# SUDOKU

Ahmed Maher • Ahmed Ezzat • Khalid Rizk • Walid Shaaban • Mariam Yehia

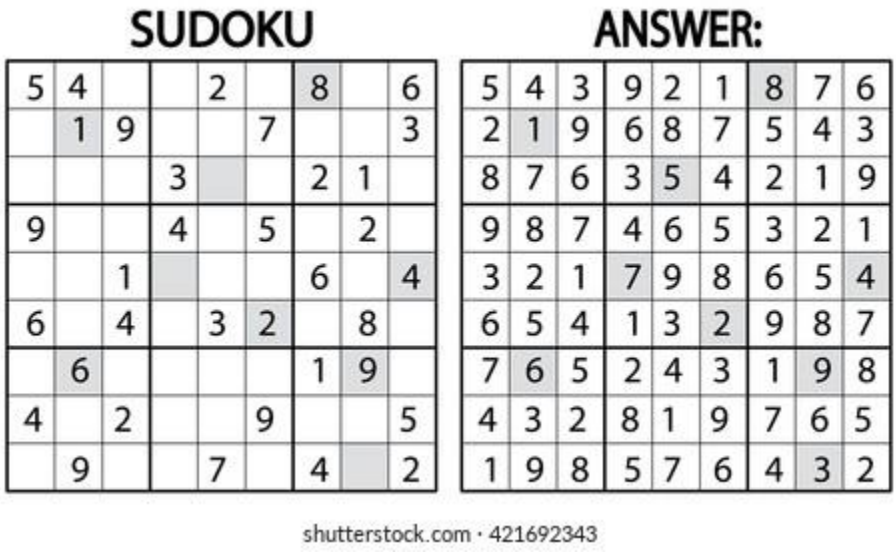
Project Link:
https://bit.ly/3qqgO5m

# Contents

# What is Sudoku game

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a *9×9* grid with digits so that each column, each row, and each of the nine *3×3* sub-grids that compose the grid (also called "boxes", "blocks", or "regions") contain all the digits from *1* to *9*. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

For example, the (left) figure demonstrates a typical Sudoku puzzle, and its solution (right).



# The idea of our project

We use this Algorithm to implement a sudoku solver that use this technique to try to fill each cell in the game with each possible number.

Sudoku solver is start from start cell and continue until reach to illegal state where that cell cannot be filled with any number the recursive call Backtrack to the nearest save state and try another path until reach the goal state .

# Main Functionalities

The main functionalities are to fill a *9×9* grid with digits so that each column, each row, and each of the nine *3×3* sub-grids that compose the grid contain all the digits from *1* to *9*.

# Similar application in the market

Android : Sudoku - The Clean One

Windows : Sudoku Free

iOS : Sudoku.com - Sudoku Puzzle

Linux : Sudoku

Website : Sudoku

# Publications (Papers) relevant to the idea

# A Pencil-and-Paper algorithm for solving Sudoku problem

Sudoku is a passion of many people all over the world. The interesting secret behind this it's a trivial puzzle which means an algorithm exists for solving it which is tree-based searching algorithm based on backtracking in it till a solution is found.

### *Definition of the Sudoku Board:*

Sudoku is played on a *9*x*9* board, there are *81* cells on the board which are broken into nine *3*x*3* sub-boards that don't overlap. We can refer to a specific cell on the board by giving the row number followed by the column number like *C (1,2)* where *C* donates the cell.

### *Preemptive sets and the Occupancy Theorem:*

The most important tool for solving sudoku puzzle is based on the definition of a sudoku puzzle

- *Definition #1 "Sudoku solution":*

  The solution requires that every column, row and box contain all numbers from *1* to *9* and every cell is occupied by only one number.

- *Definition #2 "Preemptive sets":*

  A preemptive set is composed of the set of numbers from *1* to *9* and it's of size of *2* to *9*, whose numbers are potentially occupants of *m* cells exclusively.

  A preemptive set is donated by $\{[n_1,n_2,\ldots,n_m],[c(i_1,j_1), c(i_2,j_2),\ldots c(i_m,j_m)]\}$ where $[n_1,n_2,\ldots,n_m]$, $1<n_i<=9$ for $i=1,2,..,m$, denotes numbers in the preemptive set and $[c(i_1,j_1), c(i_2,j_2),\ldots c(i_m,j_m)]$ denotes the set of *m* cells un which the set $[n_1,n_2,\ldots,n_m]$, and subsets of it, exclusively occur.

- *Definition #3 "Range of preemptive sets":*

  The range of a preemptive set is a row, column or a box in which all of the cells of the preemptive set are located. When *m=2* or *3* the range can be of the sets *[row, box]* or *[column, box]*.

  A description of a preemptive set it is a set of *m* distinct numbers from the set *[1,2, ... ,9]* and a set of *m* cells exclusively occupied by the *m* numbers or subset of them, with the property that the distribution of the m numbers across the *m* cells is not known at the time the preemptive set is discovered.

  The distribution of the *m* numbers into the *m* cells will be revealed as the solution of puzzle progress.

- *Theorem #1 "Occupancy theorem":*

  Let *x* be a preemptive set in a sudoku puzzle markup, then every number in *x* that appears in the markup of cells not in *x* over the range of *x* can't be a part of the solution.
- *Theorem #2 "Preemptive sets":*

  There is always a preemptive set that can be invoked to unhide a hidden set, which then changes the hidden set into a preemptive set except in the case of singleton.
- *Definition4 # "Sudoku violation":*

  a violation in sudoku occurs when the same number occurs two or more times in the same row, column or box.

## *Summary: Segmentation of the Algorithm:*

Steps for solving sudoku puzzles are:

1. Find all forced numbers in the puzzle.
2. Markup the puzzle.
3. Search iteratively for preemptive sets in all rows, columns and boxes taking appropriate cross out as each new preemptive set is discovers-until
4. either
   a. A solution is found or
   b. A random choice must be made for continuation.
5. If 4(a), then end. If 4(b) then go to step 3.

## *References*

[1] Michael Mepham (2005), Solving Sudoku, Crosswords Ltd., Frome, England.

# Differential Evolution Algorithm (DE) And Backtracking Algorithms

DE is very simple and effective heuristic algorithm, and it has been used to deal with a many of practical engineering optimization problems.

DE is a population based stochastic optimization algorithms (GA) and it consists of three core evolutionary operation crossover, mutation and selection, and the optimization performance of DE is affected by the mechanism used in the previous three cores and another factors like scaling factor F, crossover rate CR and population size NP.

The population of DE tends to be stuck in local optimum and unable to converge to the global optimum. and to solve this issue we use tracking and backtracking mechanism the propose of tracking mechanism to promote population convergence when one individual of the population fall in local optimum it should abandon its evolution path and track the evolution path of the individuals that are better than itself to promote the population to converge nearing of the global optimum , and backtracking mechanism help the population to escape from local optimum and trace back the stuck individual to a previous position by this we re-enhance the population diversity .

---

### *References*

[1] IEEE, 08 August 2018, 'Differential Evolution Algorithm with Tracking Mechanism and Backtracking Mechanism'

---

# Are Evolutionary Algorithms Required to Solve Sudoku Problems?

Evolutionary algorithms attempt to iteratively optimize the set of candidate solutions. Each solution is mutated randomly. Random mutations are applied to each solution, and a fitness function is used to assess whether an improvement has occurred. Evolutionary rhythms may then attempt to replicate traits of more successful candidates to others. This way we can become more like better solutions and the cycle continues. Sudoku has been used to demonstrate the ability of evolutionary algorithms. On various occasions, particle swarm, genetic algorithms and mixed meta-heuristics have been shown to be able to solve Sudoku problems.

***References***

[1] Sean McGerty and Frank Moisiadis, Feb 2014, 'Are Evolutionary Algorithms Required to Solve Sudoku Problems?', *Fourth International conference on Computer Science & Information Technology*.

# Solving the Sudoku with the Differential Evolution

When we apply Differential evolution algorithm on problem like Sudoku we need to classify the current Sudoku problem by its difficulty. As the difficulty matters in this problem, applying differential evolution algorithm we will get the solution of an easy board in about 4 thousand iterations, medium board in about 8 thousand iterations and in hard board we will get the answer in about 15 thousand iterations.

