OCR Sudoku Solver Specifications

Table of Contents

I. Framework

The project is to be carried out in groups of four (and only four). Its duration is about four months (from September to December).

This document presents the project goals, its parts and the expected completion schedule. It also presents constraints.

II. Schedule and Deliverables

- **Friday, September 17, 2021**: group submission deadline
- **Week of October 25, 2021**: first defense (late afternoon & weeknight)
- **Week of December 6, 2021**: second defense (late afternoon & weeknight)

III. Subject

1. General Specifications

Your goal is to build an O.C.R. software (*Optical Character Recognition*) the main function of which solves a Sudoku grid.

Your application will thus read an image, that represents a Sudoku grid, as and will produce the solved grid as output.

In its final version, your application will provide a graphical user interface allowing the user to load an image in a standard format, visualize it, correct some of its defects, and finally to display the completely filled and solved grid. The resolved grid should also be able to be saved.

Your application will use a machine learning algorithm, which can be a distinct program, for training a neural network and loading and saving learned parameters of the network.

2. Detailed Specifications

2.1. Main Stages

The processing of your request will be approximately as follows:
- Image loading
- Color removal (gray-scale, then black/white)
- Pre-processing
- Grid detection
- Detection of grid cells
- Retrieval of the digits present in the cells
- Character recognition (the digits)

- Reconstruction of the grid
- Grid resolution
- Display of the solved grid
- Saving the solved grid

Note that no color information is required for character recognition and in fact, working in black and white is almost mandatory.

2.2. Image Splitting

Splitting the image is necessary to send the pieces of images corresponding to digits to the recognition. It will, therefore, be necessary to identify the position of the boxes of the grid in order to extract the figures in the form of an image and also to determine the empty boxes.

2.3. Character Recognition (Neural Network)

Recognition relies on a training stage where your network will learn how to discriminate characters.

A neural network is tool that can learn a transformation function (non-linear) using training samples: during training, you will provide labeled inputs (image block with the corresponding character) and your network will answer classification probabilities. You will use the error margin to correct the internal stages of the network using gradient retroprogation. Using a sufficient number of samples, your network will, most of the time, provide the correct answers. The seminal output for that kind of network is a vector of probabilities for each character. The character with the highest probability will be the recognized character.

In order to implement your neural networks, here are some keywords to look up: ***supervised machine learning, multi-layer perceptron, retroprogation, gradient optimization, AdaBoost.***

The website http://neuralnetworksanddeeplearning.com provides a lot of theoretical and practical information on neural networks and could be useful for your task (the third chapter deals with networks tuning, which is often a difficult part).

There exist many kinds of neural networks, but for this project, you will probably need only a simple MLP with a single hidden layer. Anyway, it can be worth considering a softmax activation function (for the last layer) rather than the usual sigmoid function. You will find everything you need in the previous link.

2.4. Pre-Processing

This stage aims at enhancing image quality. This part will allow your program to work with scanned documents or pictures. Here is the pre-processing your project must be able to do:

- Deskew manually (rotate the image from an angle entered by the user)
- Deskew automatically (with skew angle detection and rotation)
- Noise canceling (image grain, stains, etc.)
- Contrast enhancement

2.5. Sudoku Grid Resolution

You will need to implement a grid sudoku solving algorithm in C language. To test this algorithm, you will need to design a small program called **`solver`**. It will be an application that will run from the command line only. It will take as input a file name in a format described in the appendix. It will solve the sudoku grid and will generate a file containing the result. The name of the output file will be the name of the input file with the addition of the extension ".result". The format of the output file will be the same as that of the input file.

For instance:

```
$ ls
grid_00 grid_01 grid_02 solver
$ ./solver grid_00
$ ls
grid_00 grid_00.result grid_01 grid_02 solver
```

2.6. Solved Grid Display Format

The choice of the display format of the solved grid is up to you. **However, the quality of the display will be taken into account in the evaluation.**

It is, therefore, strongly recommended to display the result as an image rather than as simple textual data. In the case of an image, it should be savable in a standard image format.

It is also preferable that the boxes filled in by the algorithm (those which were initially empty) display a different color from those initially filled.

