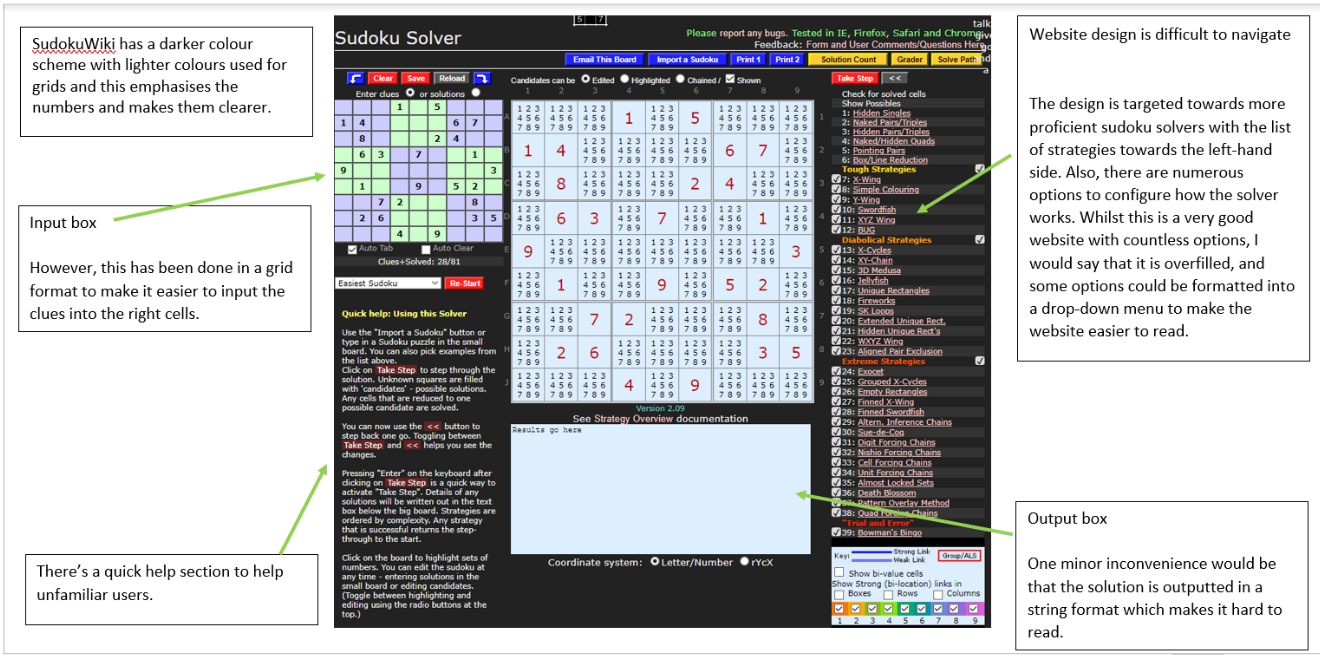
This is the website that Alfred uses and the one he mentions in the interview(Section 1.6.3). It is a very basic page design but has a range of difficulties for people of all abilities. This is a feature that I would like to integrate into my own program as it widens the target audience and doesn't restrict people because of their lack of ability. Also, the menu on the left hand side is very basic yet simple to navigate with. The website also offers a wide range of sizes of sudokus rather than only the standard 9 by 9 grid. There's also an option for user feedback which ensures that the website can be constantly improved to the user's needs and desires. Finally, the website offers a quick option to print out the daily sudoku in a given size which is ideal for users because it saves them time. Personally, I think this would be a very useful feature which has the user's needs at its core.

But there are some details that could be improved to make the website better as follows. I think that whilst a website makes it accessible for all devices, an app version would be ideal for mobile users. This means that the app would save even more time for the users as they do not need to load the website. Moreover, the app could be used to access the sudokus offline which would be helpful to solve the sudokus on the go such as on the underground where there is a limited availability of WiFi.

How does my solution differ?

By developing this as a standalone program, the user is able to access the sudoku offline as they desire because it doesn't have to be connected to the internet. Also, the solution can be accessed offline which would mean that the entire program can be used offline and this would benefit those who travel in areas with little to no network connectivity.

SudokuWiki



SudokuWiki has a darker colour scheme with lighter colours used for grids and this emphasises the numbers and makes them clearer.

Input box

However, this has been done in a grid format to make it easier to input the clues into the right cells.

There's a quick help section to help unfamiliar users.

Website design is difficult to navigate

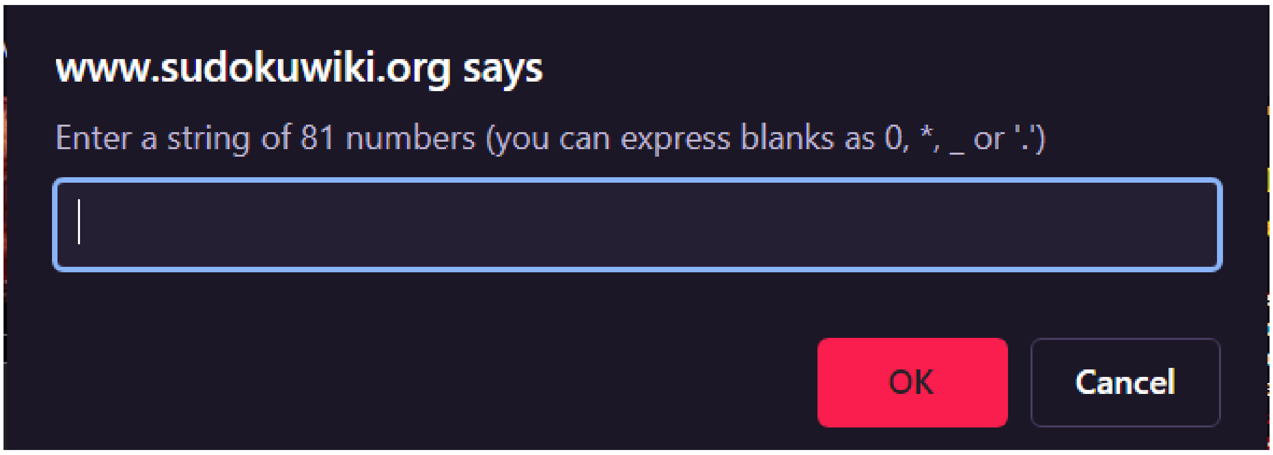
The design is targeted towards more proficient sudoku solvers with the list of strategies towards the left-hand side. Also, there are numerous options to configure how the solver works. Whilst this is a very good website with countless options, I would say that it is overfilled, and some options could be formatted into a drop-down menu to make the website easier to read.

Output box

One minor inconvenience would be that the solution is outputted in a string format which makes it hard to read.

Figure 4: SudokuWiki

This is another website designed for sudoku lovers which is targeted towards the more advanced sudoku players because it has a lot of information and is a bit more difficult to navigate through. However, this websites has steps to solve a sudoku including well known strategies and methods. The centre sudoku gives the user a template to solve the inputted sudoku. On the topic of inputting data, the website uses JavaScript to give the user an input box to send the data to the server as shown below.

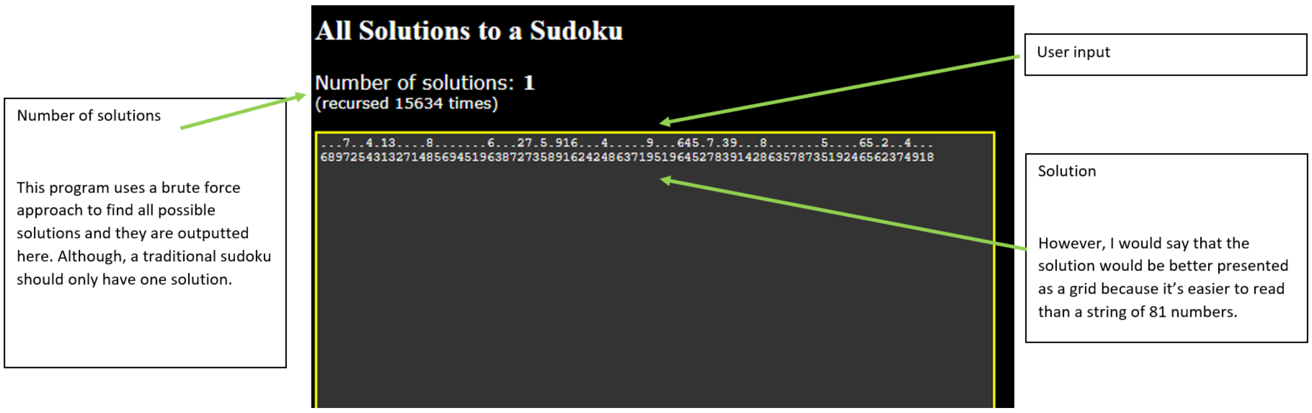


www.sudokuwiki.org says

Enter a string of 81 numbers (you can express blanks as 0, *, _ or '.')

Figure 5: Input box for users to submit clues

The website takes in an input in the form of a string and uses 0 or * or . to signify the blanks and solves the sudoku for the user and returns the solution in the format below.



All Solutions to a Sudoku

Number of solutions: **1**
(recursed 15634 times)

```
...7..4.13....8.......6...27.5.916...4.....9...645.7.39...8.......5....65.2..4...
689725431327148569451963872735891624248637195196452783914286357873519246562374918
```

Number of solutions

This program uses a brute force approach to find all possible solutions and they are outputted here. Although, a traditional sudoku should only have one solution.

User input

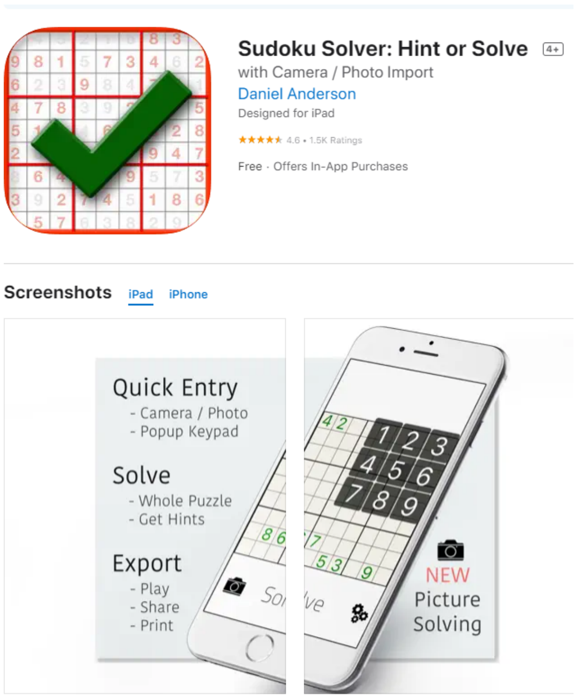
Solution

However, I would say that the solution would be better presented as a grid because it's easier to read than a string of 81 numbers.

Figure 6: Sudoku solution

My program has a similar concept to this website in the sense that my program will be able to generate and solve sudokus from an input. However, my program will output the solution in a grid format to maximise readability. In addition, I will use AI to recognise a handwritten input and this can reduce the time taken to input the clues. Moreover, it can reduce the amount of human error when inputting values. But it must be noted that the AI may misinterpret digits and thus, the user should be able to override the camera input to rectify any mistake.