Unfortunately, this approach gives worse effect than the backtrack algorithm. For example, if we have a board that DE algorithm can solve it in 17 thousand iterations, we will notice that from 11th thousand iteration there will be no improvements. That is why it is better to use the Differential Evolution algorithm to find the closest solutions to the optimal solution around fitness equals to 2.

***References***

[1] Urszula Boryczka, Przemysław Juszczuk - Published 2012 - 'Solving the Sudoku with the Differential Evolution', *Eszyty Naukowe Politechniki Białostockiej. Informatyka*.

# Backtracking for Solving Sudoku Puzzle

Backtracking is a brute force approach that tries to build a solution given specific constraints. If there is an option failed to satisfy the constraints, then it backtracks and tries another path.

Considering the sudoku puzzle, in each empty cell in the grid the algorithm chooses a number to fill this cell, if the number doesn't violate the constraints -given by the rules of sudoku- it assigns that number to the cell and moves on to the next empty one, if the number is not valid it tries the other options (the 8 other numbers) and if none of them are valid it leaves the cell empty and "backtrack" to the previous cell incrementing its value by one and repeating the whole process.

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

In the grid above, it's obvious that no valid numbers can be assigned to the yellow cell, so it leaves it empty and backtrack to the blue cell, try another option till it finds a valid number.

## *References*

[1] Charu Gupta, 2017, 'solving sudoku using backtracking algorithm', *JNV University*

# Details of the algorithms used and the result of each of them

# Solving with DE algorithm

DE algorithm tries to solve sudoku problem using too much random population and generate it to get better until reach the right solution or reach close of the right solution.

But it starts with very bad solves and have large time to reach the right solution so we use validation algorithms by try to set valid number to empty cell and try this many times after N times if it cannot validate any number, it fill the cell with random number and try with another cell this algorithm give us more efficient boards with better fitness and close to the right solution so this is our first optimization.

Then we apply DE model on our population and test many times to reach the optimum value to our hyper parameter (weight of difference optimum=3 and CR optimum =0.009).

With this test we see that our algorithm can converge most time to local minimum and cannot reach the global minimum, so we try to optimize our code by try another algorithm help us to reach global minimum, so we use resolve algorithm with small modification to DE by make it allow out range values.

to know the place of wrong cell with and evaluate our fitness function because the wrong cell with values in range have big weight because it makes another cell is wrong to like the following.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | 1 | 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 |

When we have wrong cell, it effects on another cell like the previous figure so if we use out range values to mark wrong cell it does not affect to another cell like this figure.

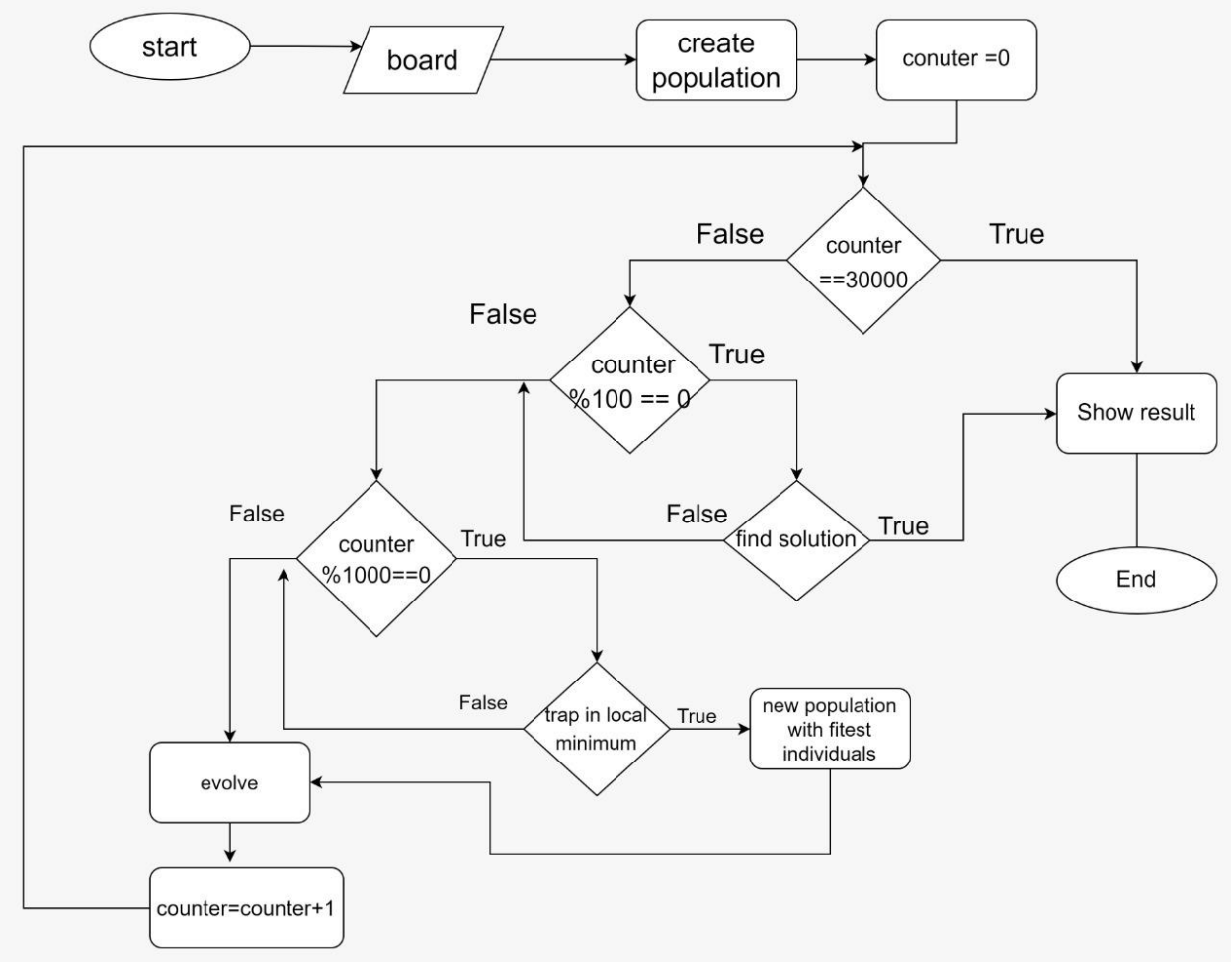
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | -1 | 7 | -5 | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | 12 | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

after we specify which cell is wrong, after each 100 generations we try to fill this cell with right answer N times and if we cannot fill it with right answer, we fill it with any value and try with another cell so this our second optimization.

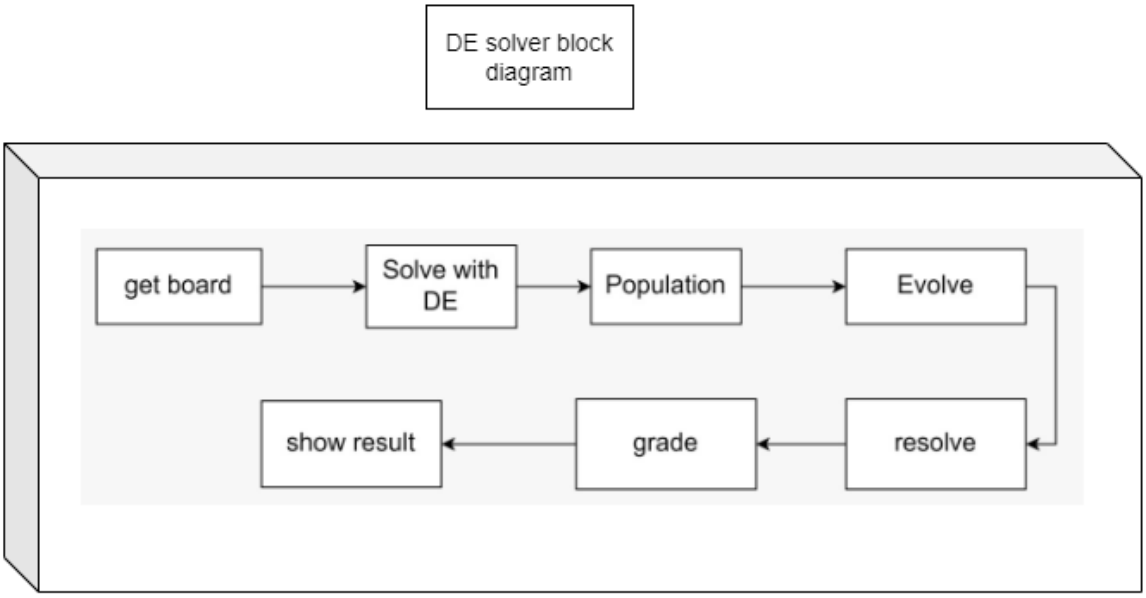
we can now conclude that it better than standard DE but still trap in local optimum to solve that we keep track the 10% fittest individuals of population and after each 1000 generations still the fittest of the past 1000 it will create another population and insert to another population the 10% fittest individuals of previous population in random places, and this is our third optimization.

