Sudoku Generator & Solver

ARNAV MISHRA
EECE 2160: COMPUTING FUNDAMENTALS FOR ENGINEERING
25 April 2023

Overview

This application serves as my final EECE: Computing Fundamentals for Engineering project. This algorithm creates solvable 9 by 9 sudokus and can also solve the same sudokus it creates or other sudokus provided by outside sources. The program also creates a view for the user that is a 9 by 9 grid that includes either a few numbers if a puzzle is generated or the solved grid with all 81 numbers in the solved version. The borders make it easy to see the nine squares that make up the grid and the numbers used are portrayed as blue. In terms of additional functionality, the sudoku generator does not show anything for numbers that are "missing" and guarantees solvable sudoku.

Application Design

The main attribute for the SUDOKU and DRAWSUDOKU class is a 2D ArrayList which is labeled as "board" and contains all 81 numbers inside the list of lists. Each sublist contains the numbers for a specific row. The two main classes for the project are SUDUKO which is the model algorithm and DRAWSUDUKO which acts as the GUI for the user. They are connected to each other in the solve_sudoku and create_board methods of the SUDOKU class. In those methods, a board is provided and a DRAWSUDOKU object is created. That object then uses the draw method to output the view to the user. The main methods are the following:

SUDOKU:

- solve_sudoku()
- create_board()
- is_Valid(board, row, col, num)
- create_solution(board)
- remove_nums()

DRAWSUDOKU:

- draw()

USEME

The system can essentially be run by two commands. First, the user will have to create a SUDOKU object and can insert a 2D list if they would like their own Sudoku to be solved (note: this sudoku must be a regular 9 by 9 sudoku board with zeroes in spots that are unknown). The user will then see a solved version of their sudoku in the view created. If the user does not insert anything as a parameter to the SUDOKU class, a randomly generated sudoku board will be shown as a pop-up as the view for the model.

Examples:

```
sudoku1 = SUDOKU()
sudoku2 = SUDOKU(listOfElementsAsListOfLists)
```

Next, if the user wants to solve the sudoku after they have created the sudoku, they can use the method: solve_sudoku to create a solved version of the sudoku.

Example:

```
sudoku1.solve_sudoku()
```

External Resources

The module used for this project was the “random” module for shuffling numbers and creating random sudokus every time. The matplot library was also used, specifically pyplot for plotting the numbers in a grid-like fashion and for creating the view. Previously, I already had quite a bit of knowledge about sudokus and would do them for fun when I was younger, so no other tools for generally solving/generating sudokus were used.

