CS 4613 **E. K. Wong**
Project 2: Sudoku Mine **Fall 2025**

Total number of points = 100.

Project Description*:* Implement the *Backtracking Algorithm* in Figure 5 (on last page) to solve the Sudoku Mine puzzle. The game board consists of 9×9 cells divided into 3×3 non-overlapping blocks (see Figure 1.) The rules of the game are:

- Place mines on the board so that each row, column, and block contain exactly three mines.
- Some of the cells have a number (ranging from 1 to 8) inside. The number indicates the number of mines that are present in a cell's 8-neighbors. The 8-neighbors of a cell are those cells that are horizontally, vertically or diagonally adjacent to the cell. A blank cell means there are no restrictions regarding the number of mines in the cell's 8-neighbors.
- Mines can only be placed on empty cells that do not have a number inside.

Figure 2 shows the solution for the puzzle in Figure 1.

1	2		2	2			2	1
		3		3	4	4		
1			4		3		4	
2	4			2				1
		3		3	3	3		2
2			3			2		
1		1	2			3	4	
3			2		2	2		
		3		1		1	2	2

Figure 1. Initial game board

1	2	●	2	2	●	●	2	1
●		3	●	3	4	4	●	
1		●	4	●	3	●	4	
2	4	●		2		●	●	1
●	●	3	●	3	3	3		2
2			3	●	●	2		●
1		1	2	●	●	3	4	●
3	●		2		2	2	●	●
●	●	3	●	1		1	2	2

Figure 2. Solution (In your output file, no need to copy the original numbers inside the cells from the input file.)

In the Backtracking Algorithm, use the *minimum remaining value* and *degree* heuristics to implement the *SELECT-UNASSIGNED-VARIABLE* function. For the *ORDER-DOMAIN-VALUES* function, simply order the domain values in the order {0,1} instead of using heuristics. For the INFERENCE function, implement the *Forward Checking* algorithm as discussed in class. (Reminder: when implementing the *minimum remaining values* and *degree heuristics*, two or more variables are neighbors if they share one or more common constraints.)

Input and output files: Your program will read in the initial game board values from an input text file and then produce an output text file that contains the solution. The format of the input file is as shown in Figure 3 below. The nine rows of *n's* are integers that contain the initial cell values of the game board, separated by blanks. The value for *n* ranges from 0 to 8, with 0 indicating a blank cell. If $n \neq 0$, it indicates the number of mines that are present in the cell's 8-neighbors. The output file contains 11 rows of integers as shown in Figure 4 below, where *d* is the level of the goal node as found by your algorithm (root node is at level 0) and *N* is the number of nodes generated in your search tree. Rows 3 to 11 contain your solution with *n* equals to 0 (no mine) or 1 (has mine.) The *n'*s are separated by blanks. (No need to copy the initial cell values from the input file to the output file.)

Testing your program: Three input test files will be provided on NYU Brightspace for you to test your program.

Recommended languages: Python, C++/C and Java. If you would like to use a different language, send me an email first.

Teammate: You can work on the project by yourself or form a team of two students to work on the project. You can discuss with your classmates on how to do the project but every team is expected to write their own code and submit their own project.

What to submit: Submit the following files on Brightspace by the due date. If you work with a partner, only one partner needs to submit but put both partners' names on the PDF report and the source code.

1. Your source code file. Put comments in your source code to make it easier for someone else to read your program. Points will be taken off if you do not have comments.
2. The output text files as generated by your program. Name your output files *Output1.txt, Output2.txt* and *Output3.txt*.
3. A PDF report that contains the following:

   a. Instructions on how to run your program. If your program requires compilation, instructions on how to compile your program should also be provided.
   b. A description of your formulation of Sudoku Mine as a constraint satisfaction problem. This includes the set of *variables* you have defined, the *domains* for the variables, the set of *constraints* you have set up, and any other additional information on your formulation.
   c. Also, copy and paste the output files and your source code onto the PDF report (to make it easier for us to grade your project.) This is in addition to the source code and output files that you have to submit separately, as described in (a) and (b) above.

```
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
```

Figure 3. Input file format: *n* is an integer between 0 and 8, with 0 indicating a blank cell. The digits are separated by blanks.

```
d
N
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
```

Figure 4. Output file format: *d, N* and *n* are integers, with *n* equals 0 (no mine) or 1 (has mine.) The digits are separated by blanks. (No need to copy the initial cell values from the input file to the output file.)

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

Figure 5. The Backtracking Algorithm for CSPs.