After that population try to evolve again until reach the right solution or reach maximum number of generations.

**Flowchart:**



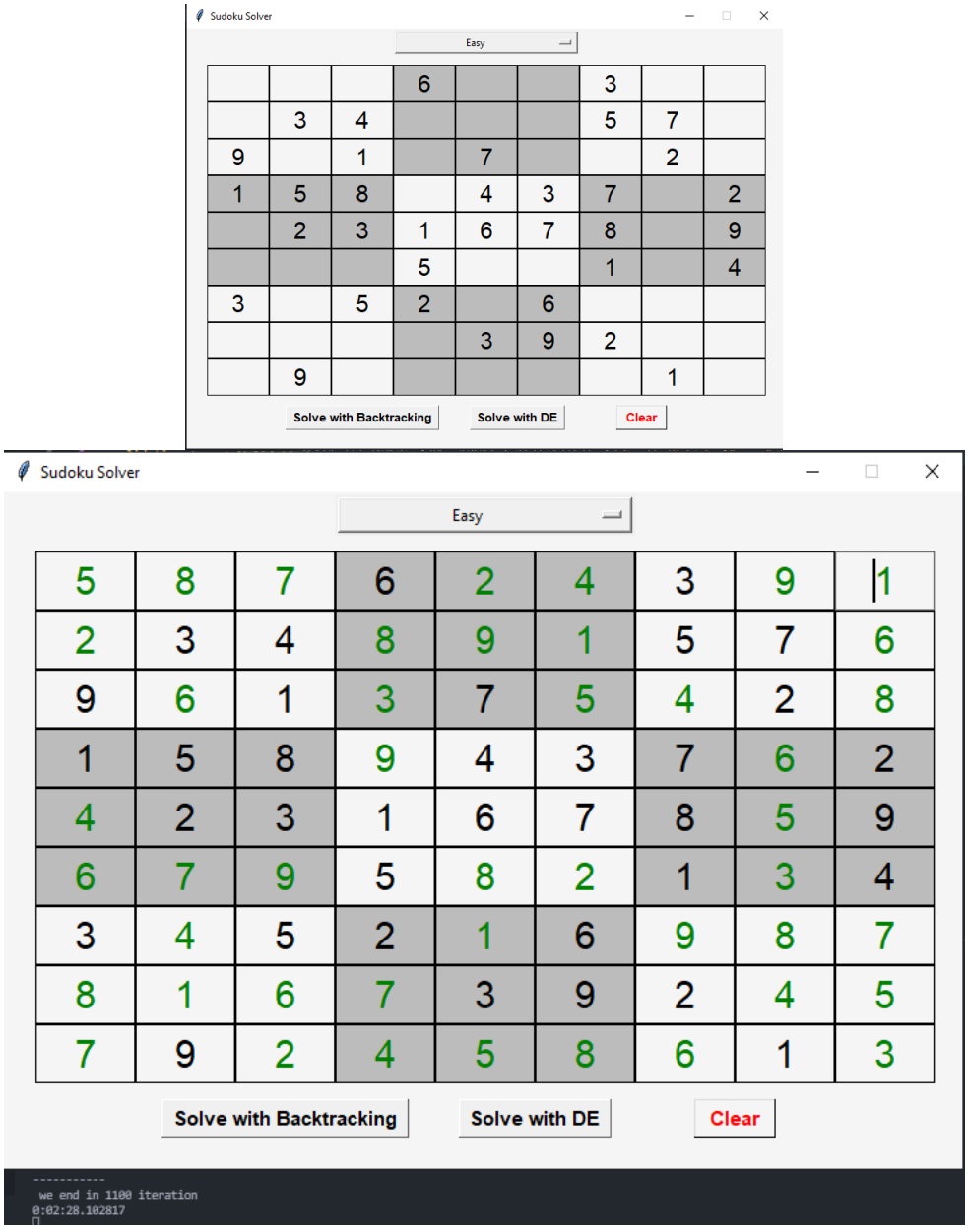
# Block diagram:



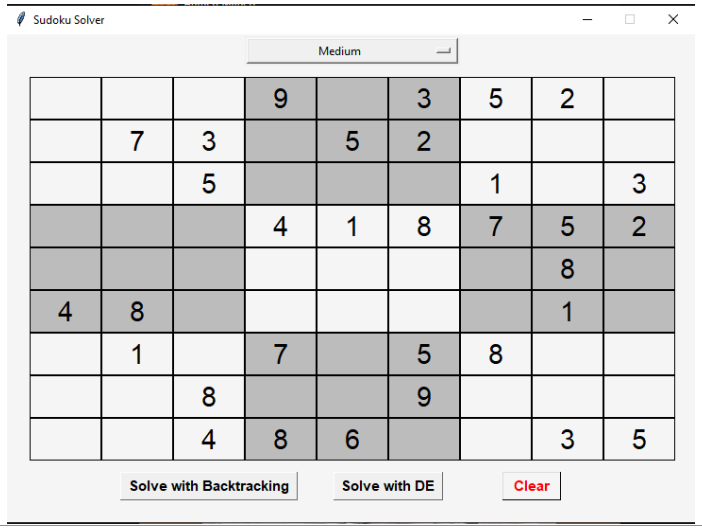

| | |
|---|---|
| Representation | Vectors |
| Recombination | Crossover |
| Crossover Rate (CR) | 0. 9% |
| Recombination probability | 100% |
| Mutation | Differential |
| Mutation probability | 100% |
| Parent selection | 3 Randoms vectors<br>+<br>Target vector |
| Survival selection | Replace worst |
| Population size | 100 |
| Number of Offspring | 1 |
| Initialization | Based on an algorithm |
| Termination condition | Check if reach the Solution after each 100 generation or evaluate 30,000 generation |