Reflection

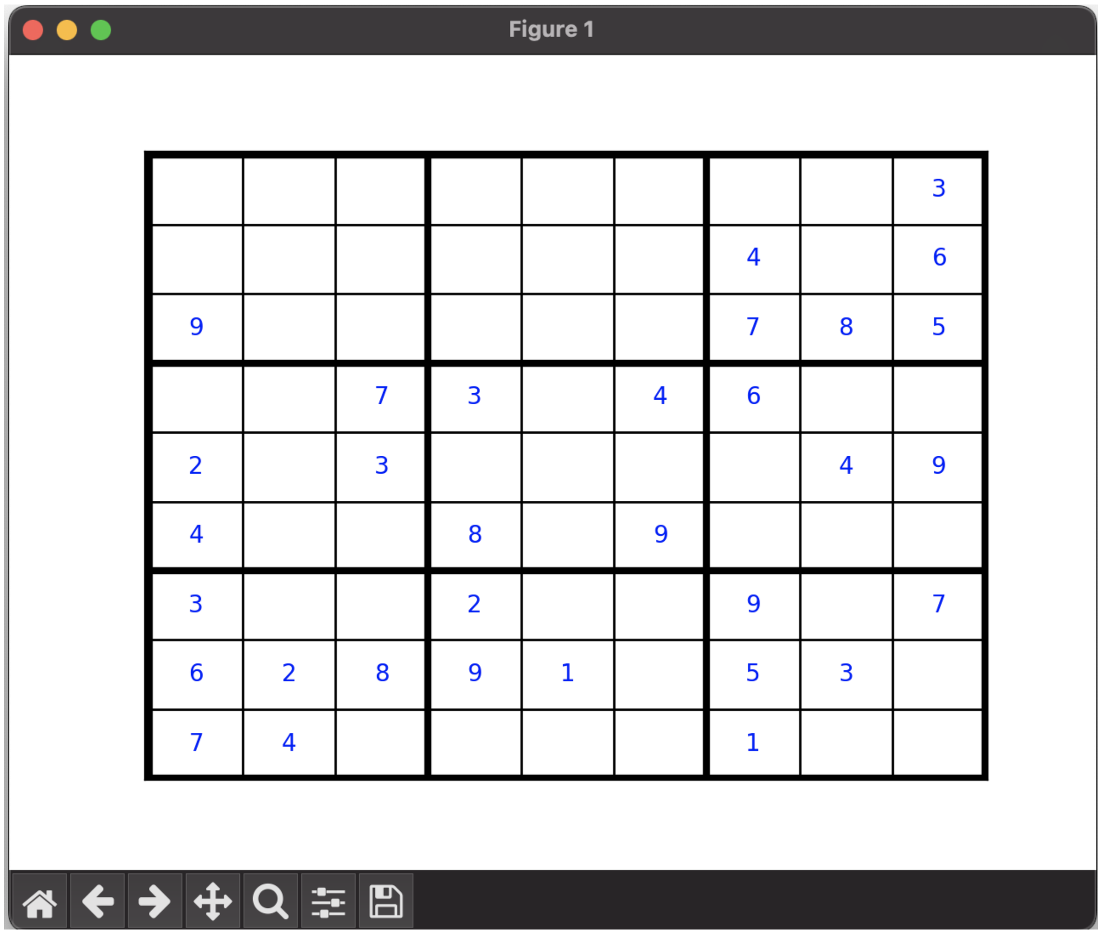
Overall, I found this assignment a great way to destress from finals week. I had a lot of fun experimenting with different methods and trying to find the best way to integrate a solver and generator in one class. It was also cool to play around with the plotting feature and create a sudoku board that looked good and was user-friendly. While most of this project was already applying what I had known, I developed a deeper understanding of how I can create methods in a class that works coherently and how to create my design to be as minimal and effective as possible. I also learned more about docstrings which was a field that I was not very familiar with beforehand.

If I had more time for this project, I would perhaps look at future extensions for this sudoku generator and solver. First, I would have liked to create a program that allows the user to play on the board given and directly enter numbers one by one onto the board. Another idea I had was implementing a sudoku board that also checked for the diagonals of the board and created or solved boards with the constraint that the diagonals of the board had to have all numbers 1-9 without repetition in them. This would push me to use numpy instead of regular 2D arrays as working with diagonals is made easier with the numpy module.

Some advice I would give to EECE 2140 students is to start earlier and create projects that interest them. I started this project a bit late and was not creative enough to make my own idea. Instead of using the examples provided (like I did), I would suggest the students find a problem they had faced their freshman year and perhaps look to create a simple algorithm that could perhaps have helped them. This helps students not only with coding but also thinking about the world we live in and how we can use computer science to solve everyday challenges.

