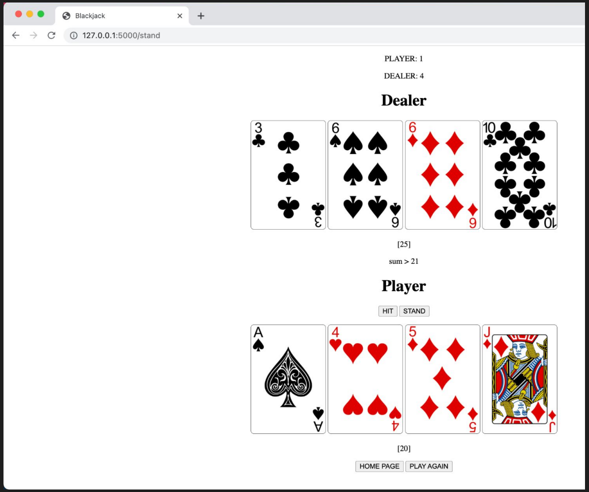
Ashley Byrne

IMP Project, June 2022

Background

Originally for my IMP project, I was programming a website where the user can play the card game Blackjack. At the same time I was also working on writing a program to solve Sudoku. Even though I implemented Blackjack, in the end, I was more excited to share and talk about the Sudoku solver that I had been working on.

Blackjack game running locally



PLAYER: 1

DEALER: 4

Dealer

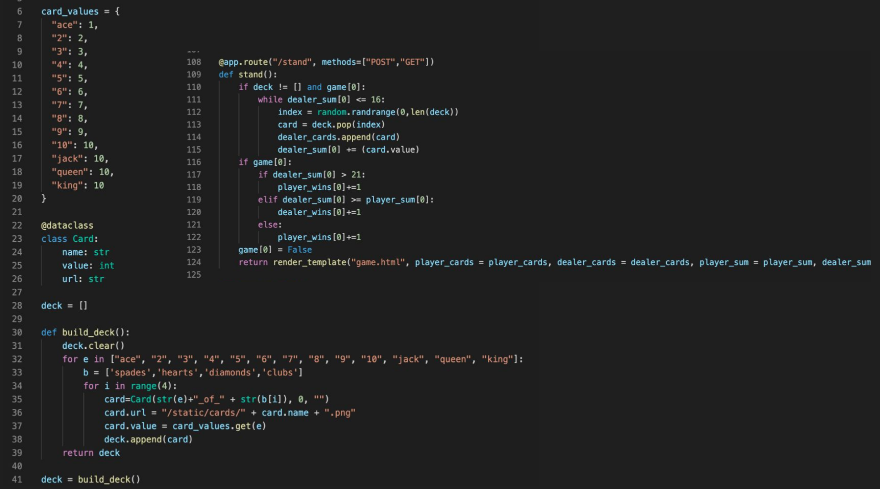
[25]

sum > 21

Player

[20]

Code

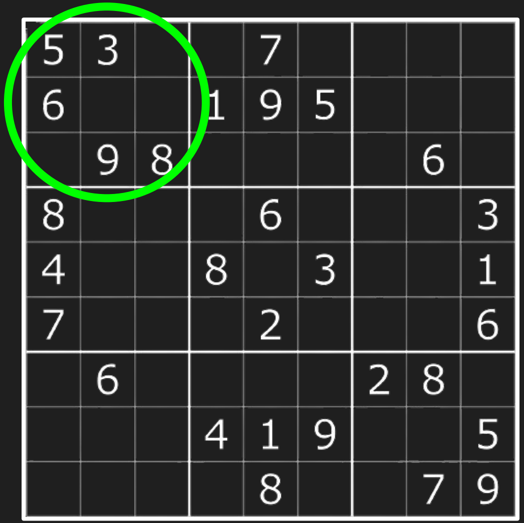


```python
card_values = {
  "ace": 1,
  "2": 2,
  "3": 3,
  "4": 4,
  "5": 5,
  "6": 6,
  "7": 7,
  "8": 8,
  "9": 9,
  "10": 10,
  "jack": 10,
  "queen": 10,
  "king": 10
}

@dataclass
class Card:
    name: str
    value: int
    url: str

deck = []

def build_deck():
    deck.clear()
    for e in ["ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "jack", "queen", "king"]:
        b = ['spades','hearts','diamonds','clubs']
        for i in range(4):
            card=Card(str(e)+"_of_" + str(b[i]), 0, "")
            card.url = "/static/cards/" + card.name + ".png"
            card.value = card_values.get(e)
            deck.append(card)
    return deck

deck = build_deck()
```