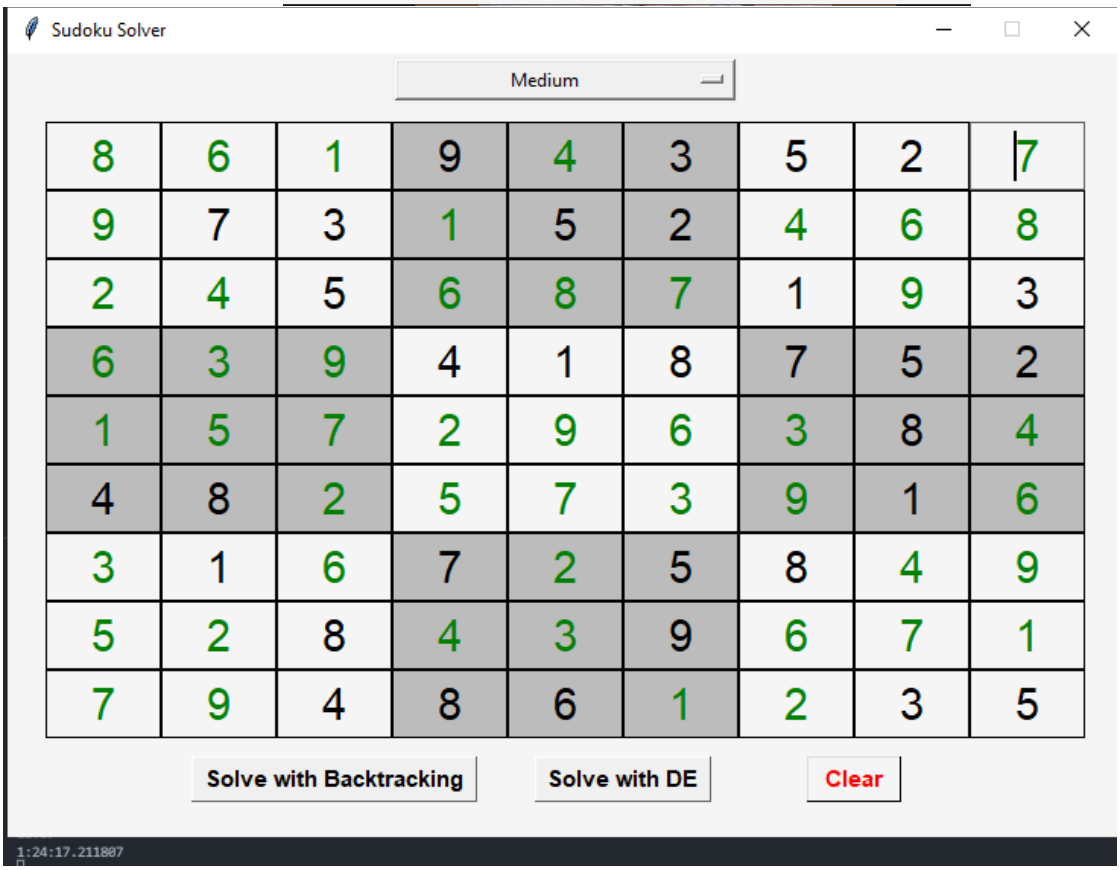
# Results:

-Easy Board: Solved in 2 minute and 28 seconds as shown in figure.

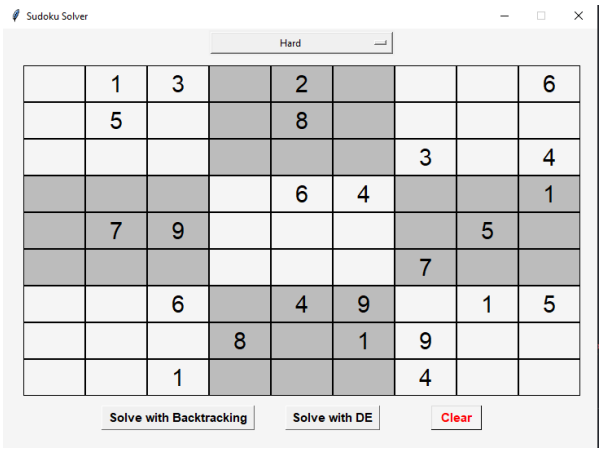


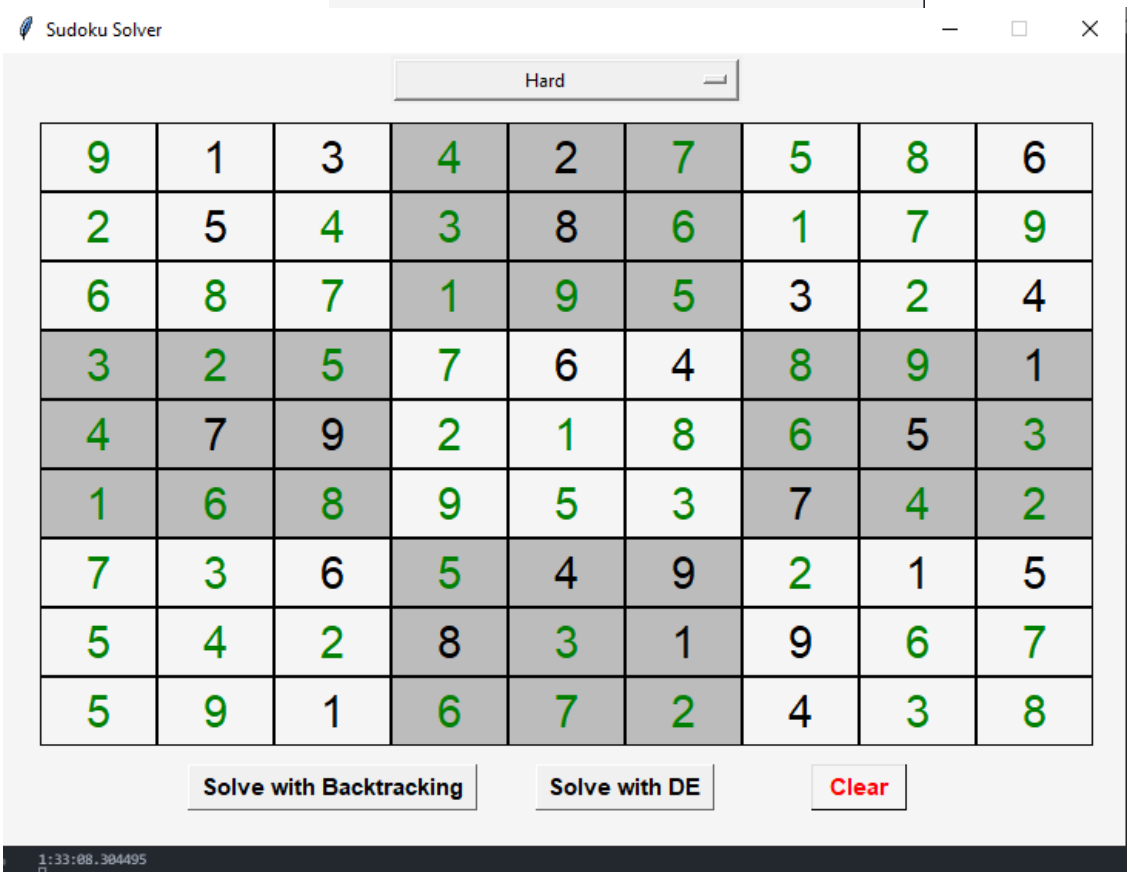
-Medium reach ***30000*** with fitness = ***2*** in ***1*** hour ***24*** minutes and ***17*** seconds.



Hard reach ***30000*** with fitness = ***3*** in ***1*** hour ***33*** minutes and ***8*** seconds.



