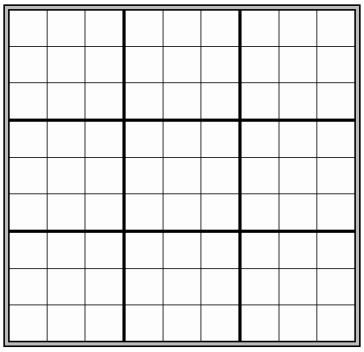
Sudoku Game

Youssef Elraggal	**7806**
Mohamed osama	**7861**
Abdelrahman tarek	**7472**

Constraint Satisfaction Problem

sudoku grid :



Variables : {(0, 0), (0, 1), (0, 2), (0, 3), ...} (81 variables) (9x9 grid)

Domains : {1, 2, 3, 4, 5, 6, 7, 8, 9} for each variable

Constraints :

1) Each square must contain a single number, from 1 to 9.

2) The same number can't appear in the same column twice.

3) The same number can't appear in the same row twice.

- constraints 1 ,2 & 3 are hard constraint , that must be satisfied in a correct solution

- unary constraint: constraint involving only one variable

- binary constraint: constraint involving two variables

Algorithms used

Backtracking Used for validating input puzzles and generating random solvable puzzles

Node consistency when all the values in a variable's domain satisfy the variable's unary constraints.

Arc consistency when all the values in a variable's domain satisfy the variable's binary constraints arc consistency To make X arc-consistent with respect to Y, remove elements from X's domain until every choice for X has a possible choice for Y (Enforced on all pairs of connected variables (cells) in the grid. Iteratively applied until no further changes can be made to the domains.)

Pseudo codes used:

Arc Consistency

```
function AC-3(csp):
    queue = all arcs in csp
    while queue non-empty:
        (X, Y) = DEQUEUE(queue)
        if REVISE(csp, X, Y):
            if size of X.domain == 0:
                return false
            for each Z in X.neighbors - {Y}:
                ENQUEUE(queue, (Z, X))
    return true
```

Arc Consistency

```
function REVISE(csp, X, Y):
    revised = false
    for x in X.domain:
        if no y in Y.domain satisfies constraint for (X, Y):
            delete x from X.domain
            revised = true
    return revised
```

```
function BACKTRACK(assignment, csp):
  if assignment complete: return assignment
  var = SELECT-UNASSIGNED-VAR(assignment, csp)
  for value in DOMAIN-VALUES(var, assignment, csp):
    if value consistent with assignment:
      add {var = value} to assignment
      inferences = INFERENCE(assignment, csp)
      if inferences ≠ failure: add inferences to assignment
      result = BACKTRACK(assignment, csp)
      if result ≠ failure: return result
    remove {var = value} and inferences from assignment
  return failure
```

Heuristic used for optimization (Extra work)

SELECT-UNASSIGNED-VAR( ). We use MRV (minimum remaining values) to build this function

MRV_heuristic: select the variable that has the smallest domain

DOMAIN-VALUES( ). We use LCV (least-constraining values) to build this function

LCV_heuristic: return variables in order by number of choices that are ruled out for neighboring variables • try least-constraining values first

Data Structures used

1. Board Representation:

- The Sudoku board is represented as a 9x9 grid, where each cell can hold a value from 1 to 9.

- It's typically represented as a 2D list (list of lists) in Python.

2. Domains:

- It's represented as a dictionary where keys are cell coordinates (tuples of row and column indices), and values are sets containing possible values for that cell.

3. Inferences:

- Inferences represent the deductions made during the solving process, which reduce the domain of certain cells.

- It's represented as a dictionary where keys are cell coordinates (tuples of row and column indices), and values are the inferred values for those cells.

4. Queue:

- The queue is used in the AC-3 (Arc Consistency 3) algorithm for maintaining a list of arcs (constraints) to be processed.

5. Directed Graph:

- The directed graph is used to represent arcs (constraints) between cells.

- It's a list of tuples, where each tuple contains two cell coordinates representing a directed arc.

6. Assignment:

- Assignment represents the current state of the Sudoku board with assigned values.