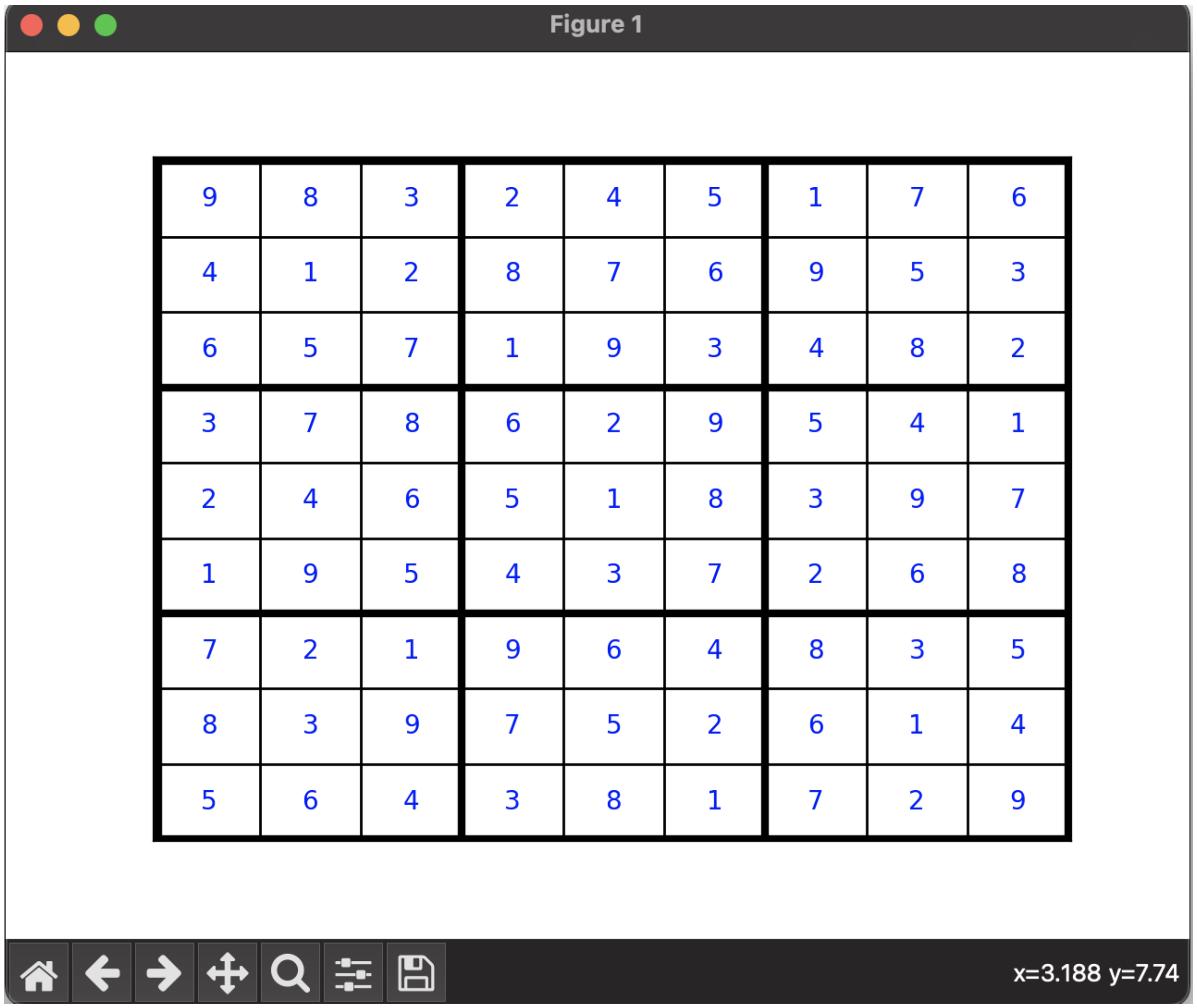
Note: Another part I could improve on is using a seeded random generator rather than the pseudo-random generator I have currently. The description does not ask for truly unique sudokus and using random with shuffle always creates new sudokus for me, so I decided to stick with that.