IOS:Sudoku Solver: Hint or Solve



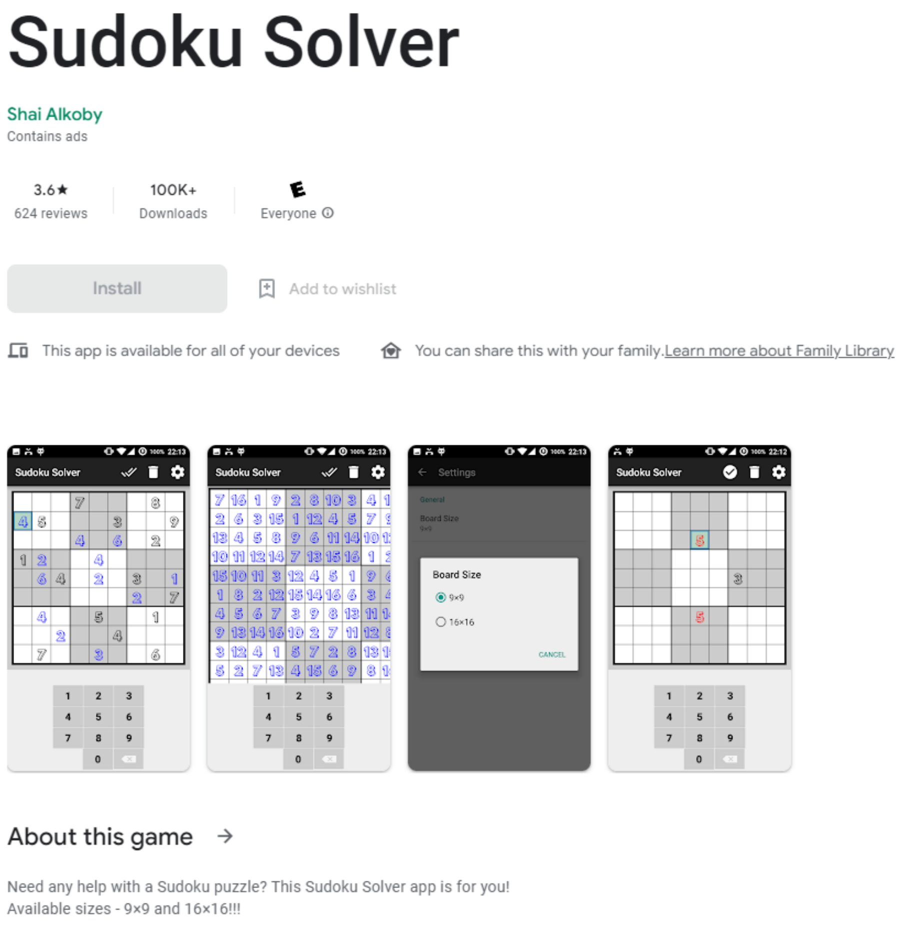
Sudoku Solver: Hint or Solve 4+
with Camera / Photo Import
Daniel Anderson
Designed for iPad
★★★★★ 4.6 • 1.5K Ratings
Free · Offers In-App Purchases

Screenshots iPad iPhone

Figure 7: IOS App

This app focuses on the ability to be able to solve sudokus and more specifically, it is able to enter numbers via the camera or photo option. This is ideal on a mobile based application as the user can easily find solutions on the go as per their needs. This aspect of camera input is a feature that I would like to implement in my solution as I believe that it facilitates the number input process and is effective in doing so. Apart from that, this app does not do much else but it also allows the user to receive hints rather than just seeing the full solution and this means that users can choose which clues they need. This is also a feature that I would be keen to implement in my own project as an extension but would require some more research into the coding process.

<u>Android: Sudoku Solver</u>
The android counterpart below has an added feature in the sense that it can work for 16 by 16 sudokus too. However, unlike the IOS app, it doesn't have the option to a camera input and especially when working with 16 by 16 sudokus, it can be extremely monotonous typing 256 digits into the relevant boxes. Therefore, I believe that, while the option to solve larger grids is useful, the app must be further refined to incorporate the whole functionality of solving larger grids. In particular, the ability to input sudokus more quickly and easily. Moreover, these larger grids wouldn't be suitable for a mobile application as it can be hard to view the whole grid at any one given time. As shown in the screenshot, once you select the 16 by 16 option, you then have to zoom out and in constantly. This can be inconvenient and thus, should be implemented on desktop or tablet applications.



This app is available for all of your devices

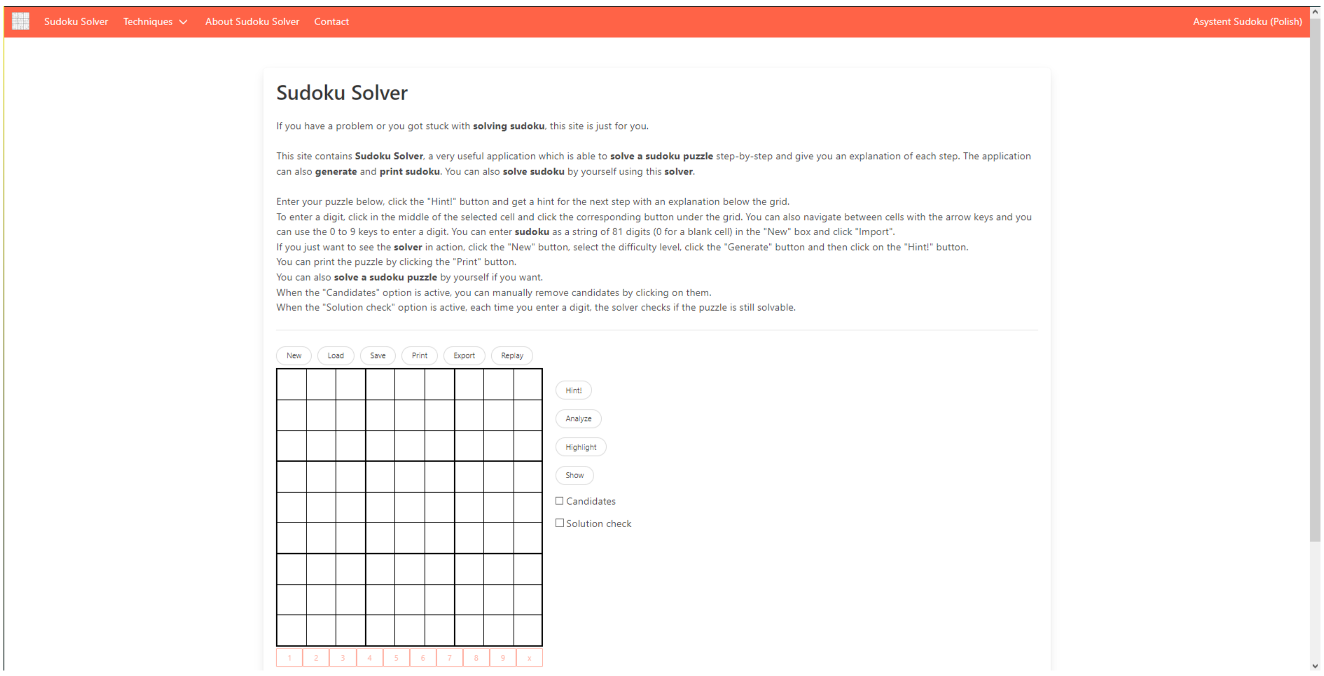
You can share this with your family. Learn more about Family Library

Need any help with a Sudoku puzzle? This Sudoku Solver app is for you!
Available sizes - 9×9 and 16×16!!!

Figure 8: Android App

<u>Website: Sudoku Solver</u>
My personal favourite of all the existing solutions is the website that I was researching into. This website has the capability to both generate and solve sudokus. Both functions have been well thought of and implemented. For the generate option, the user has a wide variety of skill levels including: very easy, easy, medium, hard, and fiendish. This gives the user freedom to select their desired difficulty level as well as selecting from various sudoku styles such as classic sudoku, sudoku X, sudoku Y and sudoku mini which gives the application more versatility. If all those options don't suffice then the user can manually design their own sudokus, so as you can see this app caters for all needs. When it comes to solving sudokus, the user can check candidates as well as solving the sudoku step by step which is desirable for newbies. The unique feature of this website is that it can offer hints to solving the sudoku at hand, which means helps players when they're at a dead end. However, the one drawback of this website is that it doesn't have a camera

input option and instead, it uses a string input which can be time-consuming as I mentioned before. But on the face of it, this website is well designed and has a minimalist user interface which isn't overloaded with options.



Sudoku Solver

If you have a problem or you got stuck with **solving sudoku**, this site is just for you.

This site contains **Sudoku Solver**, a very useful application which is able to **solve a sudoku puzzle** step-by-step and give you an explanation of each step. The application can also **generate** and **print sudoku**. You can also **solve sudoku** by yourself using this **solver**.

Enter your puzzle below, click the "Hint!" button and get a hint for the next step with an explanation below the grid.
To enter a digit, click in the middle of the selected cell and click the corresponding button under the grid. You can also navigate between cells with the arrow keys and you can use the 0 to 9 keys to enter a digit. You can enter **sudoku** as a string of 81 digits (0 for a blank cell) in the "New" box and click "Import".
If you just want to see the **solver** in action, click the "New" button, select the difficulty level, click the "Generate" button and then click on the "Hint!" button.
You can print the puzzle by clicking the "Print" button.
You can also **solve a sudoku puzzle** by yourself if you want.
When the "Candidates" option is active, you can manually remove candidates by clicking on them.
When the "Solution check" option is active, each time you enter a digit, the solver checks if the puzzle is still solvable.

Figure 9: Website Application

1.6.2 Interview 1

Prepared Questions

I have devised a set of questions to ask my stakeholder as this would help me improve my program and tailor it towards my target audience. For each question, I have explained my reasoning behind it and what I aim to learn from that question.

How do you currently get solutions for sudokus that you find online?
This question will give me an insight into how others find solutions and this links in with the existing solutions.

Would you prefer a scanner to take in an input for the sudokus?
By understanding the user's needs, I can tailor the program towards their needs and this will make the program more helpful for them. Furthermore, if a user doesn't like a feature or wouldn't use a certain feature, then it would be a waste of time to implement that feature.

Where do you go to find sudokus?
This helps me understand the current trends in the sudoku world and learning about their habits will teach me about other programs similar to mine and how I could improve my program based on the features that I like.

How important is checking your answer for a certain box?
When completing sudokus, it can be helpful to check certain numbers to see if you're on the right track so for me personally I think that it would be helpful to see if a proposed candidate is correct.

What colour would you prefer on the background of the program?
This would help me make the program more accessible to a wider audience by choosing colours that are preferred by the target audience or suitable for colour blind people. Perhaps, I could add some functionality to be able to change the background colour.

1.6.3 Interview Script

Q: How do you currently get solutions for sudokus that you find online?
A: Sometimes, there will be answers below on certain websites but I find that there's not always solutions available which is a bit frustrating because I have to search up a solution online by inputting my values manually. One particular website I use often is Marty's Daily Sudoku because the interface is simple and easy to navigate and it also has a daily sudoku option, an archive of all sudokus and even an option for kids.

Q: Would you prefer a scanner to take in an input for the sudokus?
A: Yes definitely because it would speed up the process of finding a solution because I can scan my sudoku and it will check if all the values are valid in less than a minute. After spending 15-30 minutes doing a sudoku, I'm usually too tired to check my answers and I assume they are correct because I did the sudoku.

Q: Where do you go to find sudokus?
A: I just search online for daily sudokus and use whichever link is on top. Sometimes, I look at the sudokus found on a newspaper but that's rare because it's easier to find a sudoku online.

Q: How important is checking your answer for a certain box?
A: I don't because most sudoku solvers will show me all the answers and that takes the fun out of doing sudokus. Thus, I wait till I am completely finished before checking my answers. I would say that this means that if I'm wrong, then I have to basically start again because I don't know which numbers to switch.