IV. Project Constraints

1. Technical Constraints

You must respect the following rules:

- Your project must be implemented in the C programming language
- It must compile and run using the school-provided environment (Linux)
- During the defenses, the demonstrations will be done on the school machines
- All software installed on school machines may be used for the project
- Your code must compile without any warning, using at least the following options : -Wall -Wextra
- Lines of code must not exceed 80 columns and must not contain trailing white-spaces
- All identifiers and comments must be in English
- Your deposit must contain **only** the files specified in the section *V.3 The Contents of the GIT Repository*
- Your repository must not contain:
  - Executable files
  - External source code (libraries, other projects, etc.)
  - Any other type of file not necessary for the project

2. Version Control and Submissions

For this project, you will have a *git* repository usable by all group members. We strongly encourage you to use all the possibilities of version control in *git*.

Before each defense, you must provide the code of your project through your *git* repository. You will hand in the project using *git*: we will clone your repository on the main branch (master) on the scheduled day. **Only the files present in your repository when we clone it will be available during the defense.**

V. Deliverables

1. The Defense Plan

A defense plan will be provided at the start of each defense. It will be in paper format and must fit on an A4 sheet.

2. The Report in Hard Copy

Just like the defense plan, a report presenting your project must be printed. It must be given on the day of the defense.

It must, among other things, present:

- Your group
- The task distribution
- The progress of the project
- The technical aspects

3. The Contents of the GIT Repository

3.1. The Report in Digital Version

A digital version of your report should also be present in your GIT repository. It must be in PDF format and located at the root of your repository. The file must have the following name:

- **`rapport_1.pdf`** for the first defense (15 pages minimum)
- **`rapport_2.pdf`** for the final defense (40 pages minimum)

Please note that the retrieval of reports from the GIT repository will be automated. It is, therefore, important to respect the above file names, otherwise your report will not be evaluated..

3.2. The Code

Your GIT repository must contain your complete source code and a **`Makefile`** file. The latter must provide at least the *all* and *clean* rules.

3.3. The README and AUTHORS Files

Your GIT repository must also contain:

- A **README** file concisely describing how to use your application. This file should be in text format and correctly readable in a terminal with a command like *cat* or *less*. It could possibly be in *markdown* format.
- An **AUTHORS** file containing the logins of the members of your group. It must contain one line per member of the group in the following form: login (LAST NAME First name).

3.4. The Pictures

Your GIT repository should also contain some pictures that can be used for demonstrations of your different parts. Be careful, do not overload your repository with a large number of very high resolution pictures. Three or four images at the correct resolution should suffice.

You will also need a picture set for training your neural network.

VI. The Presentations

1. General Structure

Each presentation should include an introduction, conclusion, and a demonstration of your project on the expected features described in this document. Since you're all doing the same project, it won't be necessary to introduce the project nor to rephrase the current document.

The first defense will be a progress check. Your goal is to demonstrate the current state of your project. You will also present how your neural network works. Take care in presenting how your own implementation works. Do not give a lecture on neural networks.

For the final defense, you must present a complete standalone program. It is important that this program perform all the expected flow of actions. Be sure to produce a real program and not a set of unconnected pieces of code, even if it only works on a small set of examples.

2. Project Stages and Technical Deliverables

Your project is divided in several parts. A part is said to be completed if your program is able to perform the expected task on significant examples. Of course, the complete evaluation of the part will depend on the overall quality and the robustness with regards to advanced testing.

Mandatory parts for the first defense:

- Image loading and color removal
- Manual rotation of the image
- Detection of the grid and the grid cell positions
- Splitting of the picture (saving each box as an image)
- Implementation of the sudoku grid solving algorithm. You will have to implement this algorithm in the command line program **`solver`** (see *III.2.5 Sudoku Grid Resolution)*
- A proof of concept of your neural network. For this proof, you will realize a small network able to learn the XOR function.