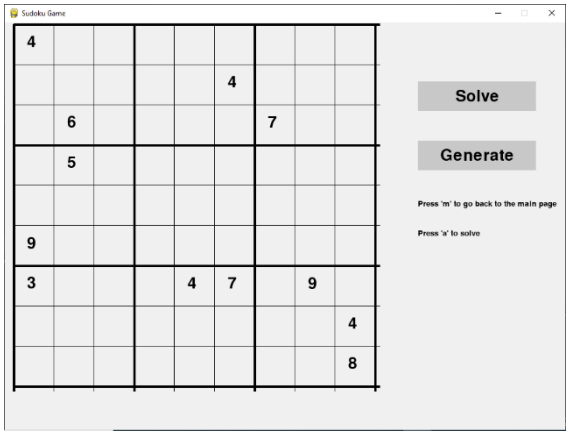
- It's represented as a dictionary where keys are cell coordinates (tuples of row and column indices), and values are the assigned values for those cells.

comparison between different initial boards (easy, intermediate, hard) and the time needed to solve puzzle.

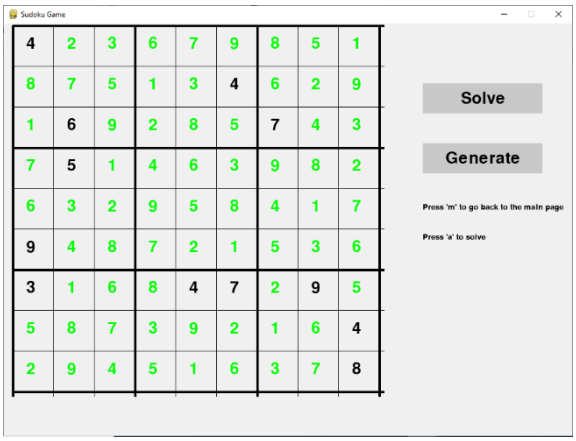
- Higher difficulty means that the board contains less initial values

```
for _ in range(np.random.randint(12,16)):  # Hard
# for _ in range(np.random.randint(20,25)):  # Mideum
# for _ in range(np.random.randint(30,45)):  # easy
```

Hard board :

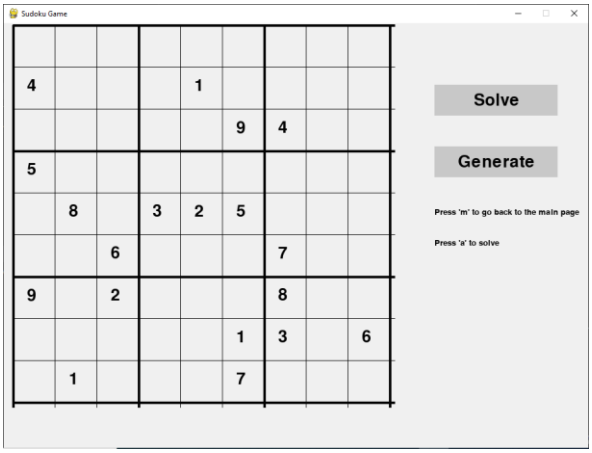


Solution :

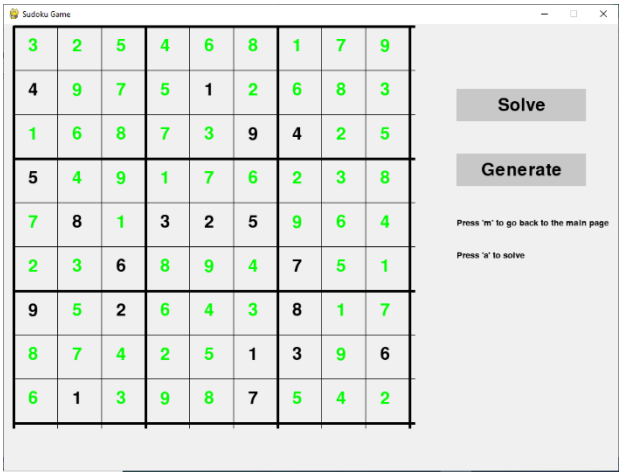


Time taken by backtracking: 0.3328273296356201

Intermediate board :