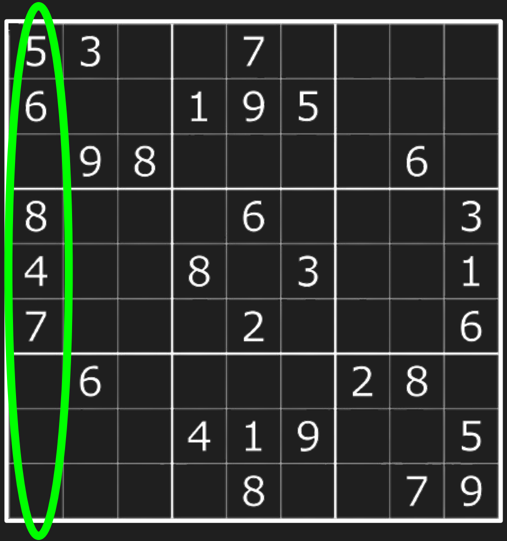
```python
@app.route("/stand", methods=["POST","GET"])
def stand():
    if deck != [] and game[0]:
        while dealer_sum[0] <= 16:
            index = random.randrange(0,len(deck))
            card = deck.pop(index)
            dealer_cards.append(card)
            dealer_sum[0] += (card.value)
    if game[0]:
        if dealer_sum[0] > 21:
            player_wins[0]+=1
        elif dealer_sum[0] >= player_sum[0]:
            dealer_wins[0]+=1
        else:
            player_wins[0]+=1
    game[0] = False
    return render_template("game.html", player_cards = player_cards, dealer_cards = dealer_cards, player_sum = player_sum, dealer_sum

```

Rules of Sudoku

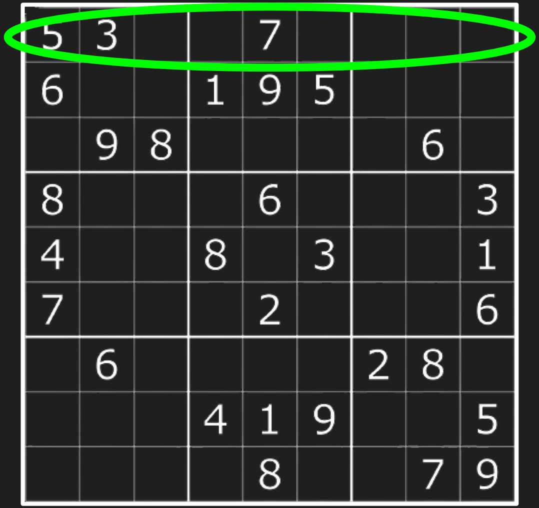
A Sudoku board is a grid (usually 9x9) with some numbers already placed in the smaller cells. Each box (3x3), row, and column must contain the numbers 1-9 only once. The puzzle is solved when the entire board is filled in.