Parts that must have been started for the first defense:

- Save and load of neural network's weights
- Set of pictures for learning
- File manipulations for result saving

Parts that must be completed and presented for the final defense:

- The complete pre-processing
- The complete and working neural network:
  - Learning
  - Recognition of numbers on the grid
- The reconstruction of the grid
- The grid resolution
- The display of the grid
- The saving of the result
- A graphical interface allowing you to use all these elements

Optional parts (only considered if mandatory parts are done)

- Recognition of handwritten numbers
- Hexadoku grid resolution (a 16x16 grid with hexadecimal notation)
- Website presenting your project
- Extras

VII. Tips and Tricks

This project is designed for a team of four students over 3 months. Some parts, like the neural network, require adjusting and sometimes time consuming computations. **It is important to start working on the project as soon as your group is created, otherwise you will probably have difficulty finishing the project correctly on schedule.**

You may find it useful to design independent tests for each part. For example, the neural network consumes, as input, images of characters already extracted and scaled. In order to test and train your neural network, it would be useful to have a set of tests where the images are already single characters in the correct format. So even if your splitting of the image is not ready, you will be able to move this part forward.

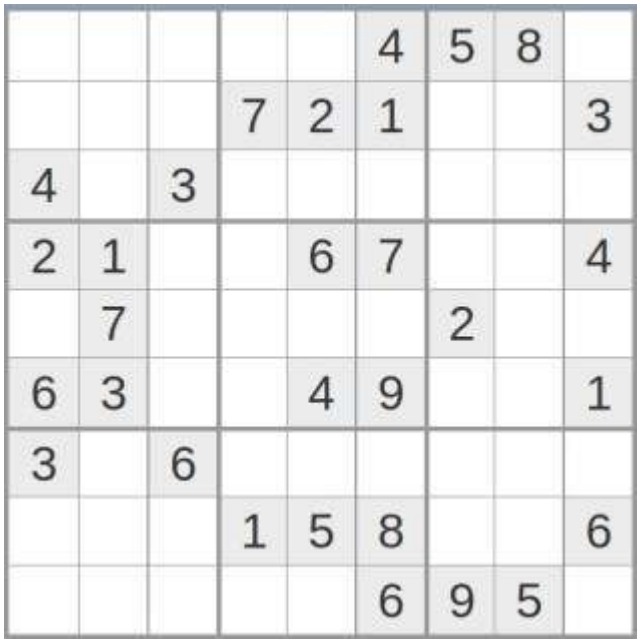
During the first defense, you will be asked to demonstrate the expected features, so you should prepare your code in order to demonstrate the different parts of your project easily (**stage visualization**, independent runs, etc.).

There are some common pitfalls in C, which could make you waste a lot of time. You will need to work carefully with a substantial amount of technical details. Develop good work habits in order to avoid wasting time. For memory issues, you could find it useful to use tools like *valgrind* or the *-fsanitize=address* of *gcc*.

VIII. Appendices

1. File Format Used for the `solver` Program

Consider the grid below:



					4	5	8	
			7	2	1			3
4		3						
2	1			6	7			4
	7					2		
6	3			4	9			1
3		6						
			1	5	8			6
					6	9	5	

The format of the corresponding file will be as follows:

```
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
```

It is a text file with the following characteristics:

- It has 11 lines:
  - 9 lines of 11 characters
  - 2 empty lines
- Empty grid cells are replaced by dots
- Horizontally, there is a space between each group of 9 cells
- Vertically, there is an empty line between each group of 9 cells

For instance:

```
$ ls
grid_00 solver
$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
$ ./solver grid_00
$ ls
grid_00 grid_00.result solver
$ cat grid_00.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
```