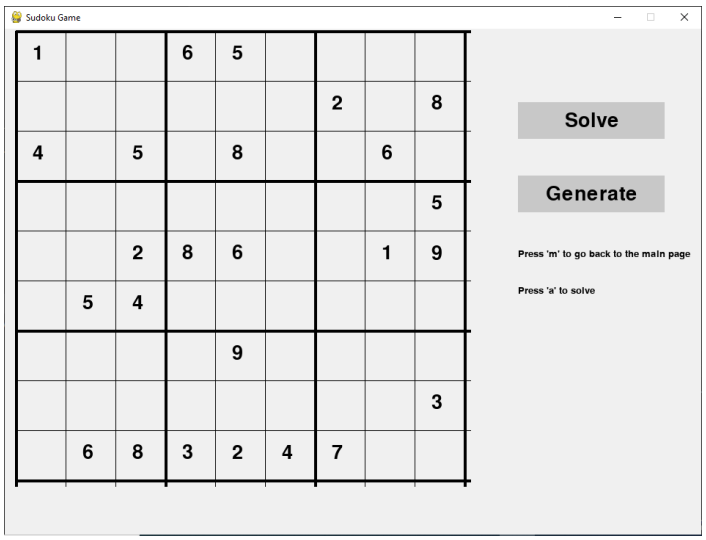


Solution :

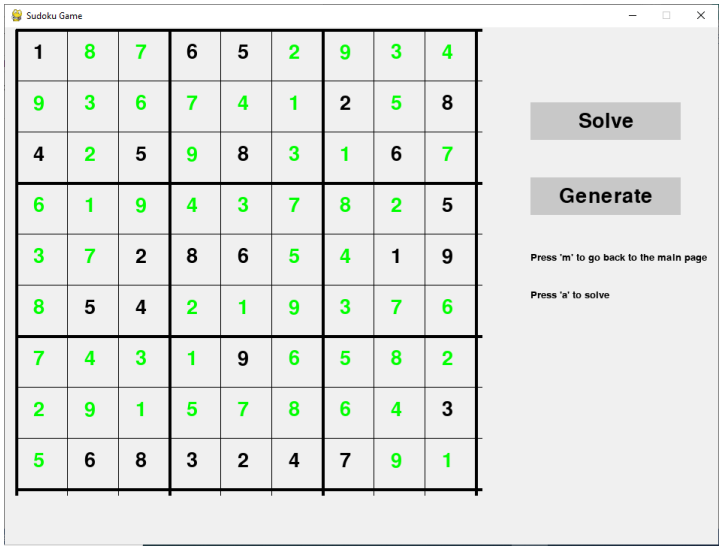


Time taken by backtracking: 0.19205021858215332

Easy board :



solution :



Time taken by backtracking: 0.0930013656616211

Assumptions Made

Backtrack assumes that the user's input in Mode 2 and Mode 3 follows Sudoku rules.

Modes description:

Mode 1: AI (generate & solve)

In this mode, the game's AI generates a Sudoku puzzle and solves it by clicking the solve button without any user input and the generate button creates a new sudoku puzzle. The AI algorithm uses backtracking to generate a solvable puzzle and then proceeds to solve it. The player's role is to observe the AI's puzzle generation and solving process.

Mode 2: User generate & AI solve

This mode allows the user to manually input the initial configuration of the Sudoku puzzle. The user can enter the numbers in the cells following Sudoku rules. Once the initial configuration is set, the player can choose to have the AI solve the puzzle by pressing the spacebar. The AI algorithm, using backtracking, attempts to solve the Sudoku puzzle generated by the user. If the puzzle is solvable, the AI provides the solution. If not, the AI informs the player that the puzzle is not solvable.

Mode 3: User (generate & solve)

In this mode, the user can manually input the initial configuration of the Sudoku puzzle. The user enters the numbers in the cells following Sudoku rules. After setting the initial configuration, the player can solve the puzzle by themselves. The AI is not involved in this mode, and the user relies on their own solving skills to complete the Sudoku puzzle.

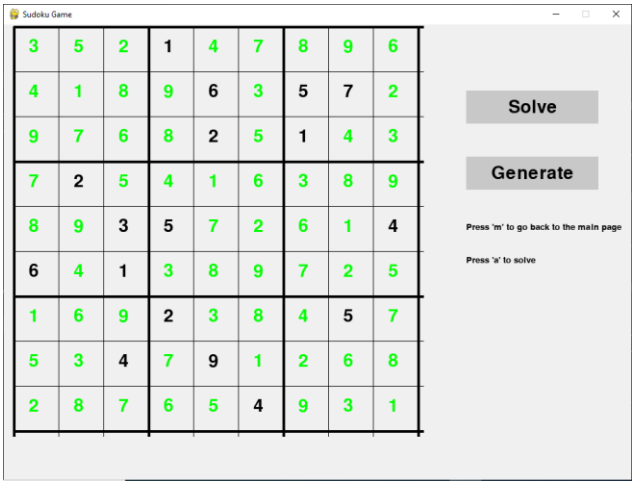
the user has 3 lives at the beginning if the user enters wrong value it will reduce till user lose.