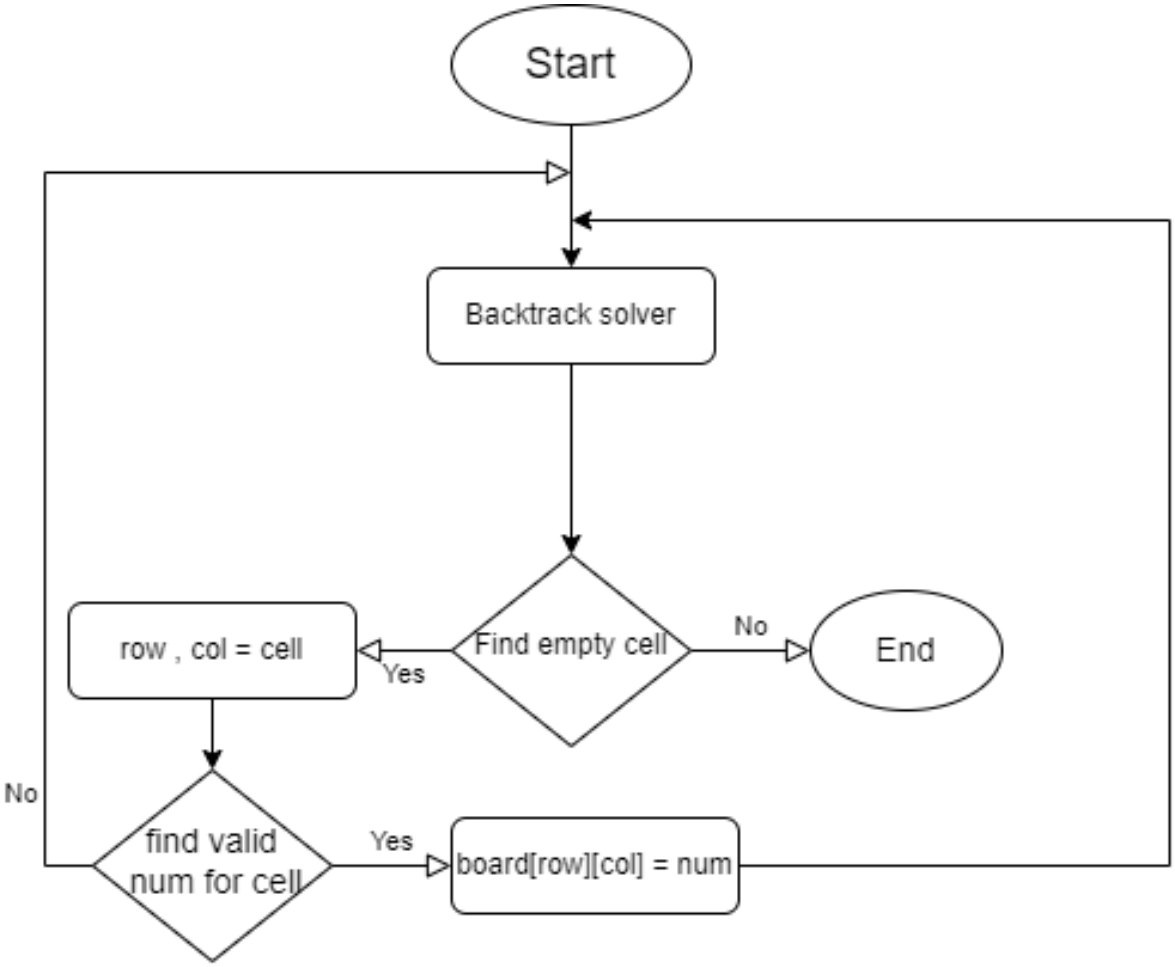

## Conclusion:

Medium, hard and evil boards will be solved in very huge time, and they require more than 30000 iterations to reach the optimal solution **(fitness = 0)** as we refer in this paper (Solving Sudoku with Differential evolution (Page 13)).

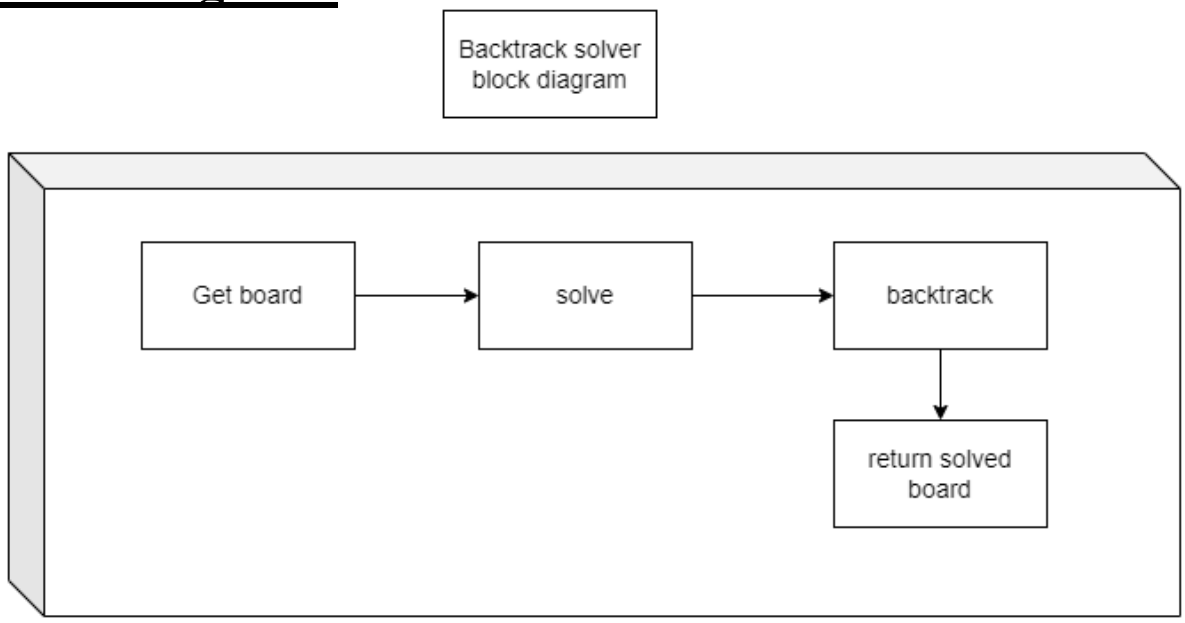
# Solving with Backtrack algorithm

The approach is to find each empty cell trying to assign the numbers from 1 to 9 to it, if there is a number that is valid - unique number in each column, row and box- then assign the number to the empty cell, change it color to green and recursively checks if it can lead to a solution to the grid, if not then the cell is unassigned and it tries the next valid number and checks again if it can lead to a solution, if there is no valid number that can lead to a solution to the grid then it returns false and that means there is no solution to the puzzle. The puzzle is solved when there are no empty cells left.

**Flowchart:**



# Block diagram:




# Results:

Easy Board: Solved in 258 iterations as shown in (figure 1.2).
Medium Board: Solved in 5663 iterations as shown in (figure 2.2).
Hard Board: Solved in 8316 iterations as shown in (figure 3.2).
Evil Board: Solved in 526242 iterations as shown in (figure 4.2).