Q: What colour would you prefer on the background of the program?
A: I think that neutral or light colours would be ideal to improve clarity of the numbers.

1.6.4 Review

Following this interview, I have learnt that the ability to check an answer is an important feature and also, the stakeholder likes the idea of being able to input all the values with a simple image. Furthermore, there's no particular preference when it comes to colour but the stakeholder has emphasised clarity of characters and so choosing a neutral or light colour would facilitate this. Whilst Alfred doesn't commute often, there may be others who like to complete sudokus on the go and therefore, the ability to find answers for sudokus found on newspapers could be beneficial.

1.7 Hardware and software requirements

Hardware:

- Any computer or device which has a keyboard input - This allows the user to input values into the program and navigate through the menu. A mouse can also be used to navigate through the Graphical User Interface (GUI) and when a picture input is used rather than manually typing in values.

Note: Whilst OpenCV works on mobile devices like Android and IOS, there is some further tweaking and coding to be done for it to work. Thus, it will be an extension if I have time to improve my program.

Software:

- Python Interpreter - this can vary between OS and device but any python interpreter to run the code will suffice
- Numpy for Python - this is a library in python which is used for making grids

- Tensorflow for Python - this is a library in python used for OCR
- OpenCV for Python - this is a library in python used to analyse images

1.8 Success Criteria

List of success criteria				
Number	Section	Objective	Purpose	Completed?
1	GUI	Create an interactive GUI	Allow the user to navigate the program using the GUI	
2		Create buttons on GUI	Create buttons which can be clicked to navigate the app	
3		Change background colour	Change the background colour to a neutral colour to maximise readability and be accessible for the visually impaired	
4	GenerateClues	Generate clues for the sudoku	Generate sudokus based on NumberOfClues	
5	FinalSudoku	Output sudoku	Output sudoku grid with empty blanks shown	
6	KeyboardInput	Take keyboard input of digits	Validate data input and if all clues have been supplied	
7	CameraInput	Handwriting model	Develop and train a model for a neural network	
8		Develop handwriting recognition AI	Develop and train the AI using the neural network	
9		uploadFile	Take the image input of a sudoku from user	
10		translateData	Convert the image into a data form suitable for the computer such as a string or array	
11		Process image	Process the image to then be able to extract the digits from it	
12	SolveSudoku	Validate input	Check if it's a valid sudoku which has been inputted	
13	Solution	Develop backtracking algorithm	Develop and implement the backtracking algorithm	
14		Formulate solution	Use algorithm to develop a solution	
15		Output	Output the solution in a grid and string format	
16	Extension	SolutionColour	Change the colour of the digits in the solution but not the given clues	
17		User change background colour	Allow the user to change the bg colour as they wish	
18		Save previously solved sudokus	Save the solutions to a text file in case the user wants to refer back to them	
19		Help text file	Create a text file which outlines how to use the program	
20	Extension	Make program accessible to mobile users	Allow users to run the program from mobile devices such as Android or IOS devices	

2 Design

2.1 Problem Decomposition

The main problem here is the decline in sudoku players and the loss of popularity for sudokus. In our ever so busy lives, we are increasingly obsessed with our phones and our brains are constantly stimulated by videos and social media. As such, games like sudokus and crosswords, which boost brain performance, are on the decline and fewer people are engaging in these activities. Noticeably, the older generation's unfamiliarity to technology has meant that some of them have kept up their old hobbies, like completing sudokus.

Below I have broken down the problem into sections and how they relate to the decline of sudokus.

2.1.1 Sudoku source

While there are numerous avenues to find sudokus such as newspapers, websites and apps, there is limited customization of sudokus such as selecting the number of given clues to manage the difficulty. Moreover, not everyone can access these resources easily. For example, certain websites and apps may not be accessible for the visually impaired. In other cases, the user is expected to buy a subscription which may not incentivize them to continue to attempt a sudoku. Furthermore, for those who cannot go outside, such as the vulnerable, elderly and socially disadvantaged, there's limited access to sudokus. Due to the reasons above, fewer people are willing to go through all the effort to find a sudoku and complete them. To solve this problem, I will have to provide users a way to easily find sudokus in one place which is accessible to everyone.

2.1.2 Sudoku difficulties

Sudokus can be unfamiliar to begin with, and it is a learning process and players will start to notice certain patterns and tricks to make the sudoku easier. But apps and websites are notably pioneered for the more skilled and this could explain the decline in players because it's challenging for a beginner to enter the world of sudokus. Also newspapers try to make sudokus accessible for everyone but the wide range of skill levels can often mean that newbies struggle and are discouraged to continue. My program will allow users to moderate the difficulty of their sudokus to cater to their needs. By developing it as a program rather than a website, users will not require a network connection to use the program. This allows users to access the program in areas of limited connection such as underground, or in countries where there's limited network connectivity.

2.1.3 Sudoku solutions

As a sudoku player myself, I find it frustrating that sudoku solutions are not supplied alongside the sudoku and therefore, dead ends can be frustrating. Also for some sudoku solvers, only one solution is outputted but certain sudokus prove to have more than one solution. While this shouldn't be the case, according to studies by University College Dublin, Gary McGuire theorizes that sudokus with less than 17 clues can mathematically have multiple solutions. It would be futile for a sudoku player to copy the answers down, as they gain nothing from it and there is no other reason to avoid supplying solutions for sudokus. This can be rectified by my program which has the ability to solve any given 3*3 sudoku and by combining all these features in a single program, sudoku players will only need this one program.

2.1.4 Facilitate sudoku solver inputs

Current sudoku solvers are mainly website-based, to make it accessible to a larger audience which is useful. However, these online solvers require users to enter all 81 digits ,including blanks, manually and this can be monotonous. On top of that, it is very easy to make a small mistake and if unnoticed, this can change the solution completely. There are some app-based sudoku solvers, however these tend to be specific to operating systems such as Android or IOS and there's no universal sudoku solver for all platforms. Also, some apps require the user to pay small amounts of money to cover the relevant app store fees. My program will use python to operate and this means that any device with a python interpreter will be able to run it and it will be completely free.

2.2 Solution Decomposition

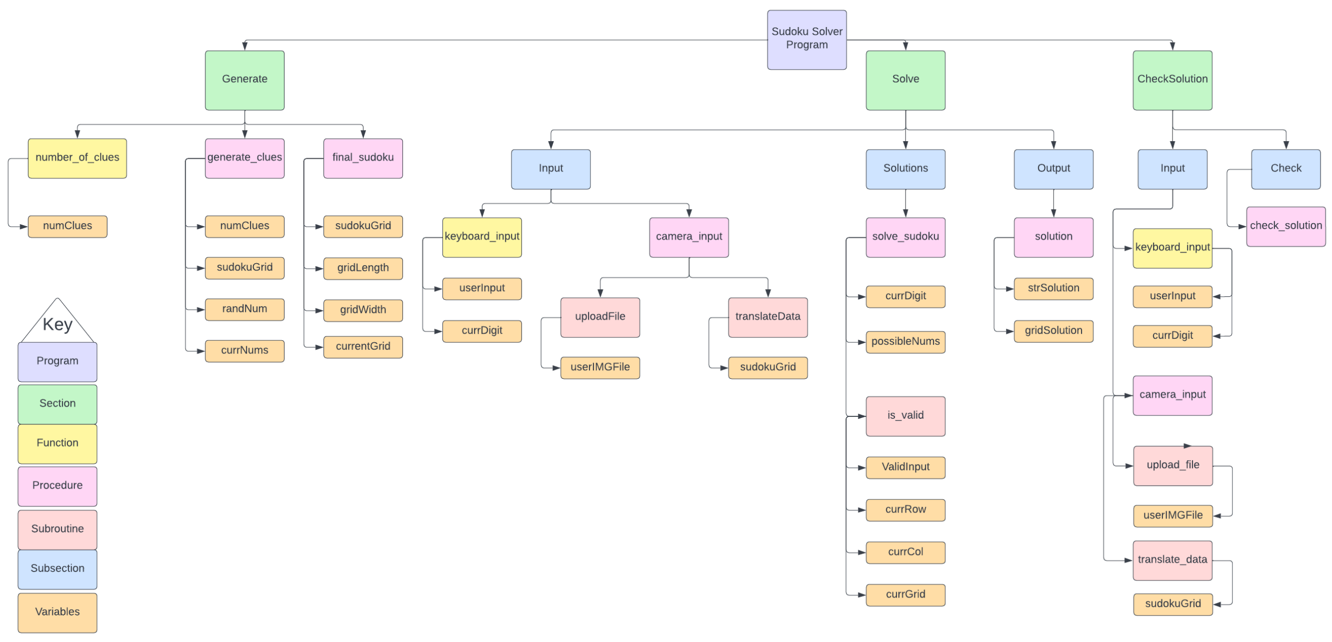


Figure 10: Sudoku solution decomposition structure diagram

Similar to the problem decomposition, I have broken down my solution into sections. There are three main sections which are generate, solve and check. This allows me to break my problem down into smaller sections and see the solution on a higher level, to simplify the coding of the program. In each section, I have thought of the different procedures which make up that section and all the variables, which will be needed in those procedures. I will be approaching the solution in an modular approach and this means that I can code the solution in modules and these can be reused when necessary. A modular approach has many advantages and a few of these include: re-usability of code; easy testing and modifying; and easy to optimise code. As I will explain in greater detail later, the modular testing is helpful as each module will have a clear purpose and if it fulfills that purpose, then the module is functioning as expected. Also, modular approaches are beneficial to groups as they can code sections and collate the modules to form a solution.

In the "Generate" section, the first function is number_of_clues and this allows the user to input the number of clues that they want in their generated sudoku. This is an integer input which the user types in ,and will be stored to be used in the next procedure, generate_clues. number_of_clues is a function because it produces information whereas procedures only perform a task. generate_clues is also a function because it uses the information from number_of_clues to generate and return a random set of clues to place in the sudoku ,whilst ensuring that all the clues are valid. This will be done by using the procedure is_valid found in the "Solve" section of my program. Here we see the re-usability of modules allowing us to save time by using a module of code which has already been coded. This links to success criteria number 4 where I aim to successfully generate valid clues for a blank sudoku. Also, part of this section is criteria number 5, which describes outputting the generated sudokus in a grid format to the user to maximise readability. This procedure will use both generate _clues and number_of_clues to output a randomly generated sudoku with the number of digits that the user requested. This solves the problem of sourcing sudokus that are also tailored to the user's ability.

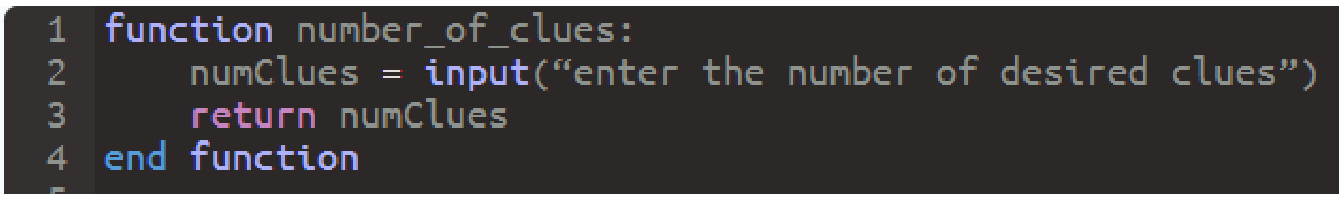
In the next section "Solve", I break it down further into 3 smaller subsections. The first of these is Input where I have to receive the user's input in two ways. One is a standard string input from the user and the other is using an uploaded image. The string input is a basic input function which returns the clues that the user has. For the user to be able to upload an image, the user has to be able to open files within the python program and select the image, and then the image has to be processed and decoded to isolate the numbers from the image. For this, a handwriting recognition artificial intelligence will be used. The following objectives come under the camera_section and refer to criteria 7-11. The artificial intelligence will be trained from a neural network by teaching it from numerous models. After translating the data from the image input, said data should be sent to the computer to be solved in the next section.