Appendix



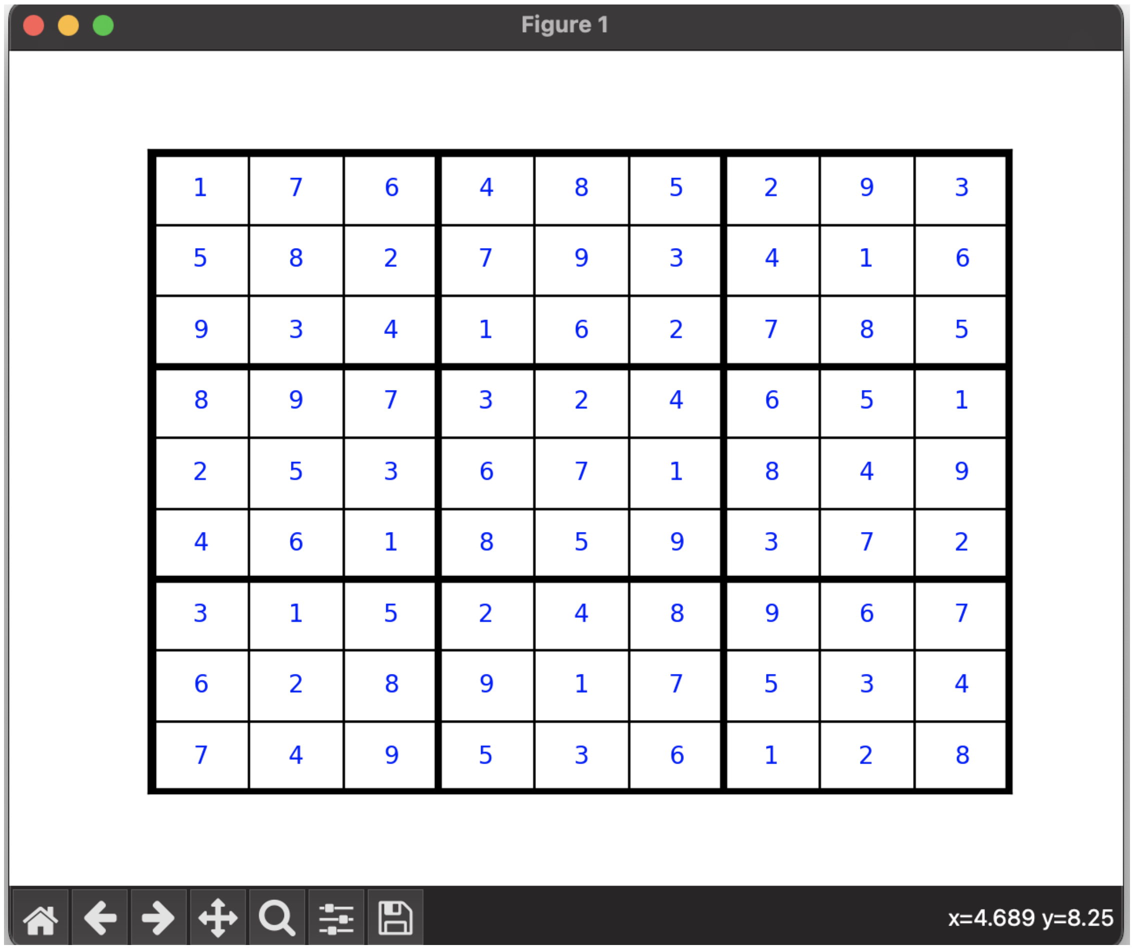
								3
						4		6
9						7	8	5
		7	3		4	6		
2		3					4	9
4			8		9			
3			2			9		7
6	2	8	9	1		5	3	
7	4					1		

Appendix A: Generating a random sudoku board with no parameter given
Command: sudoku1 = SUDOKU()



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

Appendix B: Generating a solved Sudoku board assuming a valid Sudoku board is provided
Command: sudoku2 = SUDOKU([[9, 0, 0, 0, 0, 0, 0, 0, 6], [0, 1, 0, 8, 7, 0, 0, 0, 0], [0, 5, 7, 1, 0, 0, 4, 0, 0], [0, 0, 0, 0, 0, 0, 0, 4, 1], [2, 4, 0, 5, 0, 0, 3, 9, 0], [0, 0, 5, 4, 0, 7, 0, 0, 0], [0, 0, 0, 9, 0, 4, 0, 3, 0], [0, 3, 9, 0, 0, 2, 6, 0, 0], [0, 0, 0, 0, 8, 1, 0, 0, 0]])
Command: sudoku2.solve_sudoku