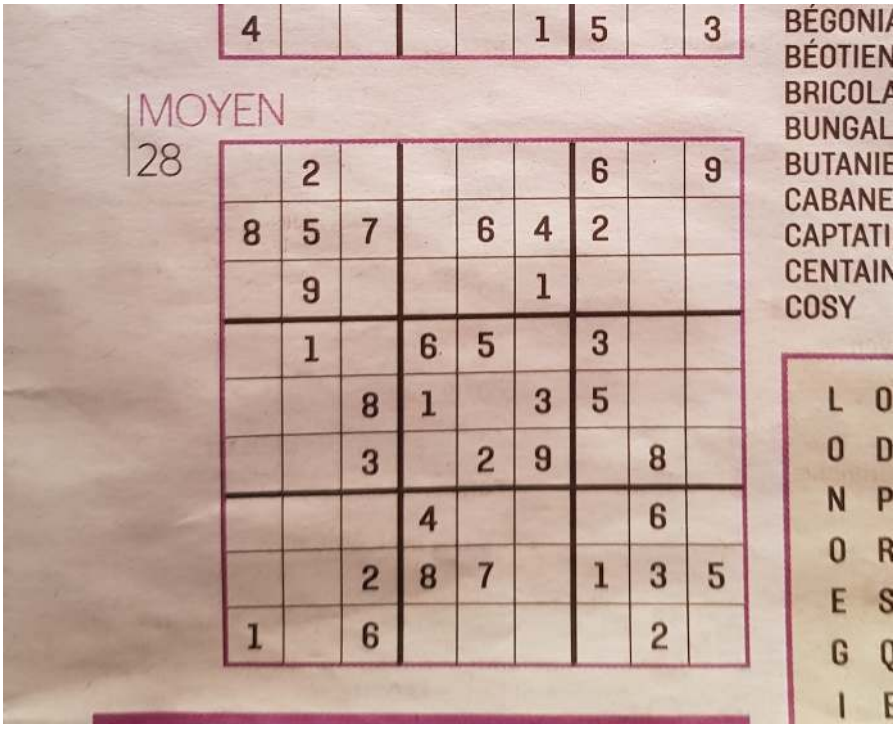
2. Examples of Pictures

Below are some pictures that can be used to test your code. The resolution of these images is much greater than what is shown below. Some PDF readers (like Evince) allow you to save these pictures in different formats.