The next section, "Solution", is where the solution to the sudoku is devised and produced. First, the sudoku is verified to check if it's a valid sudoku and the clues are valid. After this subroutine, the program uses the backtracking algorithm to begin producing a solution. This relates to success criteria number 12 and 13; where the algorithm is implemented and a solution is built up number by number. Then after a solution is found, the program moves to the next subsection which is output. In this subsection, the program outputs the solution in both a string and grid format for the user. By outputting both formats, the user can read the numbers easily in a grid format, as well as being able to copy and paste the solution if necessary.

The final section is "Check" which allows the user to check the solutions that they have suggested. Although this can be done manually, it would be much more efficient to use the program to check the solution. The input functions will be reused to allow the users to input their completed sudoku and original sudoku to check if all the solutions match and will return if the user's solution is valid or not.

Going into more detail about using the picture input, there are multiple steps that I have to undergo to use the data coming from the picture. First, I load up the picture and then apply the Gaussian Blur and Adaptive Threshold. This separates the foreground and background using the intensity of the pixels and blurs all details in the background. Then invert the colours, so that the grid lines will have non-zero pixel values. Then, the image is dilated and increases the size of the grid lines. Next, find the contours in the image and order this list by size. The largest contour will be at index 0 and then retrieve the coordinates of the contour. Following this, find the longest side and crop the sudoku grid. Using perspective transform, a function from the image library, transform the image into a square. Then the functions get_digit and extract_digits can be used to extract the numbers.

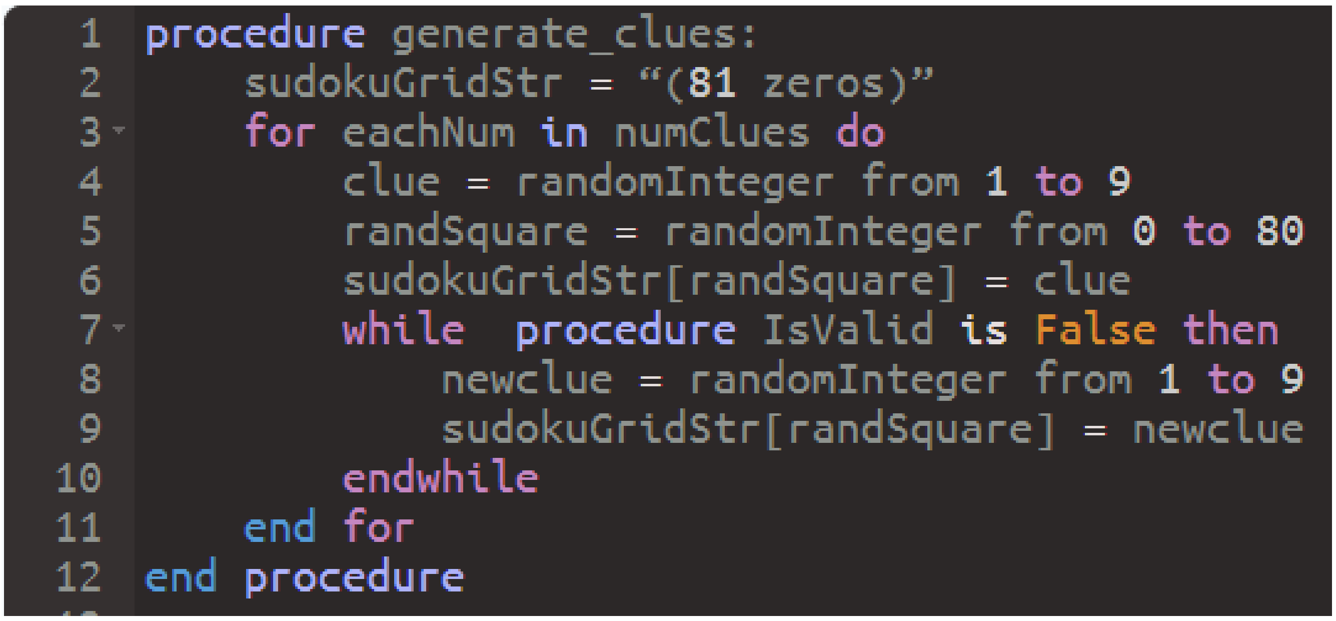
2.3 Pseudocodes



```
function number_of_clues:
    numClues = input(“enter the number of desired clues”)
    return numClues
end function
```

Figure 11: number_of_clues

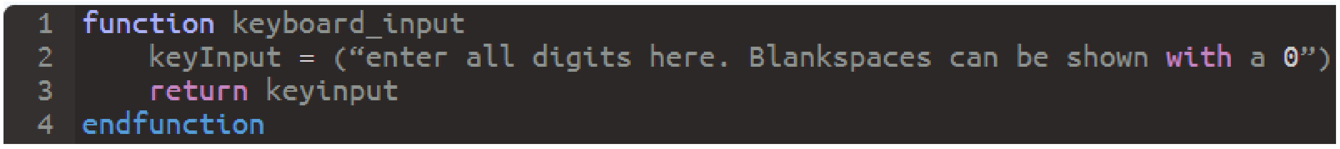
A basic function to allow the user to determine the number of clues that they want. This number must be between 17 and 81 as explained later.



```
procedure generate_clues:
    sudokuGridStr = “(81 zeros)”
    for eachNum in numClues do
        clue = randomInteger from 1 to 9
        randSquare = randomInteger from 0 to 80
        sudokuGridStr[randSquare] = clue
        while  procedure IsValid is False then
            newclue = randomInteger from 1 to 9
            sudokuGridStr[randSquare] = newclue
        endwhile
    end for
end procedure
```

Figure 12: generate_clues

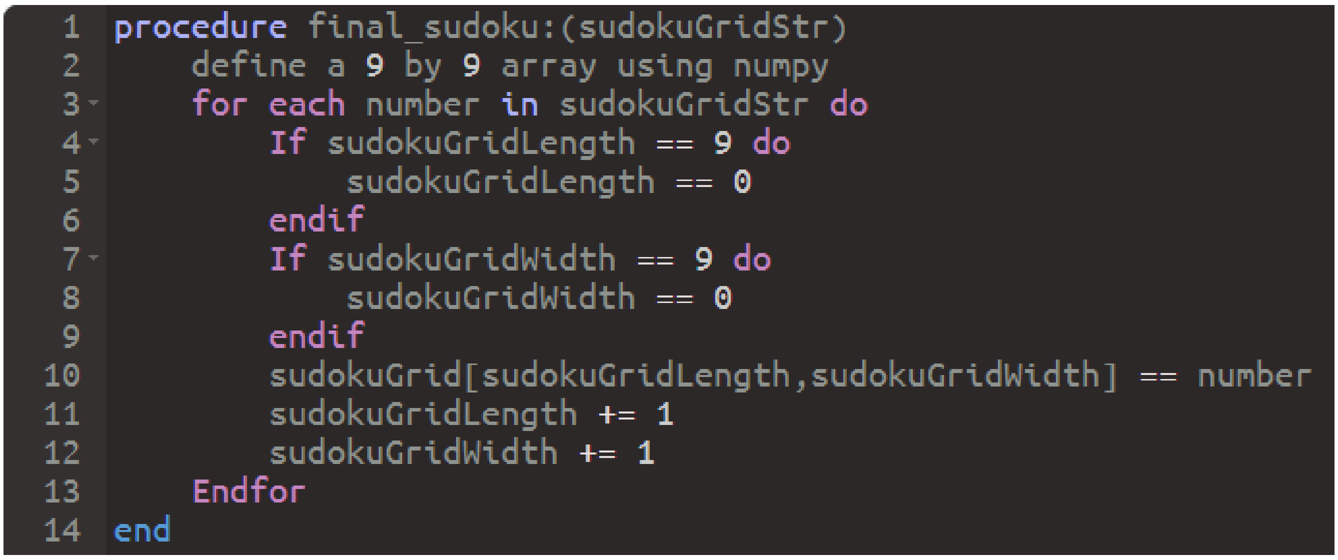
This procedure is called after receiving the user’s input and the variable numClues is used to generate the clues for a new sudoku with the requested number of clues.



```
function keyboard_input
    keyInput = (“enter all digits here. Blankspaces can be shown with a 0”)
    return keyinput
endfunction
```

Figure 13: keyboard_input

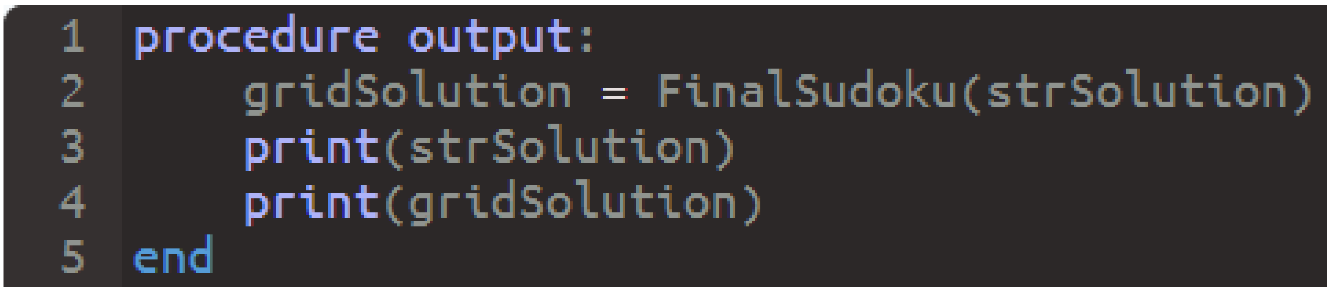
Keyboard_input is used to allow the user to enter the sudoku they want to solve, in a string format. This is then returned to the program to be solved.



```
procedure final_sudoku:(sudokuGridStr)
    define a 9 by 9 array using numpy
    for each number in sudokuGridStr do
        If sudokuGridLength == 9 do
            sudokuGridLength == 0
        endif
        If sudokuGridWidth == 9 do
            sudokuGridWidth == 0
        endif
        sudokuGrid[sudokuGridLength,sudokuGridWidth] == number
        sudokuGridLength += 1
        sudokuGridWidth += 1
    Endfor
end
```

Figure 14: final_sudoku

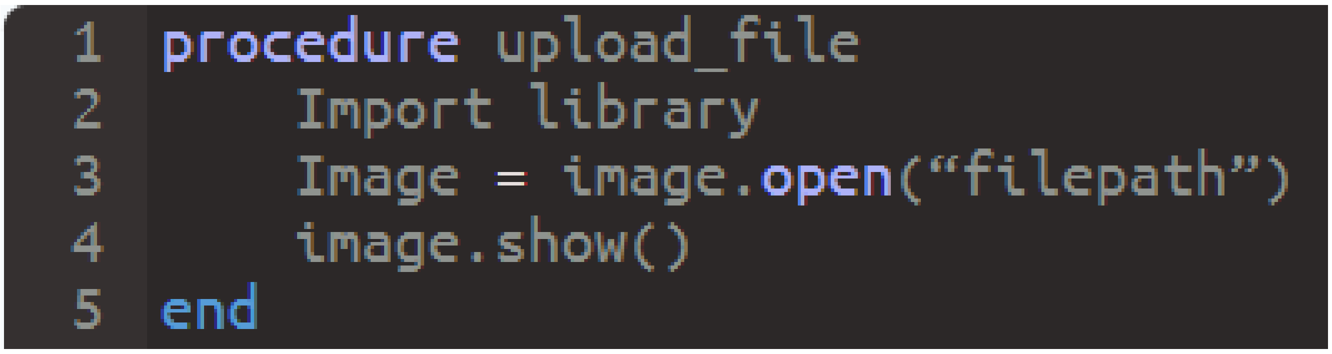
This procedure produces the generated sudoku in a grid format to maximise readability for the user but the string format will also be supplied, to facilitate finding a relevant solution.



```
procedure output:
    gridSolution = FinalSudoku(strSolution)
    print(strSolution)
    print(gridSolution)
end
```