Mode 4: User (solve)

In this mode, the Sudoku puzzle is already generated and displayed to the user. The user's task is to solve the puzzle by themselves, without any assistance from the AI. They can input numbers into the empty cells of the puzzle, following the Sudoku rules. The game checks the validity of the user's moves and provides feedback on whether the entered numbers comply with the Sudoku constraints. The player aims to solve the puzzle correctly without violating any Sudoku rules. The user has 3 lives at the beginning if the user enters wrong value it will reduce till user lose.

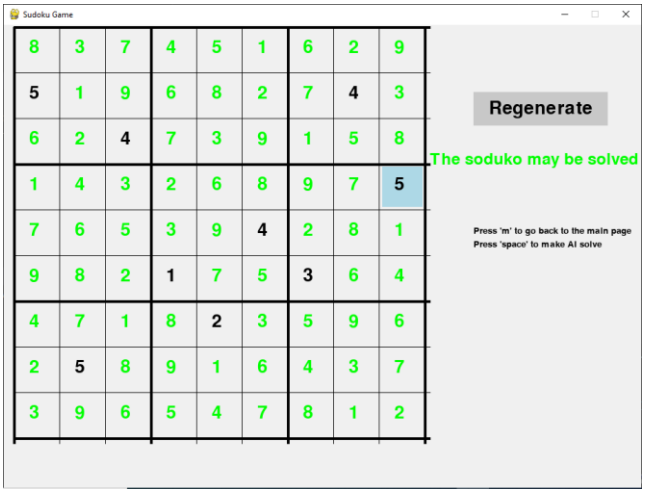
Sample runs

Mode 1:



- The corresponding Arc consistency trees in file (mode1.log)

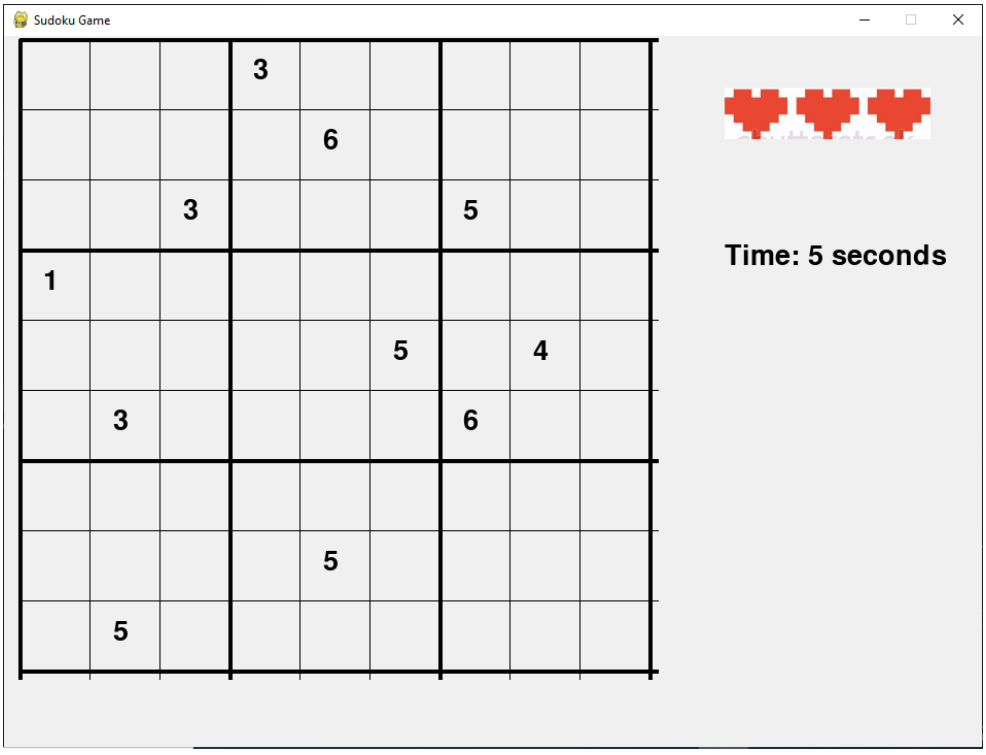
Mode 2 :



- The corresponding Arc consistency trees in file (mode2.log)

Mode 3 :

After user enter board



After lose 2 lives after violates the constraint twice