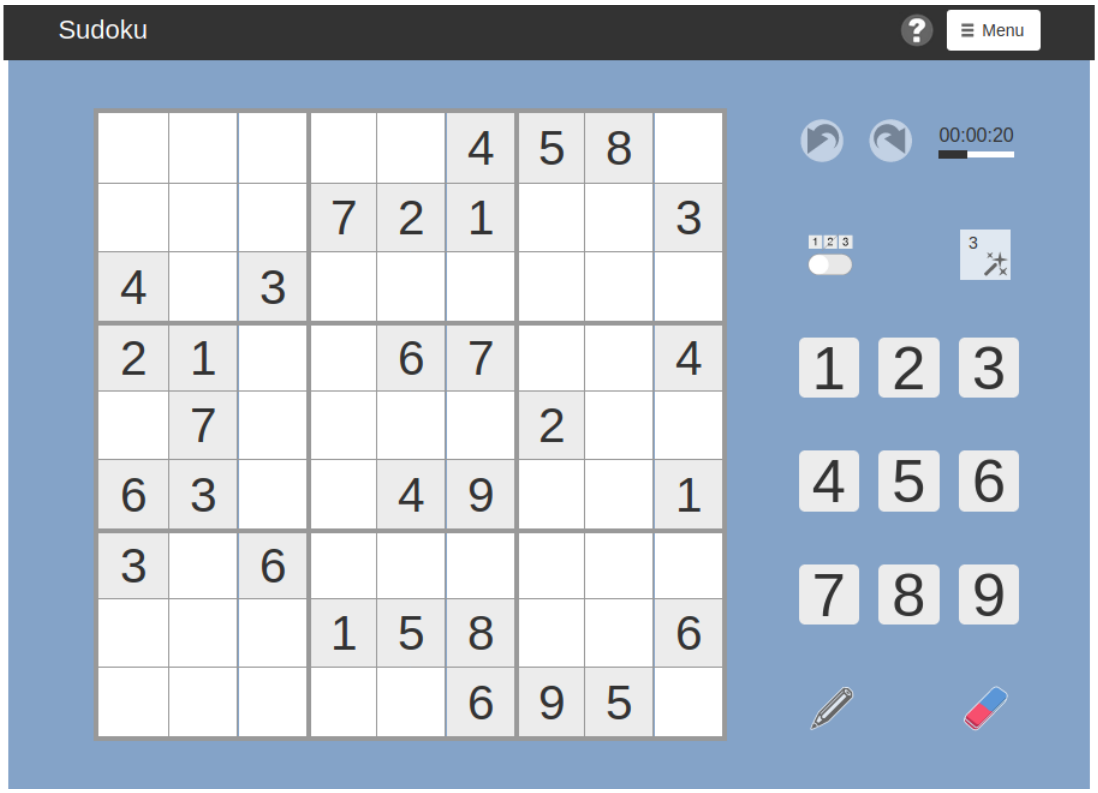
2.1. Picture 1

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

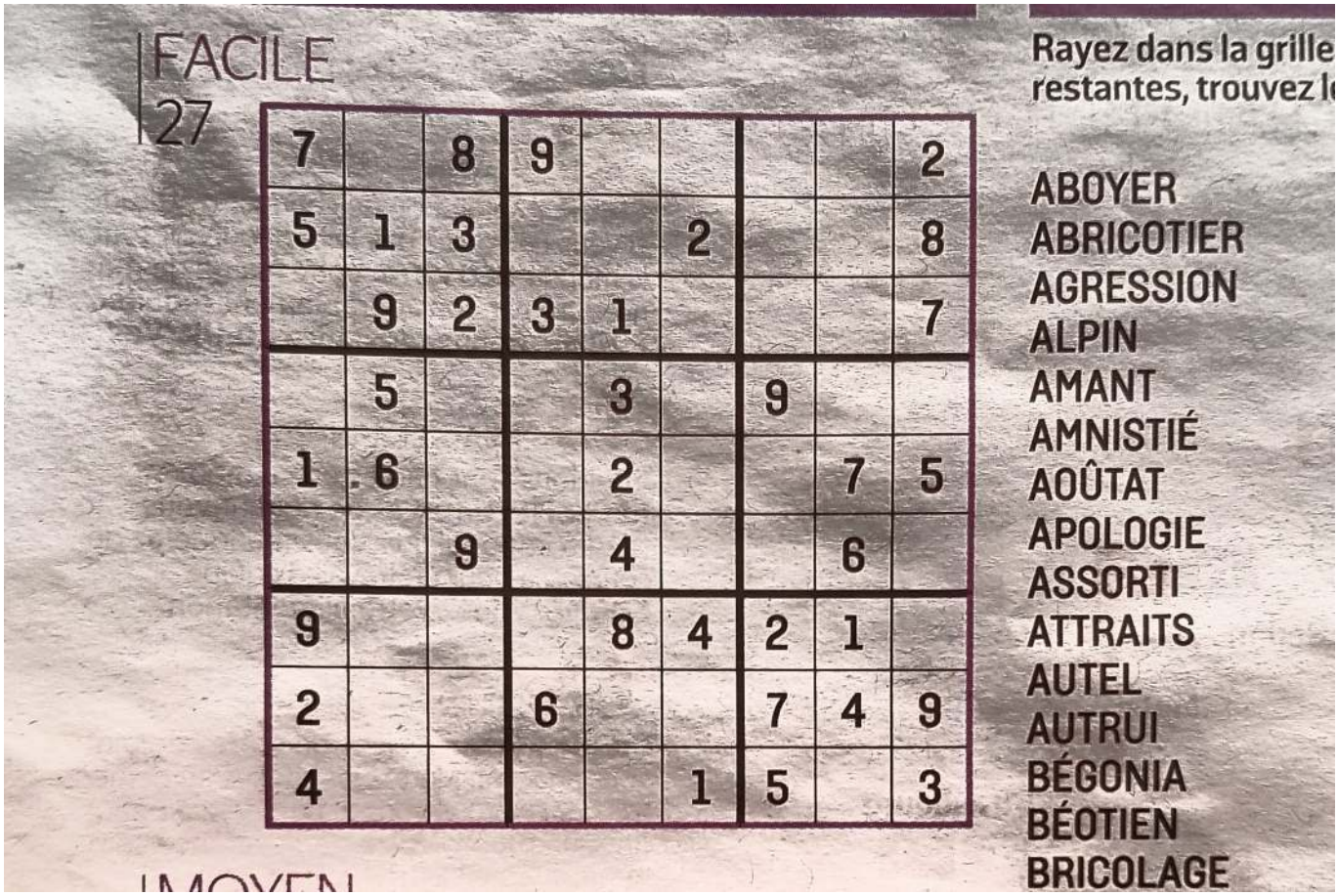
2.2. Picture 2



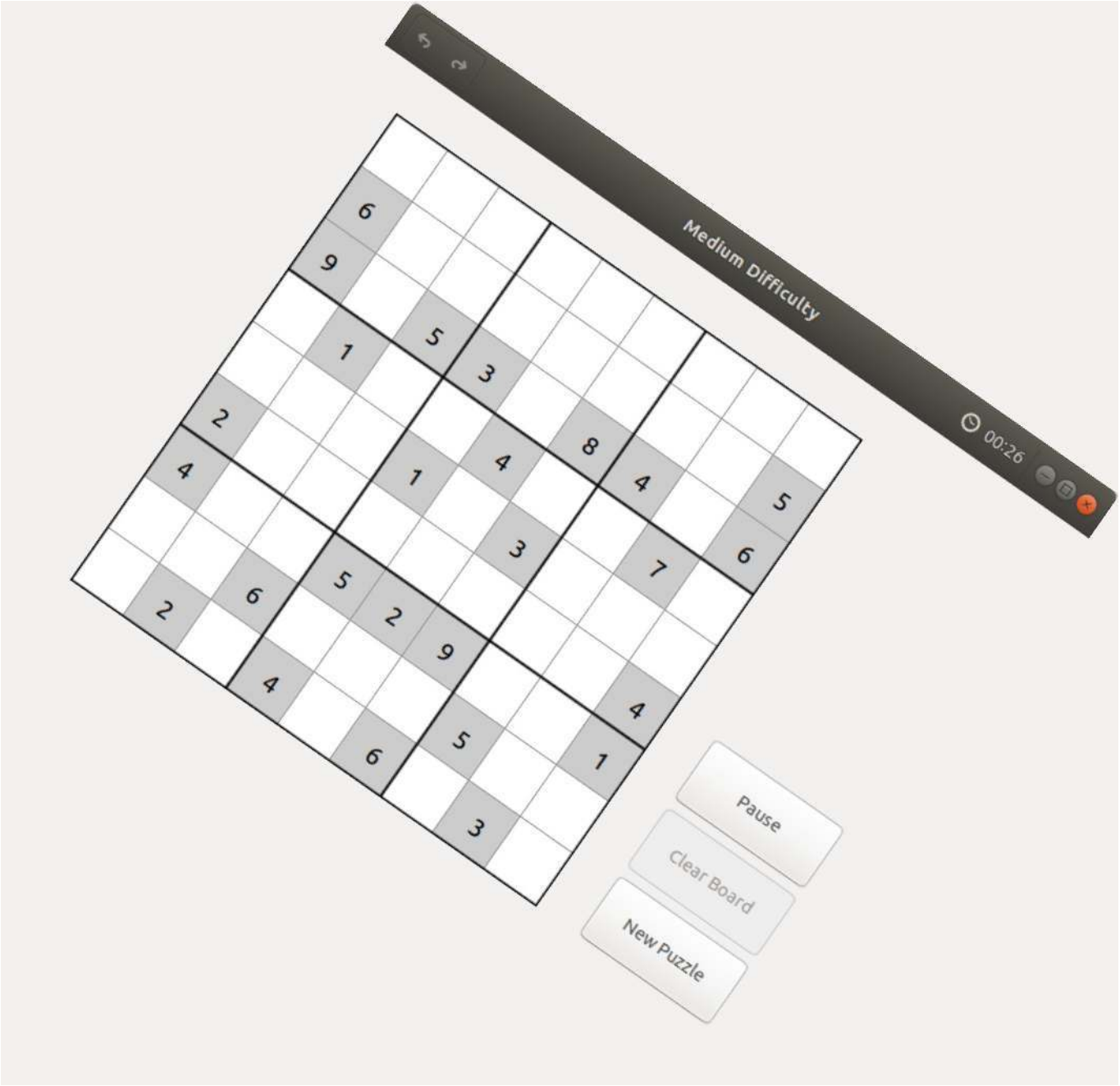
2.3. Picture 3



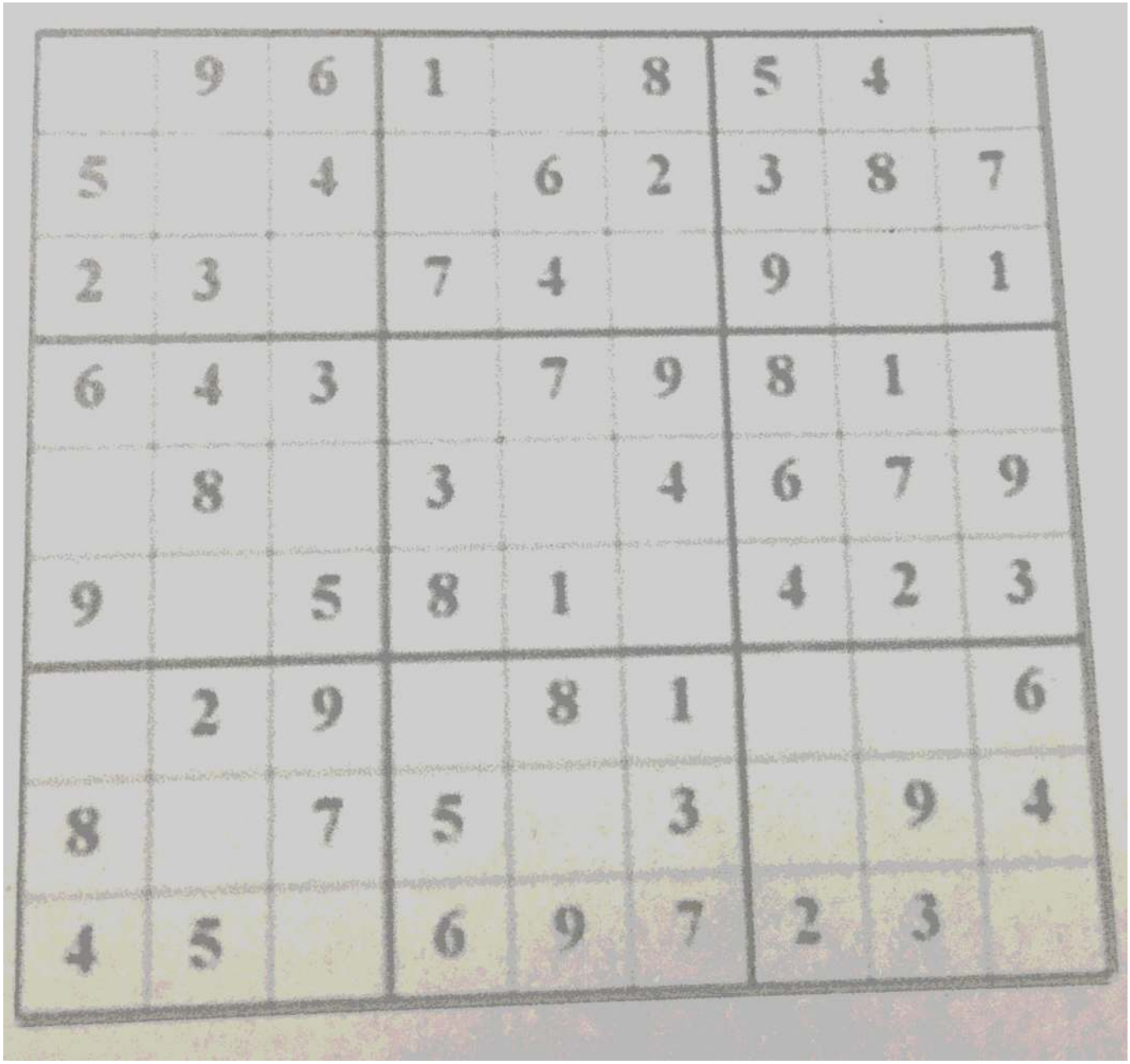
2.4. Picture 4



2.5. Picture 5



2.6. Picture 6



	9	6	1		8	5	4	
5		4		6	2	3	8	7
2	3		7	4		9		1
6	4	3		7	9	8	1	
	8		3		4	6	7	9
9		5	8	1		4	2	3
	2	9		8	1			6
8		7	5		3		9	4
4	5		6	9	7	2	3	

2.7. Download Links

If necessary, these images can be downloaded in JPEG format using the links below:

- http://www.debug-pro.com/epita/prog/s3/project/image_01.jpeg
- http://www.debug-pro.com/epita/prog/s3/project/image_02.jpeg
- http://www.debug-pro.com/epita/prog/s3/project/image_03.jpeg
- http://www.debug-pro.com/epita/prog/s3/project/image_04.jpeg
- http://www.debug-pro.com/epita/prog/s3/project/image_05.jpeg
- http://www.debug-pro.com/epita/prog/s3/project/image_06.jpeg