Figure 15: output

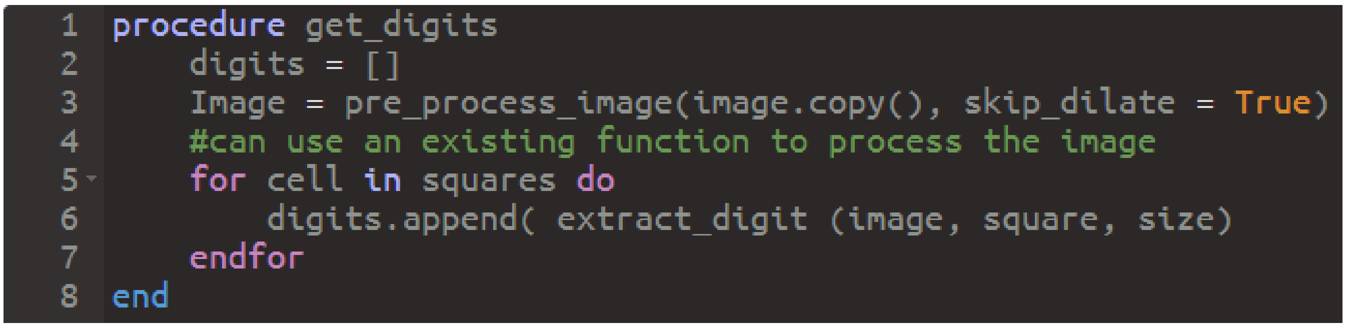
After producing a solution, output prints the solution in a grid format and string format to the user.



```
procedure upload_file
    Import library
    Image = image.open(“filepath”)
    image.show()
end
```

Figure 16: upload_file

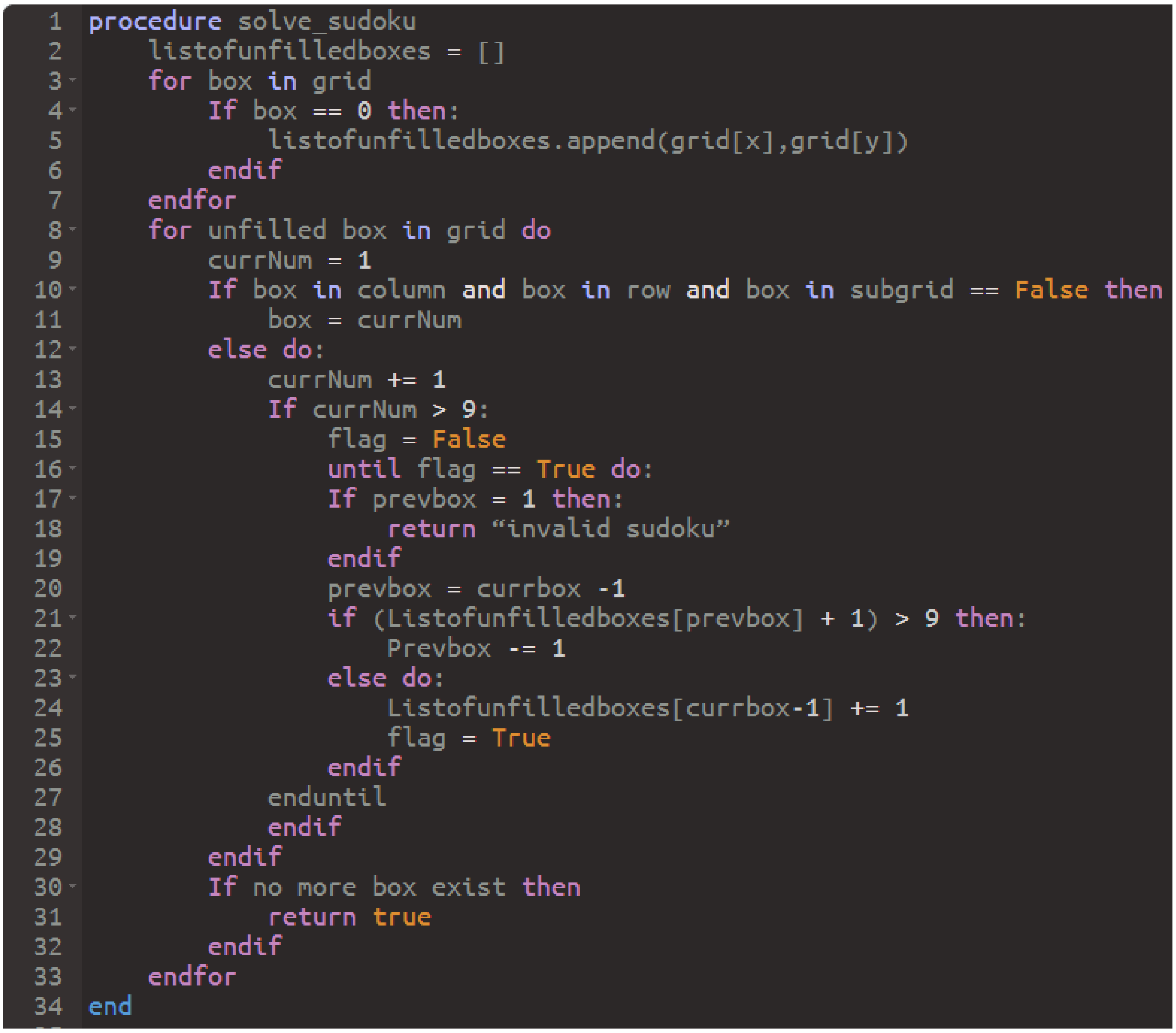
Allows the user to upload an image file in a png or JPEG format to be processed and solved in the other procedures.



```
procedure get_digits
    digits = []
    Image = pre_process_image(image.copy(), skip_dilate = True)
    #can use an existing function to process the image
    for cell in squares do
        digits.append( extract_digit (image, square, size)
    endfor
end
```

Figure 17: get_digits

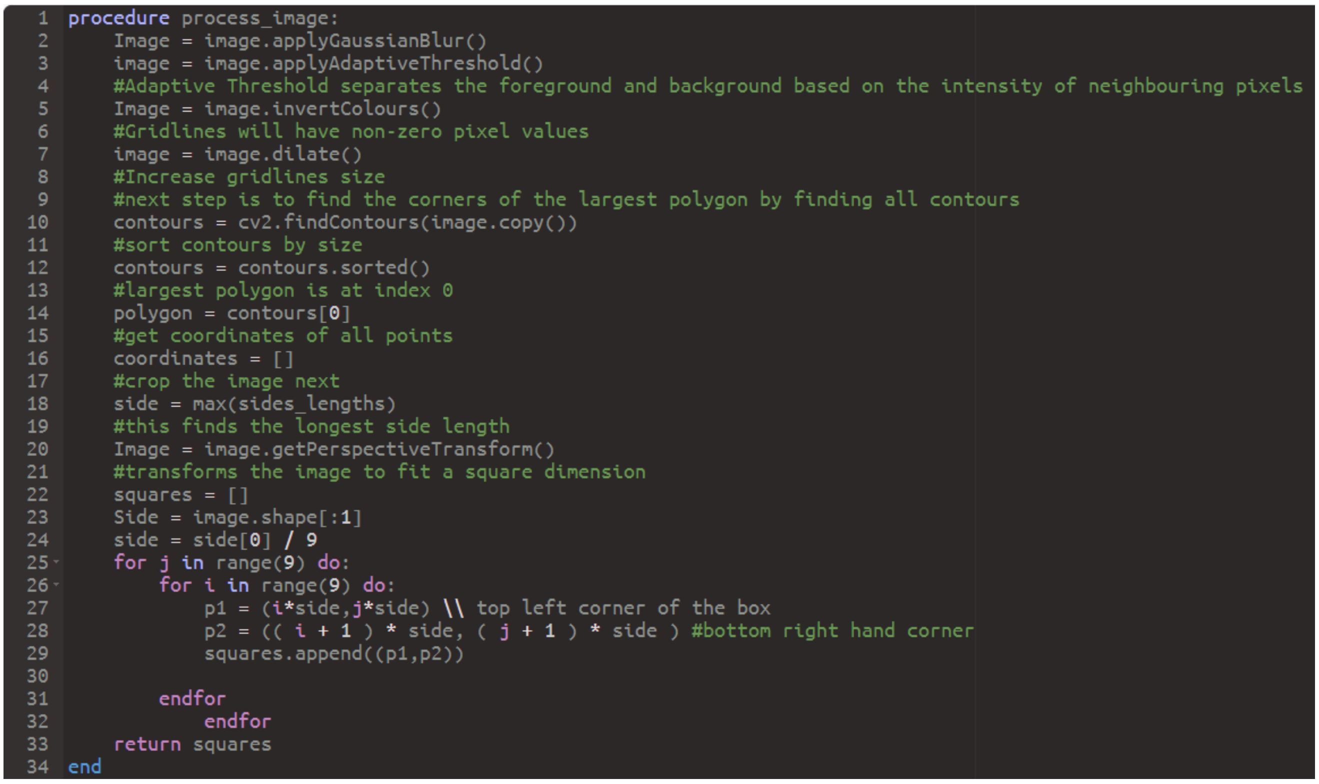
Extracts the digits from the processed image file.



```
procedure solve_sudoku
    listofunfilledboxes = []
    for box in grid
        If box == 0 then:
            listofunfilledboxes.append(grid[x],grid[y])
        endif
    endfor
    for unfilled box in grid do
        currNum = 1
        If box in column and box in row and box in subgrid == False then
            box = currNum
        else do:
            currNum += 1
            If currNum > 9:
                flag = False
                until flag == True do:
                If prevbox = 1 then:
                    return “invalid sudoku”
                endif
                prevbox = currbox -1
                if (Listofunfilledboxes[prevbox] + 1) > 9 then:
                    Prevbox -= 1
                else do:
                    Listofunfilledboxes[currbox-1] += 1
                    flag = True
                endif
            enduntil
            endif
        endif
        If no more box exist then
            return true
        endif
    endfor
end
```

Figure 18: solve_sudoku

The main procedure used to solve the sudoku and return the solution in a string format and grid format to the output procedure.