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

Appendix C: Solving a given sudoku board
Command: sudoku1.solve_sudoku

```
import random
import matplotlib.pyplot as plt

class SUDOKU:

   """
   A class used to represent a 9 x 9 Sudoku

   ...

   Attributes
   ----------
   board : List of List
       a List of Lists that represents the numbers in the sudoku with each
       inner list representing a row.

   Methods
   -------
   solve_sudoku()
       Creates a solution for the Sudoku and displays it using the DrawSudoku class.

   create_board()
       Generates a new Sodoku board with a possible solution and randomly eliminates
numbers from the board
       by turning them into 0. The board is still solvable, however.

   is_Valid(board, row, col, num)
       Checks to see if a number put onto the board is valid by the rules of Sudoku.
It checks all
       rows, column, and squares that the number is in to make sure it is not
repeated. If it is a valid
       number to use, the method returns True. Otherwise, it returns False.

   next_empty(board)
       Finds and returns the next square that has a 0 on the board

   solve(board)
       Solves the Sudoku board using backtracking and recursion.
```

```
create_solution(board)
    Generates and returns a solved version of the Sudoku board.

get_nums_squares(board)
    Returns a list of all squares that are not empty(are non-zero)


remove_nums()
    Removes numbers randomly while still ensuring that the sudoku is solvable.
    Changes some values to 0.

"""

def __init__(self,board=None):

    """Checks if the user wants to generate a new Sudoku board or solve one. It
        also checks if the user has entered a 9 by 9 Sudoku board.

    If the argument `board` isn't passed in, it is assumed the user wants to create
    a new unsolved Sudoku board.

    Parameters
    ----------
    board : List of Lists, optional
        A List of Lists that represents the numbers in the sudoku with each
        inner list representing a row.

    Raises
    ------
    ValueError
        If invalid sudoku board is provided.
    """

    self.solutions = 0
    if board == None:
        self.board = [[0 for x in range(9)] for y in range(9)]
        self.create_board()

    else:
        if len(board[0]) == 9 and len(board) == 9:
            self.board = board
```

```
                self.solve_sudoku()
            else:
                raise ValueError("Invalid sudoku board is provided")

    def solve_sudoku(self):

        """
        Solves the Sudoku and creates a DRAWSUDOKU object to make a view for the user.
        """

        self.create_solution(self.board)
        sudoku2 = DRAWSUDOKU(self.board)
        sudoku2.draw()

    def create_board(self):

        """
        Generates a new board for the user and presents the unsolved version by
creating a DRAWSUDOKU object.
        """

        self.create_solution(self.board)
        self.remove_nums()
        sudoku2 = DRAWSUDOKU(self.board)
        sudoku2.draw()



    def is_valid(self, board, row, col, num):

        """Checks if the computer generated guess is feasible for the Sudoku puzzle

        It checks if the number is unique in its row, column, and square.

        Parameters
        ----------
        board : List of Lists, optional
            A List of Lists that represents the numbers in the sudoku with each
            inner list representing a row.
        row : int
            A number that represents the row the number: "num" is in.
        col : int
            A number that represents the column the number: "num" is in.
        num : int
            A number from 1-9 that is the current guess that will be checked if it is a
valid guess

        """

        row1 = (row // 3) * 3
        col1 = (col // 3)  * 3
        # checking rows
        if num in board[row]:
            return False
        # checking columns
        for x in range(9):
            if board[x][col] == num:
                return False
        #checking squares
        for x in range(row1, (row1 + 3)):
            for y in range(col1, (col1 + 3)):
                if board[x][y] == num:
                    return False
        return True

    def next_empty(self,board):

        """
        Finds and returns the next empty element that will allow the computer to guess
a number over it.
        """

        for x in range(len(board)):
            for y in range(len(board[0])):
                if board[x][y] == 0:
                    return (x, y)

    def solve(self, board):

        """Solves the Sudoku board using backtracking and recursion.

        Parameters
        ----------
        board : List of Lists, optional
            A List of Lists that represents the numbers in the sudoku with each
            inner list representing a row.

        This algorithm recursively calls itself and runs through every possible outcome
to
        check if all of the numbers in the board are consistent with the solution. This
method
        is specifically for the remove function as it preserves the zeros as a seperate
board.


        """


        for x in range(81):
            row = x // 9
            col = x % 9
            if board[row][col] == 0:
                for x in range(1, 10):
                    if self.is_valid(board, row, col, x):
                        board[row][col] = x
                        if not self.next_empty(board):
                            self.solutions += 1
                            break
                        else:
                            if self.solve(board):
                                return True
                break
        board[row][col] = 0
        return False

    def create_solution(self, board):

        """Generates and returns a solved version of the Sudoku board.

        Parameters
        ----------
        board : List of Lists, optional
            A List of Lists that represents the numbers in the sudoku with each
            inner list representing a row.

        This algorithm recursively calls itself and runs through every possible outcome
to
```

```
        check if all of the numbers in the board are consistent with the solution.

        """

        nums = list(range(1, 10))
        for x in range(0,81):
            row = x // 9
            col = x % 9
            if board[row][col]==0:
                random.shuffle(nums)
                for num in nums:
                    if self.is_valid(board,row,col,num):

                        board[row][col]=num
                        if not self.next_empty(board):
                            return True
                        else:
                            if self.create_solution(board):
                                #if the grid is full
                                return True
                break
        board[row][col] = 0
        return False

    def get_nums_squares(self,board):

        """Returns a list of all squares that are not empty(are non-zero)

        Parameters
        ----------
        board : List of Lists, optional
            A List of Lists that represents the numbers in the sudoku with each
            inner list representing a row.

        """

        num_squares = []
        for x in range(len(board)):
            for y in range(len(board)):
                if board[x][y] != 0:
                    num_squares.append((x, y))
        random.shuffle(num_squares)
        return num_squares

    def remove_nums(self):

        """
        Removes numbers randomly while still ensuring that the sudoku is solvable.
        Changes some values to 0. It checks through all the possible outcomes and if
        the method determines that another number needs to be shown, it does so
randomly.
        """

        nums_squares = self.get_nums_squares(self.board)
        count_squares = len(nums_squares)
        trials = 9

        # Needs to be at least 17 clues to make a unique Sudoku solution
        while trials > 0 and count_squares >= 17:

            row, col = nums_squares.pop()
            count_squares -= 1
            removed_square = self.board[row][col]
            self.board[row][col] = 0

            # changes the number of solutions to zero
            self.solutions = 0
            self.solve(self.board)

            # Shows one more number if there are more than 1 possible Sudoku solutions
possible in the current layout :
            if self.solutions != 1:
                self.board[row][col] = removed_square
                count_squares += 1
                trials -= 1
        return


class DRAWSUDOKU:

    """
    A class used to represent the view for a SUDOKU

    ...
    Attributes
    ----------
    board : List of List
        a List of Lists that represents the numbers in the sudoku with each
        inner list representing a row.

    Methods
    -------

    draw()
        draws the board with all of the numbers in the Sudoku puzzle. Any 0 will not be
shown on the Sudoku grid.

    """



    def __init__(self,board):

        """Checks to see if a valid Sudoku board was provided and initializes the
board.

        Parameters
        ----------

        board : List of Lists, optional
            A List of Lists that represents the numbers in the sudoku with each
            inner list representing a row.

        Raises
        ------
        ValueError
            If invalid sudoku board is provided.
        """

        if len(board[0]) == 9 and len(board) == 9:
                self.board = board

        else:
            raise ValueError("Invalid sudoku board is provided")

    def draw(self):
        """
        Generates a view of the Sudoku board with blank spaces if the value is 0.
        """

        plt.figure()
        plt.grid()
        for x in range(10):
            if x % 3 == 0:
                width = 3
            else:
                width = 1
            plt.plot([0, 9], [x, x], 'k', lw = width)
            plt.plot([x, x], [0, 9], 'k', lw = width)

        for x in range(len(self.board)):
            for y in range(len(self.board[0])):
                if self.board[x][y] != 0:
                    plt.text(x + 0.5, y + 0.5, str(self.board[x][y]), color = 'blue',
ha = 'center', va = 'center')

        plt.axis([-0.05, 9.025, -0.025, 9.05])
        plt.xticks([])
        plt.yticks([])
        plt.show()

sudoku1 = SUDOKU()
sudoku1.solve_sudoku()
```

Appendix D: Full Code for SUDOKU and DRAWSUDOKU classes