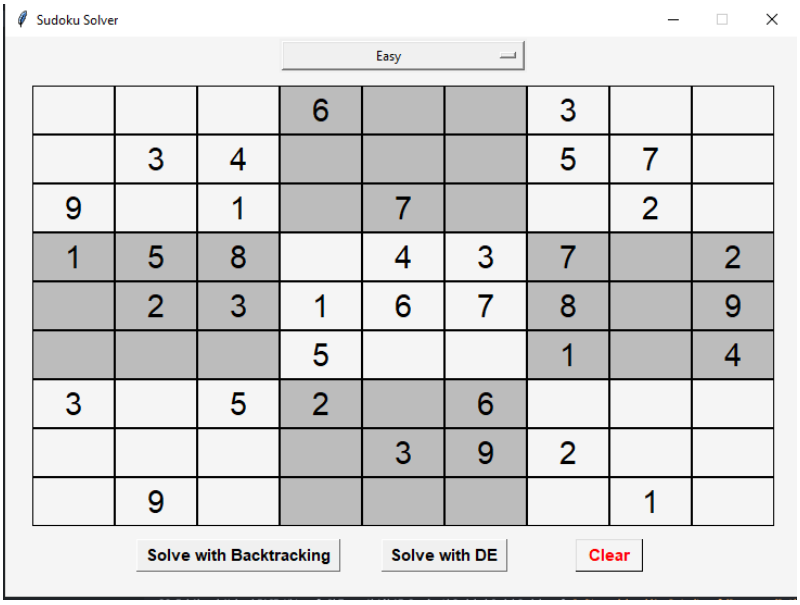



**Figure 1.1: easy puzzle.**

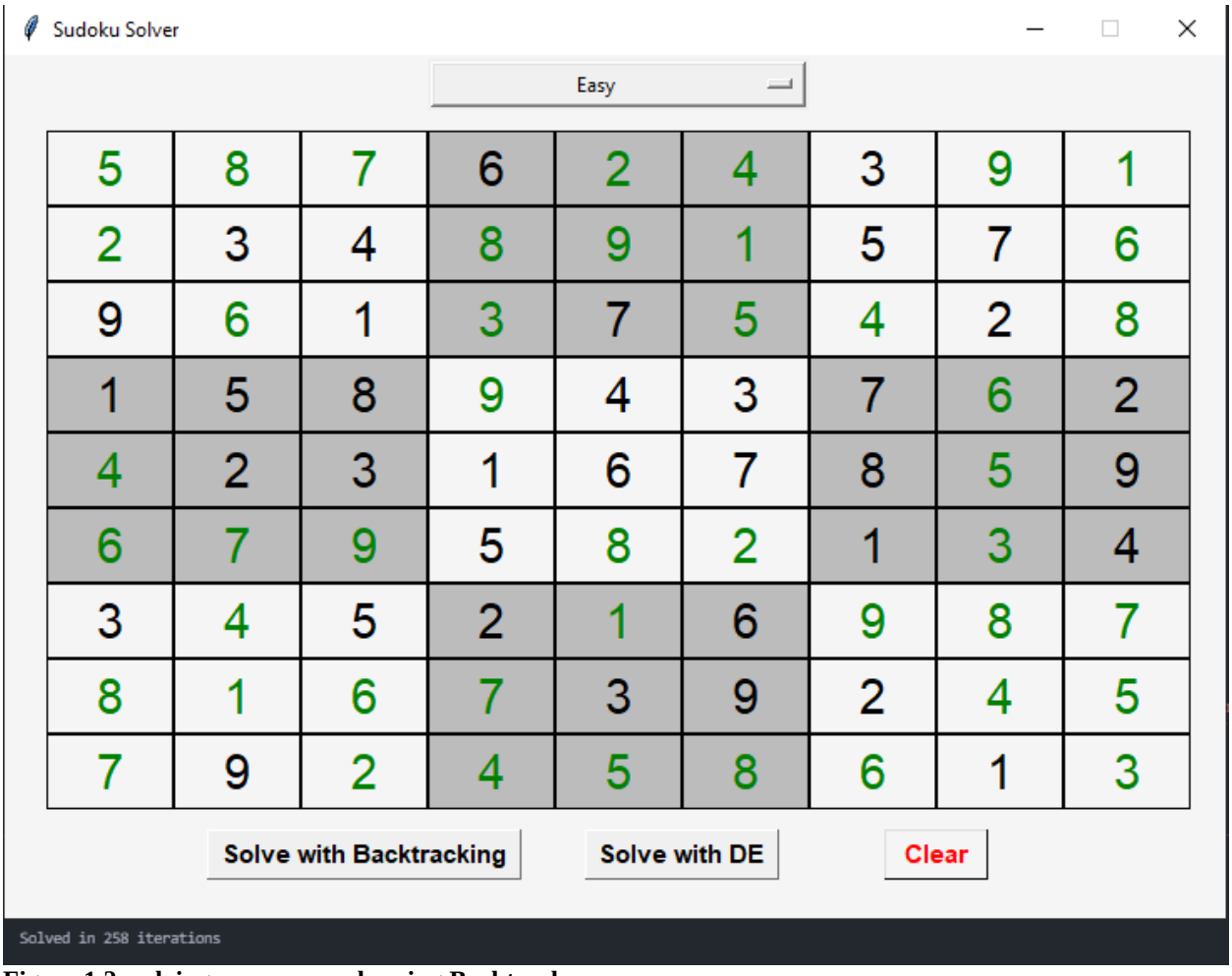



**Figure 1.2: solving an easy puzzle using Backtrack.**

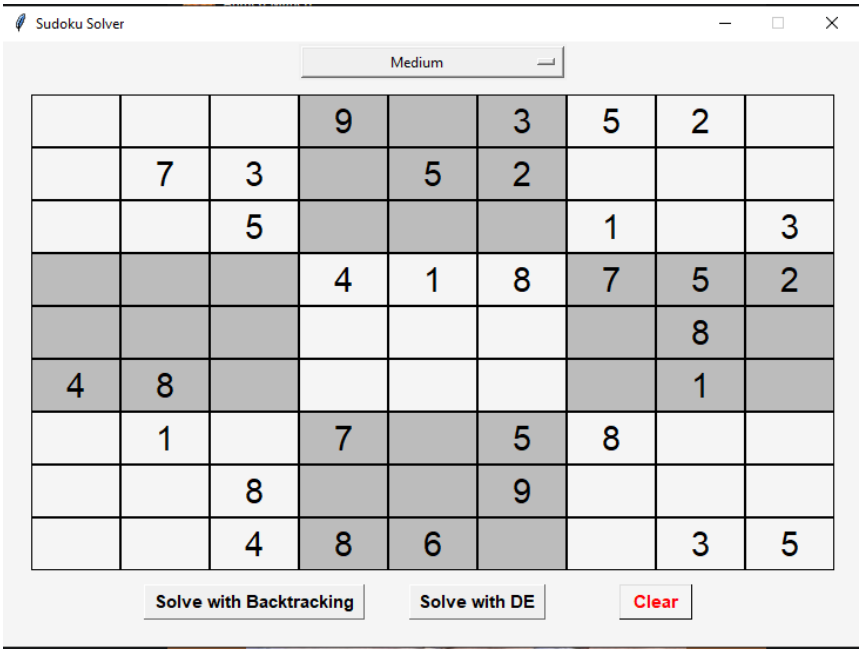



**Figure 2.1: medium puzzle.**

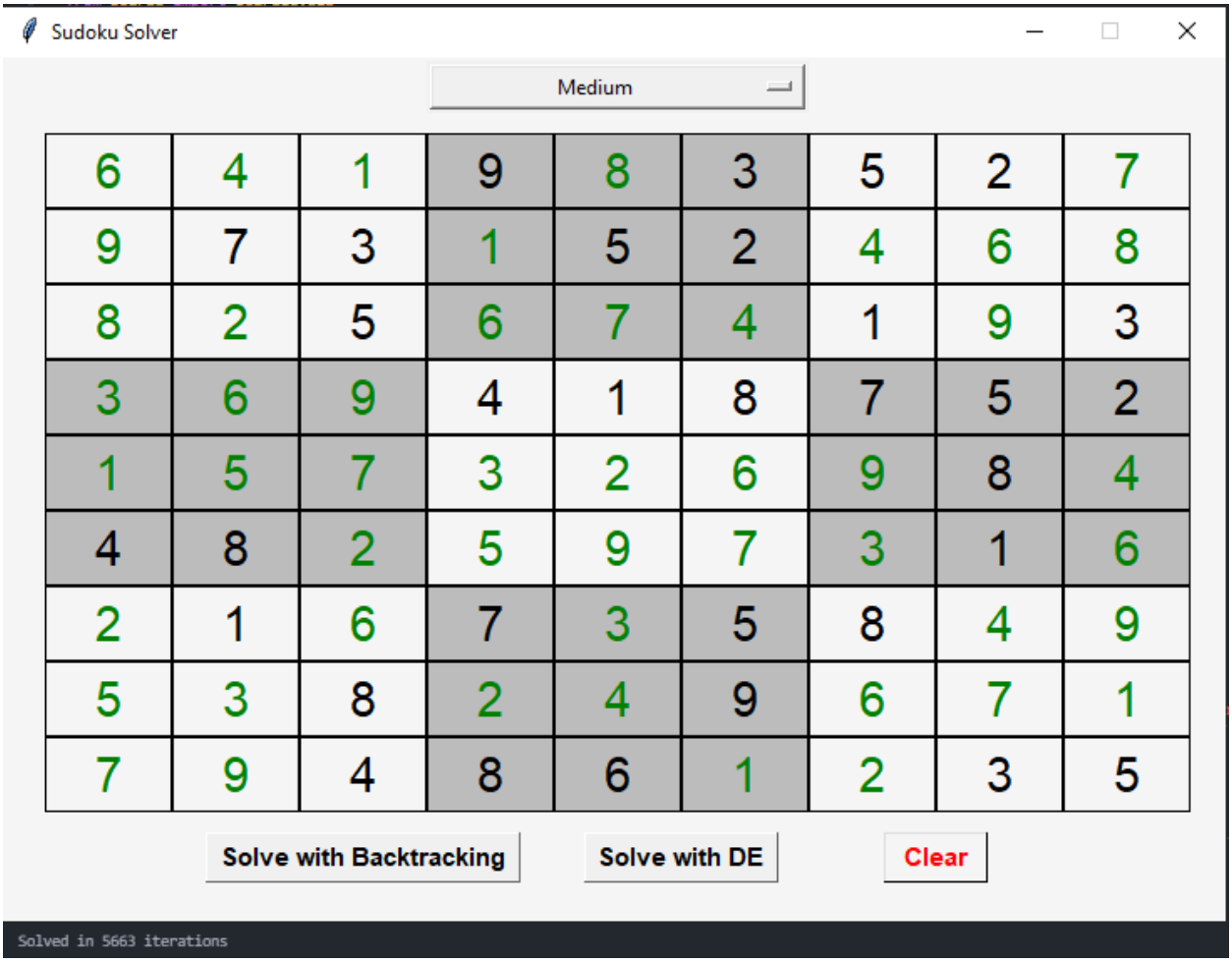