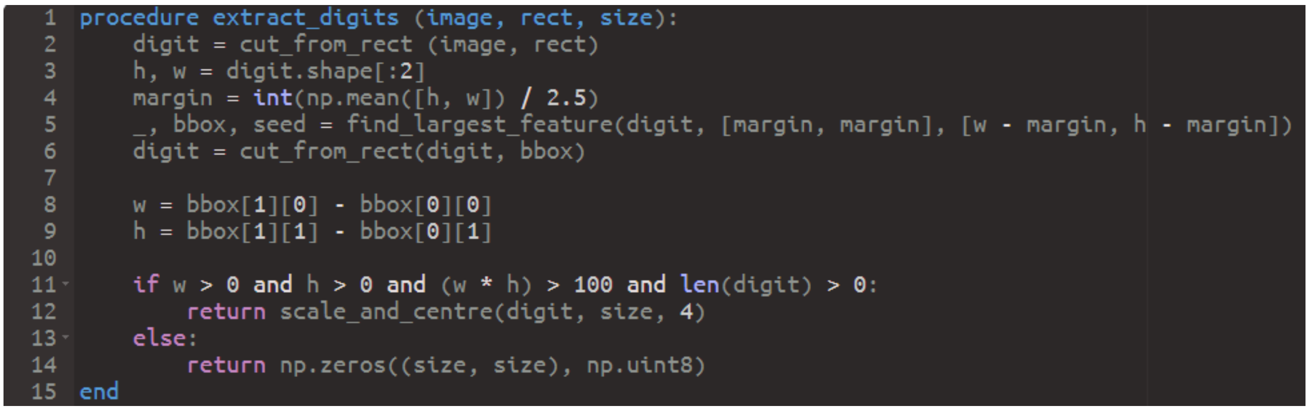


```
procedure process_image:
    Image = image.applyGaussianBlur()
    image = image.applyAdaptiveThreshold()
    #Adaptive Threshold separates the foreground and background based on the intensity of neighbouring pixels
    Image = image.invertColours()
    #Gridlines will have non-zero pixel values
    image = image.dilate()
    #Increase gridlines size
    #next step is to find the corners of the largest polygon by finding all contours
    contours = cv2.findContours(image.copy())
    #sort contours by size
    contours = contours.sorted()
    #largest polygon is at index 0
    polygon = contours[0]
    #get coordinates of all points
    coordinates = []
    #crop the image next
    side = max(sides_lengths)
    #this finds the longest side length
    Image = image.getPerspectiveTransform()
    #transforms the image to fit a square dimension
    squares = []
    Side = image.shape[:1]
    side = side[0] / 9
    for j in range(9) do:
        for i in range(9) do:
            p1 = (i*side,j*side) \\ top left corner of the box
            p2 = (( i + 1 ) * side, ( j + 1 ) * side ) #bottom right hand corner
            squares.append((p1,p2))

        endfor
            endfor
    return squares
end
```

Figure 19: process_image

As mentioned above, the image has to be processed before extracting the digits.



```
procedure extract_digits (image, rect, size):
    digit = cut_from_rect (image, rect)
    h, w = digit.shape[:2]
    margin = int(np.mean([h, w]) / 2.5)
    _, bbox, seed = find_largest_feature(digit, [margin, margin], [w - margin, h - margin])
    digit = cut_from_rect(digit, bbox)

    w = bbox[1][0] - bbox[0][0]
    h = bbox[1][1] - bbox[0][1]

    if w > 0 and h > 0 and (w * h) > 100 and len(digit) > 0:
        return scale_and_centre(digit, size, 4)
    else:
        return np.zeros((size, size), np.uint8)
end
```

Figure 20: extract_digits

This procedure extracts the digits from the processed image and returns the image where the digits are found.

2.4 Algorithms

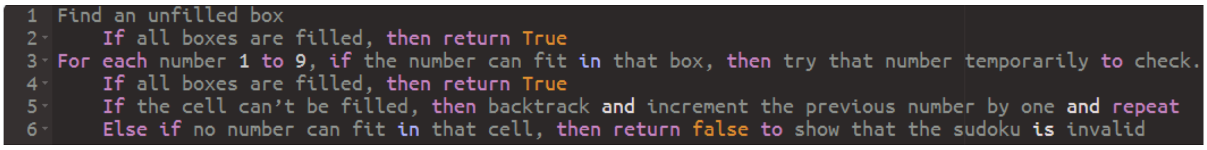
The main algorithm used in solving the sudokus is a backtracking algorithm - a type of brute force approach. Backtracking refers to a technique which is used to check all possible combinations in order to find a solution. The algorithm iterates through each box until the proposed solution is invalid and in which case, the algorithm returns to the previous node and moves horizontally to the next node along. It is a recursive algorithm which builds a solution part by part.

In backtracking algorithms, there are 3 types:

1. Decision Problem - where any feasible solution is found
2. Optimization Problem - where the best solution is found
3. Enumeration Problem- all solutions are found

In the case of the sudoku solver, the problem is a decision problem where only one solution should exist and the backtracking algorithm will find the first feasible solution. Given enough time, the algorithm will complete an exhaustive search to find any possible solution but the algorithm isn't particularly efficient.

Below there are loosely structured steps of the backtracking algorithm to aid me in my explanation of how it works.



```
1 Find an unfilled box
2     If all boxes are filled, then return True
3 For each number 1 to 9, if the number can fit in that box, then try that number temporarily to check.
4     If all boxes are filled, then return True
5     If the cell can't be filled, then backtrack and increment the previous number by one and repeat
6     Else if no number can fit in that cell, then return false to show that the sudoku is invalid
```

Figure 21: Backtracking algorithm basic pseudo-code

Step 1 checks if there are any unfilled cells or the algorithm will deduce that the sudoku is complete. It also skips over any filled cells which avoids making any changes to the existing clues. Then, numbers are placed one at a time in vacant cells and tested to see if they are candidates. This process is continued until a match is not found and in which case, the candidate is incremented by 1. Once all the possible candidates have been tested, the previous cell is incremented by 1, and the testing resumes. This process is repeated until all cells have been filled, or there's no valid number to fit in the cell and in which case, the sudoku is invalid. After placing each candidate in its own cell, the candidate is compared to the row, column and sub grid to check whether there's a repeated digit. If so, the candidate is invalid and is updated. Whilst backtracking may seem like a brute force algorithm, it only checks possible candidates and so, all numbers will not be tested as a brute force algorithm would have. The time complexity of said algorithm is

$$O(K^n)$$

where k is the number of times that the algorithm calls itself.

Other algorithms include Crook's Pen and Paper algorithm and although it is not a well-known computing algorithm like the backtracking algorithm, it is a rule-based algorithm specific to solving sudokus. It's more of a combination of methods which are used in turn to solve the sudoku. These involve a simple solving algorithm, the place-finding method, the candidate-checking method, and the method of preemptive sets. Each of these methods is used to simplify the problem further and ultimately reduce the amount of unnecessary numbers being tested. However, from a computational perspective, it's harder to code such an algorithm and strictly speaking, it is not an unique algorithm but rather a series of other methods.

Another algorithm I considered was the brute force algorithm where every possibility is tested based on the clues given and this also includes testing possibilities when certain candidates are invalid. This is a very slow process and there's no logic behind it, except that there must be a solution so every possibility will be tested until said solution is found. Sooner or later, a solution will be found so it fulfils our need to solve the sudoku, and the implementation is very easy but it is neither very efficient nor does it demonstrate my coding ability or ability to understand difficult algorithms.

Constraint programming is the name of another programming technique which is a form of declarative programming. Constraint is a mathematical term used to describe a relationship between variables. In the sense of making a sudoku solver, this means that variables in the same column, row and subgrid cannot be the same. Whilst there's more to constraint programming, this is the basis of it and the programmer details how all variables link together and this can then be used to solve the sudoku much more quickly because there's only a finite number of possibilities where the variables in each column, row and subgrid are distinct.