**Figure 2.2: solving a medium puzzle using Backtrack.**

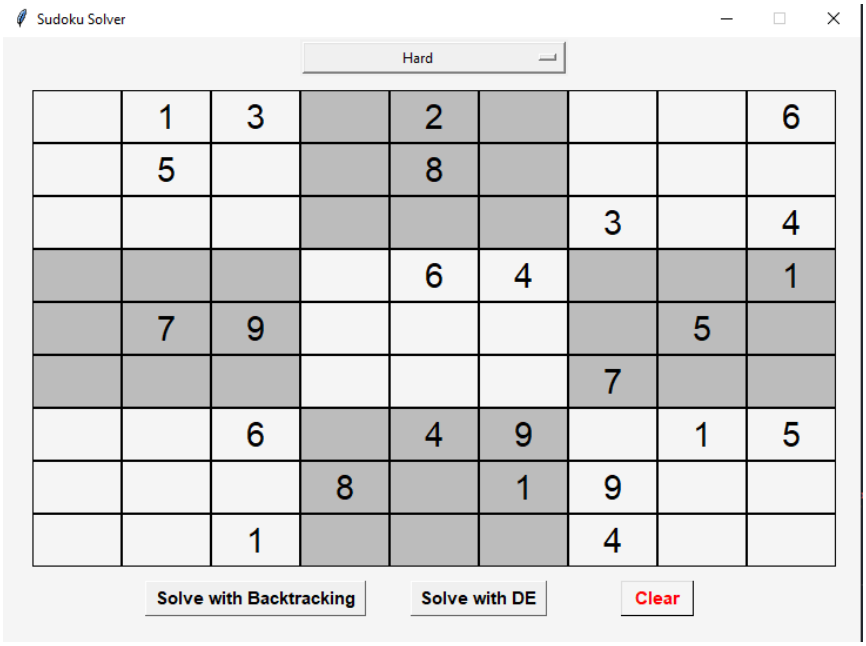



**Figure 3.1: Hard puzzle.**

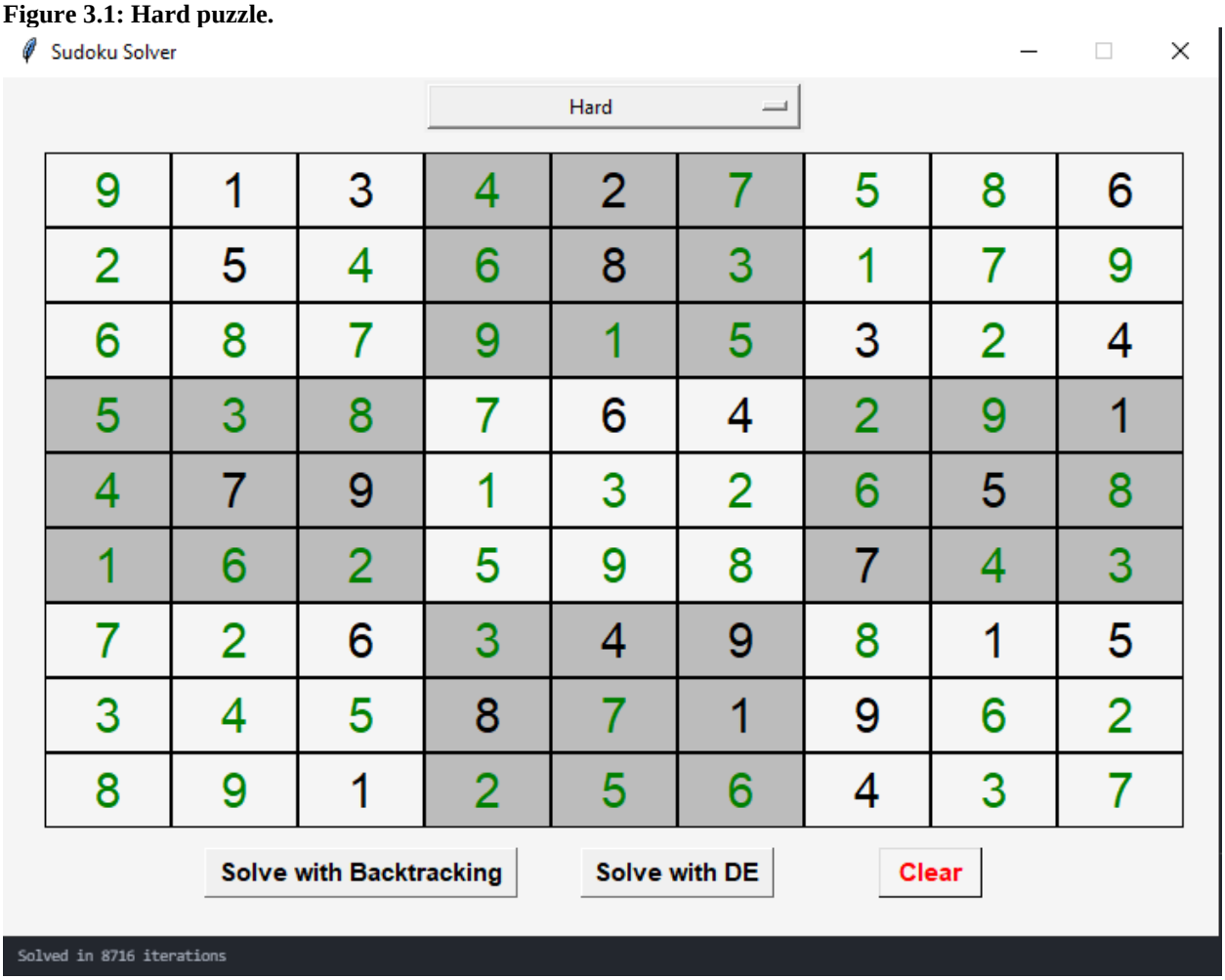



**Figure 3.2: solving a hard puzzle using Backtrack.**

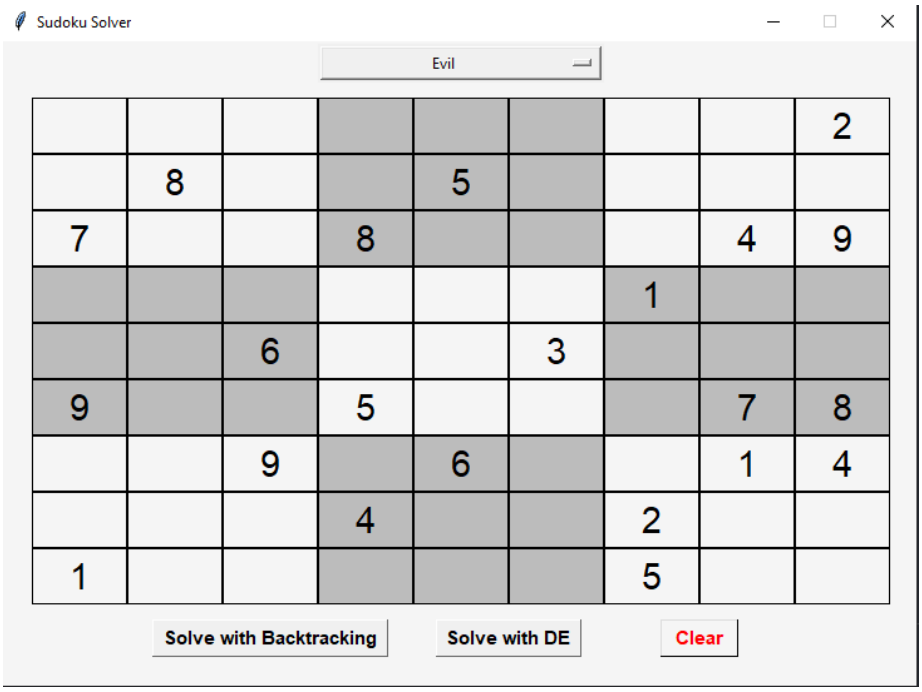