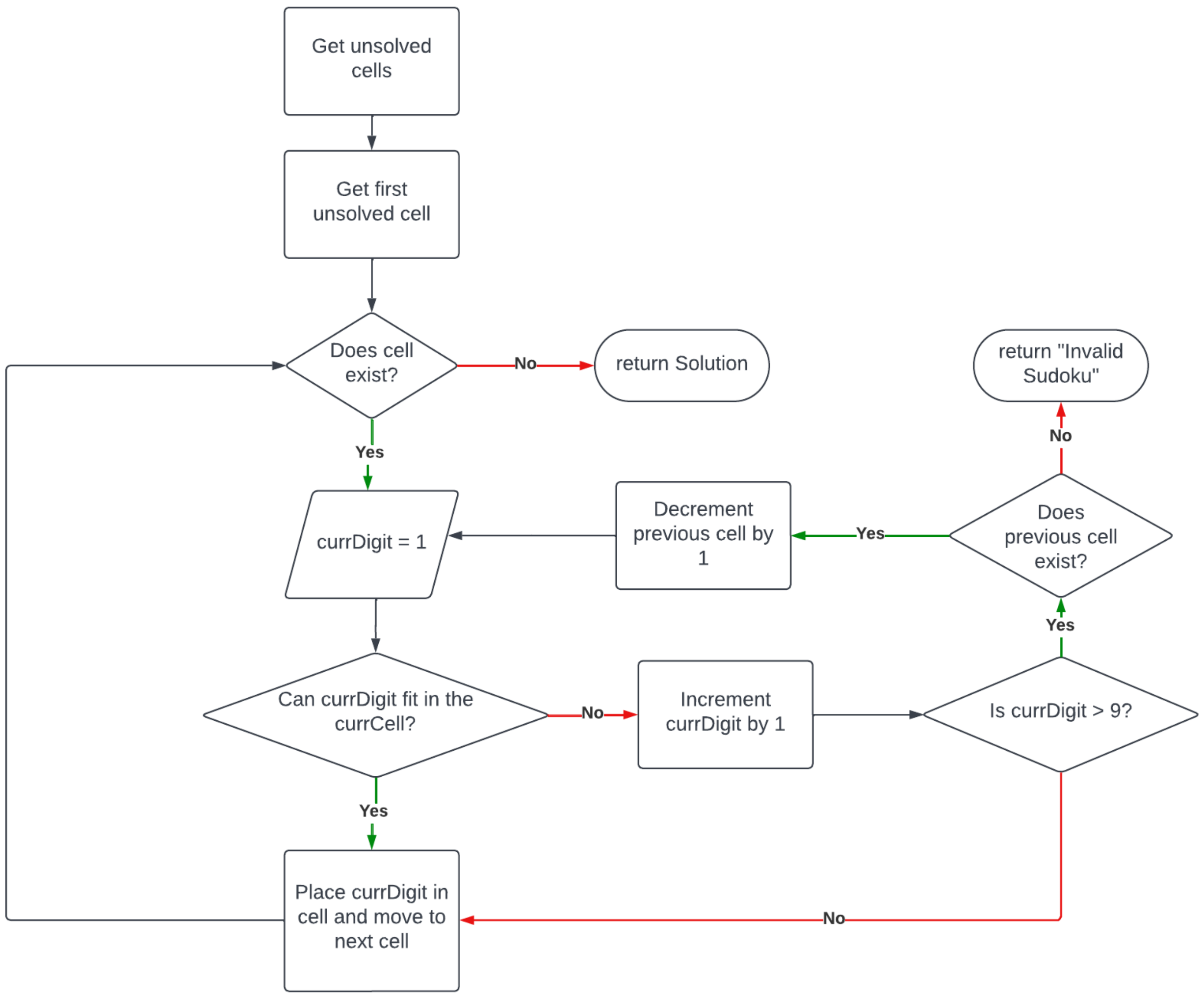


Figure 22: Backtracking algorithm flowchart

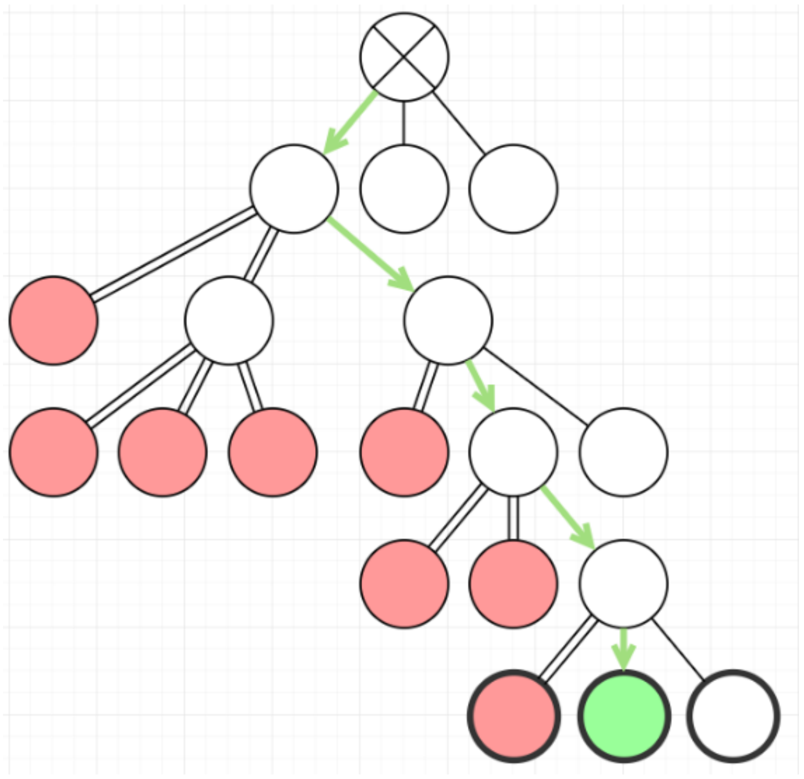


Figure 23: Backtracking algorithm flowchart

It's also possible to use a tree of possibilities to understand how backtracking works. We begin at the starting node and begin searching each node, until we meet an incorrect solution. This is marked in red on the diagram. We backtrack along the path that we just followed and return to the previous node and this is indicated on the diagram by a double line. We continue to the second node down from the starting node, and we see that all the possible solutions are incorrect and therefore, that parent node will not provide a valid solution. Similar to a depth first search, the algorithm moves across to the next node and begins searching all the possible nodes until a solution is found. This is shown by the green shaded node and the green arrows show the solution path. As we can see, the algorithm searches all nodes for a possible solution until a solution is found in which case the solution is returned, or no solution is found and the computer returns this. In the case of our sudoku, each unfilled cell can be represented by each level and each node represents all the possibilities for the unfilled cell.

2.5 Usability Features

I have designed a template for my program's menu below while considering the user's needs at the heart of it. Although it is a simple design, the program is targeted towards all and as such, the minimal interface facilitates navigation and use of the program without too much difficulty for all users. There's a consistent layout of buttons and colour scheme for different objects. The font size is large enough to be read even on smaller devices such as mobile phones without being too intrusive or overfilling the page. The larger text on the screen ensures that user's do not struggle with seeing the difference between similar numbers on the screen and makes the program more user-friendly. The purple lettering on the blue background is used to outline the letters more clearly and makes it stand out more. Also, the orange and green backgrounds distinguish input boxes and buttons from the light blue background. In further maintenance and development of the program, I would allocate some time to improve user customisation such as changing colours of buttons and letters to suit the user's needs. A proposed thought was to use curved boxes rather than typical boxes, but I felt that it would take away from the minimalist design and would also reduce the sharpness of objects and make them stand out less. Another idea was to make the buttons and text larger than the current size, to make it more readable for those with weaker eyesight, but ultimately, this would look unprofessional and adding an option to zoom in would allow users to zoom in and out as they desired, rather than forcing a decision on all users.

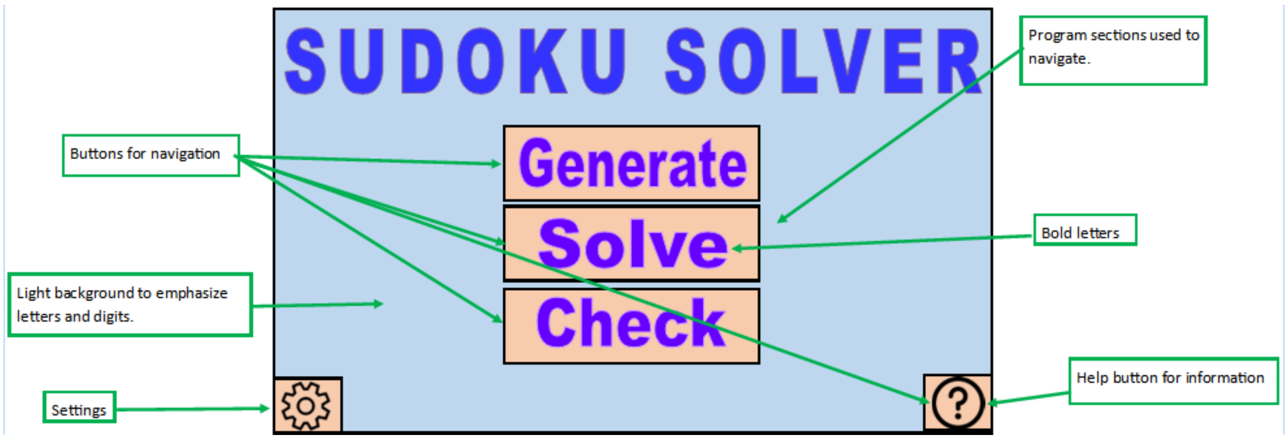

SUDOKU SOLVER
Generate
Solve
Check
Program sections used to navigate.
Buttons for navigation
Bold letters
Light background to emphasize letters and digits.
Help button for information
Settings

Figure 24: Menu Homepage

First, I have a homepage for the program which is what the user sees upon starting the program. The light colour palette has been used to improve clarity of words and letters. The most common type of colour blindness is red-green and as such, the blue scheme should make the program accessible to as many as possible. In the case that the user has tritanopia, the inability to perceive blue light, I considered adding an customisation option, where the user has control over the colour scheming. However, this would be an added feature on top of the key functionality of the program. The settings button in the bottom left would host any settings the user may want to configure. Also on the homepage, there's a help menu button which can be seen across all the pages, and it has a brief help message about how to use the program. This allows new users to easily familiarise themselves with the program. At the centre of the screen, the 3 main features are listed as buttons to navigate through the program.

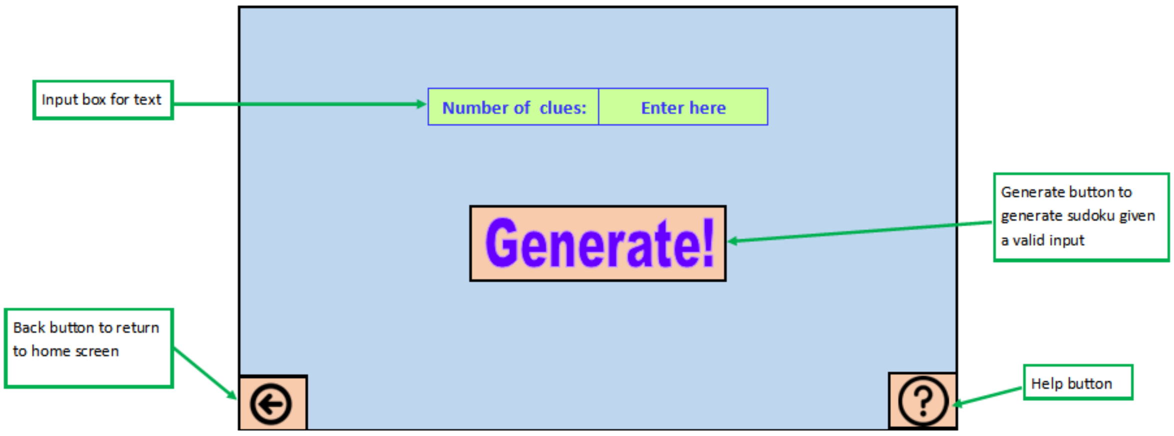

Input box for text
Number of clues:
Enter here
Generate!
Generate button to generate sudoku given a valid input
Back button to return to home screen
Help button

Figure 25: Generate page

Upon clicking the generate button, user is brought to a page which allows the user to generate sudoku. There's an input box in light green to contrast with the blue background and the dark blue text reads "number of clues:" and then on the other half, "enter here". This allows the user to enter the number of clues that they want, and this input will be used to generate a new sudoku upon clicking the generate button. To help simplify the program further, I have used the same colours for the same objects such as text boxes and buttons. The settings icon is replaced by a back button, which returns the user to the homepage.

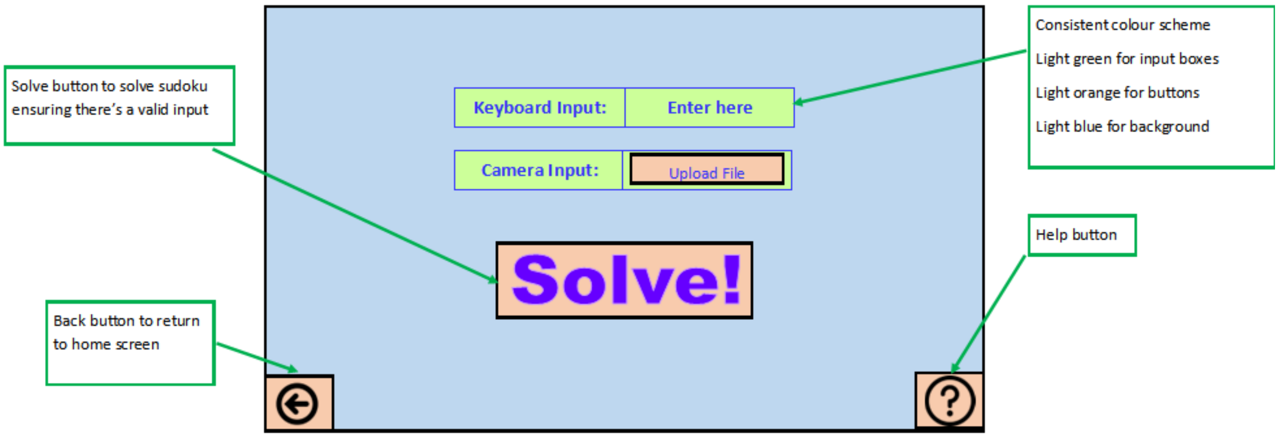

Solve button to solve sudoku ensuring there's a valid input
Back button to return to home screen
Keyboard Input:
Enter here
Camera Input:
Upload File
Solve!
Consistent colour scheme
Light green for input boxes
Light orange for buttons
Light blue for background
Help button

Figure 26: Solve page

Moving onto the solve page, the user is confronted by a couple of options this time, where they can input their sudoku using a keyboard or with an image file. The top box is a simple input box to enter the clues as a string input. The button below reads "Upload file" where they can upload the image file onto the program to be processed and extract the digits. The pale orange box has been used once again, to show that it's a button which can be clicked. After uploading the clues as necessary, the inputs are validated and the user can click the solve button to receive a solution. If the input isn't valid, then the user will be notified with a message on the screen.