**Figure 4.1: Evil puzzle.**

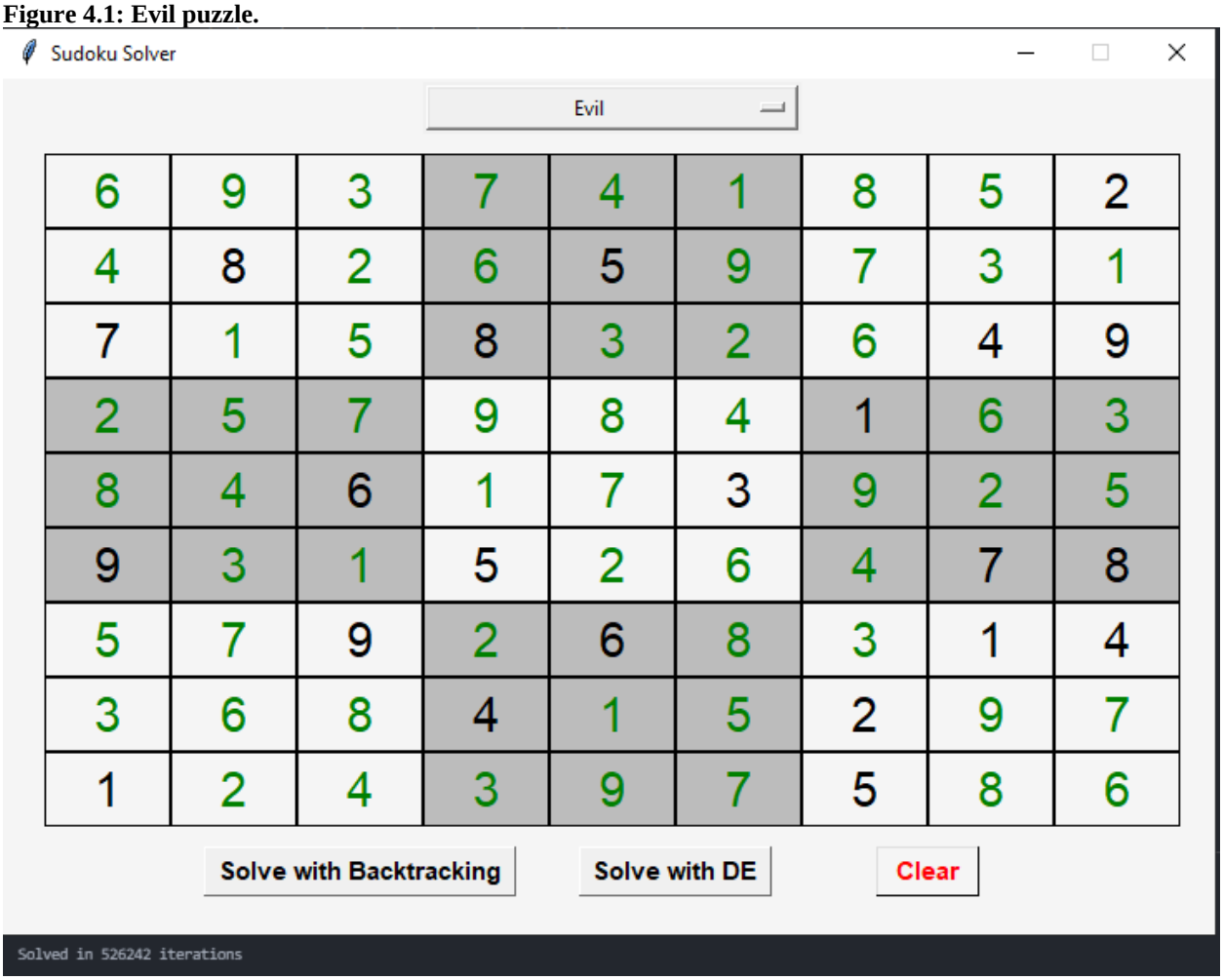
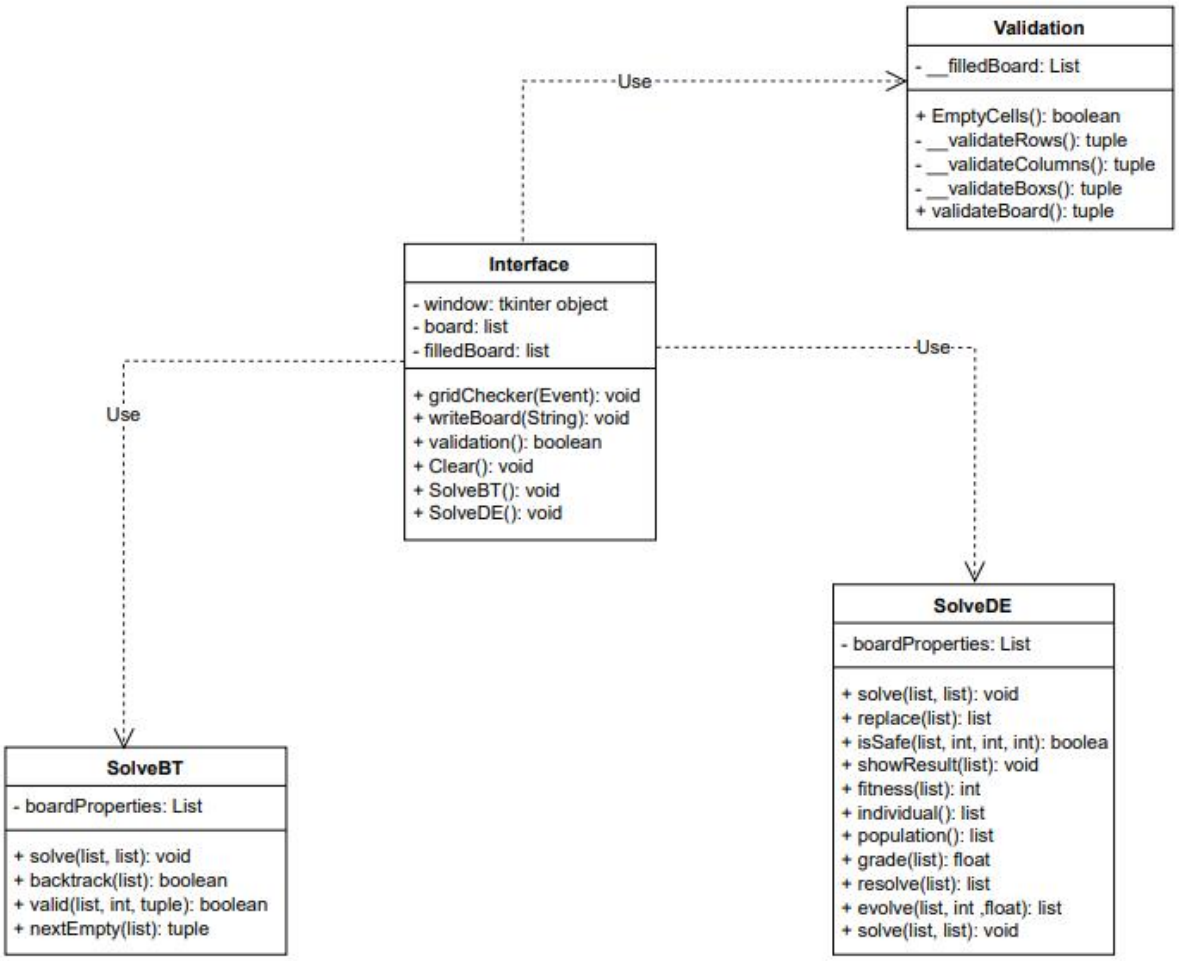



**Figure 4.2: solving an evil puzzle using Backtrack.**

# Class Diagram



# Development Platform

**Programming languages:** *Python*.

**Libraries:** *NumPy, random, validation and tkinter*.