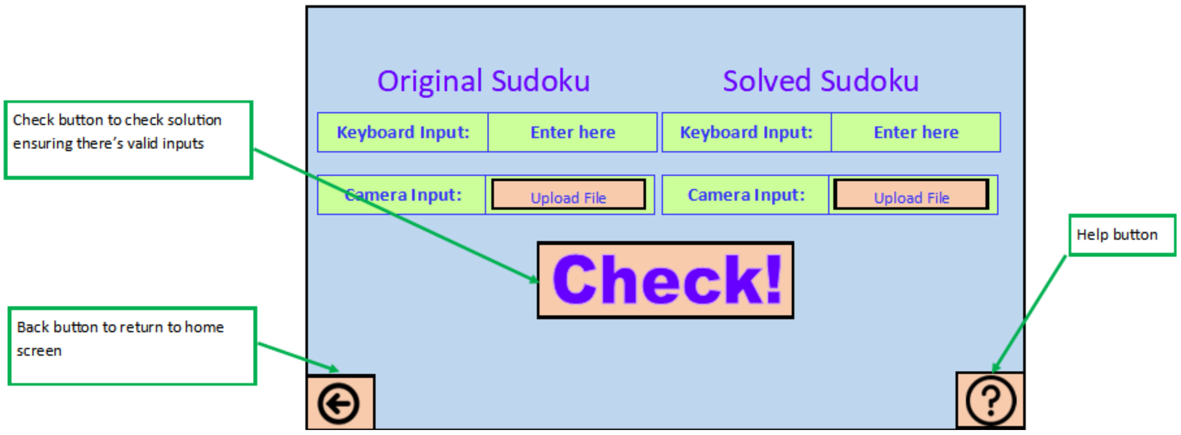


Figure 27: Check page

The final page is the check page, which has two sets of input choices. For the check section, the user is required to upload the original sudoku without them having completed anything on it and this is used to produce a solution. The user also uploads their solved sudoku in either input and the program compares both sudokus and returns if the filled in sudoku is correct. This requires two valid inputs from the user in either a string or image format.

2.5.1 Stakeholder Input

Following the design of my menu, I encountered my stakeholder again to gain an insight in to his views. The following transcript is between me and Alfred, who is reflecting on my menu design and suggesting any improvements I could make.

Q:What do you think of the colour scheming? Does it sharpen the numbers?
A: Yeah I think that the combination of a light background with strong, dark colours for text really makes the text stand out and sharpens the outlines of numbers. This is definitely ideal for daytime use but I would say that the lighter colours may be too strong later on in the day

when it's darker. Perhaps you could rectify this with a dark mode? Or it could be something you introduce in an update. But overall, the simple interface and the colour scheme has been well thought and I think it works well.

Q:How do you think curved boxes would have changed the appearance of the program?
A: Well, perhaps it would give the program a more refined, modern look but I would say that curves take away from the sharp look and would reduce the simple interface that we are currently presented with. Also, the most important thing for users is the clarity of the numbers and this is supported by the sharp corners which outline buttons clearly. If the objects were to become rounded, then it should be consistent and this should include the back button and help button.

Q:Would an option to change colours be beneficial for you?
A: For me personally, I'm not particularly bothered by the colours in the program, but I guess that it would be helpful to others to have that feature at their disposal. This could include colour blind people or simply those who prefer to customise their programs.

Q:Are there any other features you would like to see implemented in the long term?
A:Hmm, perhaps improving the check feature to be able to recognise different colours, which means that users can complete the sudoku in a different colour and then be able to check the sudoku. This saves the user taking two pictures before and after, or having to type in the existing clues before hand. Another feature that I would be interested in is solving larger squares, such as 4*4 squares. This would be a little harder to implement, but I think that it would be a rare feature found on not many sudoku solvers.

After hearing this interview, I have a few more extensions that I could complete to further improve the functionality of my program. But focusing on the current objectives, Alfred says that the curved boxes would create a more sleek and modern design. Thus, I will try to implement this into my program. He did mention that it would draw away from the clarity of the numbers, but if I outline the numbers with a different complimentary colour, then I can maintain the clarity whilst improving the appearance. Also, he says that a customisation feature isn't too important for him personally, but implementing this feature make my program more accessible. Another improvement Alfred suggested was the addition of a dark mode, which I had not thought of before. I would certainly be looking to add this because it is not too difficult and is a very popular feature amongst numerous popular apps these days. Overall, this interview has highlighted some new features and also some improvements which I can be thinking about while I begin my development.

2.6 Validation

Validation is important to ensure that the program can function properly and not return errors due to false or invalid inputs. This validation will happen at various points during the program to mitigate any errors. All inputs should be validated to check if it complies with the program's needs. For instance, whilst inputting the sudoku clues as a string, there must be 81 digits, else the digits will be misaligned with the corresponding cells. This could result in an invalid sudoku, or the wrong solution being returned to the user. In some circumstances, the user may have to validate the input because the computer has no way to do so. In particular, one instance would be when the user chooses to input via the camera input option, and the user will have to validate if the computer has recognized characters properly. After presenting the interpreted input to the solution, the user will have to confirm that this input is correct with the use of a button. But also the speed of the program should be considered, and an increased number of validation steps would slow the program down. Therefore, I will have to make an informed decision about which variables must be validated to produce a functioning program without compromising too much on the speed of the program.

List of validation data			
Validation Data	Why?	How?	Where?
numClues	Has to be between 17 and 81	numClues >16 and numClues < 81	numberOfClues function
keyboardInput	Only contain numbers, must have 81 digits	isNummeric()	Keyboard Input function
cameraInput	Must contain square with numbers. Also user check if recognised digits are correct	Output interpreted grid to user	cameraInput function

Here the table above shows the 3 different inputs in my program which have to be validated. Firstly, in the generate section of the program, the user is given the chance to determine how many clues they desire, and this number must be between 17 and 81 where 81 would result in a completed sudoku being outputted. The lower limit of 17 comes from a breakthrough by Gary McGuire of University College Dublin. He and his team spent 7 million CPU hours and a brute force approach to deduce that any sudoku with 16 or fewer clues would have multiple solutions and sudokus are only intended to have one valid solution.

The next input would be in the solve section of the program where the user inputs their clues via string or camera input and both of these would have to be validated to ensure that the program solves the relevant sudoku. The string input is much easier to validate by checking if all the characters are integers firstly, and then checking that len(stringInput) < 81. Here there is no lower threshold of 17, as users are attempting to find a solution for the existing sudoku that they have, and although it's invalid, a solution can be formulated and returned to the user. It's true that the outputted solution may not match the user's solution but that doesn't mean that the user is incorrect as invalid sudokus can have multiple valid solutions.

Finally, the uploaded image file has to be verified to ensure that the AI has recognised the correct characters and interpreted the data correctly. For example, the digits have to be placed in the appropriate boxes to solve the correct sudoku. This can be validated by the user who is also given the chance to update any of the digits before being solved by the algorithm. I will display the interpreted sudoku problem on the screen and then allow the user to change certain values if the handwriting recognition AI had misinterpreted the number.

2.7 Key Data Structures

List of subroutines		
Subroutine	Purpose	Any variables returned?
number_of_clues	Take an input from the user which determines the number of clues they want in their sudoku	numClues
generate_clues	Generate the sudoku with the given number of clues	No
final _sudoku	Output the generated sudoku to the user in a sudoku format	No
keyboard_input	Takes an input for the sudoku they want to solve	strinput
upload_file	Takes an image input for the sudoku they want to solve	imageInput
translate_data	Converts the data from the image into a solvable sudoku grid	sudokuStr
is_valid	Checks if the sudoku input is a valid sudoku	True/False
solution	Outputs the solution in a grid and string format	No

All these variables will be defined locally and returned to the main routine when needed. For example, when numClues is needed, the subroutine number_of_clues will be called and the value will be returned to then be used. By using a modular approach, I can test modules more easily and it's easier to find bugs and make changes to the program. An alternative method would be to use classes and define functions in those classes to be used, however, there are not many variables which are common to multiple subroutines. Thus, there won't be many attributes in the classes and it would be useless to use a class approach.

2.8 Testing method

Whilst coding, I will be testing each module to ensure that the module functions correctly. This will be done in the form of white box testing to confirm that the code is optimised, and also, the test cases can be targeted to certain parts of the program. Following the testing, I will fix any bugs and improve the program and this will be an iterative testing process. The majority of the testing will be post development and done in the form of black box testing, to imitate how the user would use the program. Black box testing focuses on the functionality of the code which is the most important thing to the end user. The testing will consist of all types of test data such as normal, boundary and erroneous to simulate a user interacting with the program. The results can be used to refine the program and improve any features that are not functioning properly which comes under the maintenance section of the SDLC. First it's important to test the key components of the program which are essential for the functionality of the whole program. This includes fundamental functions such as interacting with the GUI, inputting values and even solving the sudoku to get a solution.

Test Data Table			
Test Data	Subroutine	Test Data Type	Expected Result
16	number_of_clues	Invalid	Error message to user
17	number_of_clues	Boundary	Sudoku Generated
20	number_of_clues	Valid	Sudoku Generated
81	number_of_clues	Boundary	Sudoku generated (Would be completed already)
82	number_of_clues	Invalid	Error message to user
Press generate button	generate_clues	Valid (given appropriate input)	Sudoku should be generated and outputted in grid format with given input
10 digits inputted	keyboard_input	Invalid	Error message to user
81 digits inputted	keyboard_input	Valid	Accepted input
82 digits inputted	keyboard_input	Invalid	Error message to user
81 digits with invalid character such as a letter	keyboard_input	Invalid	Error message to user
Png image file	upload_file and get_digits	Valid	Recognised sudoku board displayed to user
Png image file with no sudoku board	upload_file and get_digits	Invalid	Error message to user
jpeg image file	upload_file and get_digits	Valid	Recognised sudoku board displayed to user
png image file with illegible numbers	upload_file and get_digits	Boundary	Recognised sudoku board displayed to user
Press solve button	solve_sudoku	Valid(with a valid input)	Correct solution
Press check button	check_sudoku	Valid(with valid inputs)	Correct/Incorrect depending on if the completed sudoku is correct/incorrect

A few tests here are worth exploring, especially the boundary test data ones. For instance, inputting 17 clues into the generate_sudoku will fulfil the condition of >16 and a valid sudoku with only one solution should be formulated according to the research of University College Dublin. Also, when requesting the program for a program with 81 clues, it should output a filled in sudoku which should also be correct. Furthermore, the upload_file subroutine should accept JPEG and png files, which are the two most common image file formats, and process both images similarly. However, there may be some bugs that arise due to the different formats and these should be rectified. In the eyes of the user, they may want to upload the image file in a JPEG or png format or even other image formats, but JPEG and png are the two most common formats. This ensures that the user experience is not compromised and requires them to convert formats and improves the versatility of the program. Also, the accuracy of the handwriting recognition artificial intelligence should be tested after being trained. Whilst an increased duration of testing would certainly improve the accuracy of the model, the time constraints of this project limits me from doing so and as I mentioned earlier, these small improvements can be part of the maintenance section after development.

2.9 Development Methodology

An iterative development methodology such as an agile development process will be used to develop this program to design multiple prototypes and improve between iterations by finding bugs and completing some testing to test functionality of the program. It's unlikely that the program will be perfect upon its initial completion, however, the agile approach allows me to gain frequent feedback from my consumer and also from my testing, and both of these can help refine the program. Sections of the program can be produced in sprints and later, tested according to the test data table. An alternative methodology that I considered was the waterfall methodology, but it is quite outdated and is designed for projects with clear requirements which cannot change, while my program's may change to suit the user's needs. Also, waterfall methodology produces a working solution and doesn't involve the user whereas, the agile approach works in close collaboration with a user to ensure that the program fulfils the user's needs.

3 Developing the Solution

3.1 Sprint 1: Generate sudokus

3.1.1 Improvement 1

3.1.2 Error 1

3.2 Sprint 2: Solve sudokus

3.2.1 Cell validation

3.3 Interview 2: Feedback on sprint 1 and 2

3.4 Sprint 3: Digit extraction using a NN

3.4.1 Process image

3.5 Sprint 4: Main file

4 Evaluation

4.1 Testing to inform evaluation

4.2 Success of the solution

4.3 Describe the final product

4.4 Maintenance and development

5 Bibliography

References

[] *What is an NP-complete in computer science?* `https://stackoverflow.com/questions/210829/what-is-an-np-complete-in-computer-science`. Accessed: 2022-09-29.