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



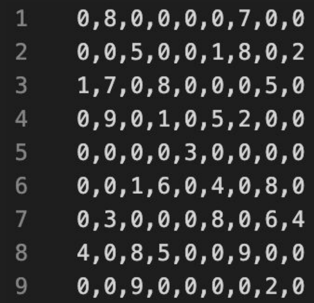
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



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

Steps

1. Write a program to solve a Sudoku board given clean data; input read from a CSV (comma-separated values) file
   DONE



```
0,8,0,0,0,0,7,0,0
0,0,5,0,0,1,8,0,2
1,7,0,8,0,0,0,5,0
0,9,0,1,0,5,2,0,0
0,0,0,0,3,0,0,0,0
0,0,1,6,0,4,0,8,0
0,3,0,0,0,8,0,6,4
4,0,8,5,0,0,9,0,0
0,0,9,0,0,0,0,2,0
```

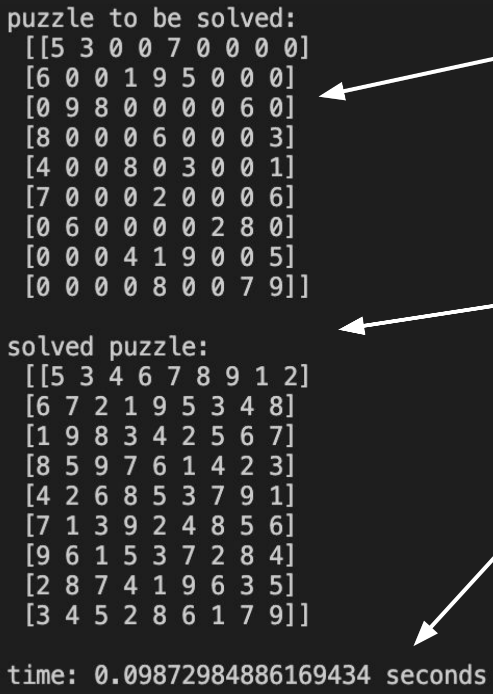
2. Write a program to read data from an preprocessed image of a Sudoku board using computer vision and digit recognition
   DONE

3. Use a live image from computer webcam/phone and convert the working program into a phone app
   WORK IN PROGRESS

Step 1

Sudoku Solver in Action

Example #1



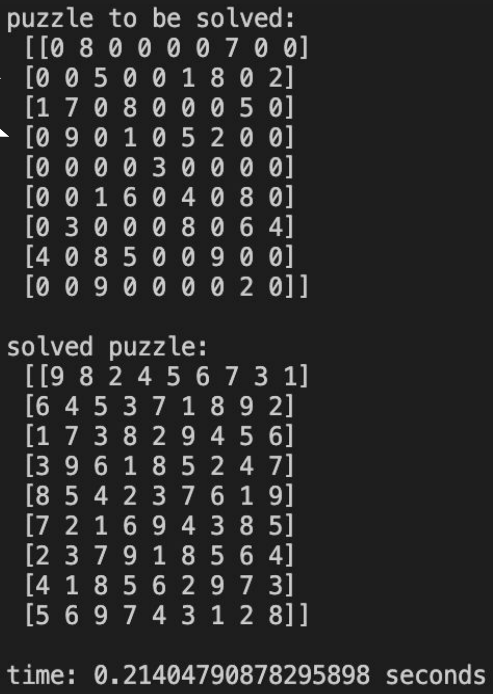
```
puzzle to be solved:
 [[5 3 0 0 7 0 0 0 0]
 [6 0 0 1 9 5 0 0 0]
 [0 9 8 0 0 0 0 6 0]
 [8 0 0 0 6 0 0 0 3]
 [4 0 0 8 0 3 0 0 1]
 [7 0 0 0 2 0 0 0 6]
 [0 6 0 0 0 0 2 8 0]
 [0 0 0 4 1 9 0 0 5]
 [0 0 0 0 8 0 0 7 9]]

solved puzzle:
 [[5 3 4 6 7 8 9 1 2]
 [6 7 2 1 9 5 3 4 8]
 [1 9 8 3 4 2 5 6 7]
 [8 5 9 7 6 1 4 2 3]
 [4 2 6 8 5 3 7 9 1]
 [7 1 3 9 2 4 8 5 6]
 [9 6 1 5 3 7 2 8 4]
 [2 8 7 4 1 9 6 3 5]
 [3 4 5 2 8 6 1 7 9]]

time: 0.09872984886169434 seconds
```

Empty “cells” are set to 0

Solution

Time taken to solve puzzle

Example #2

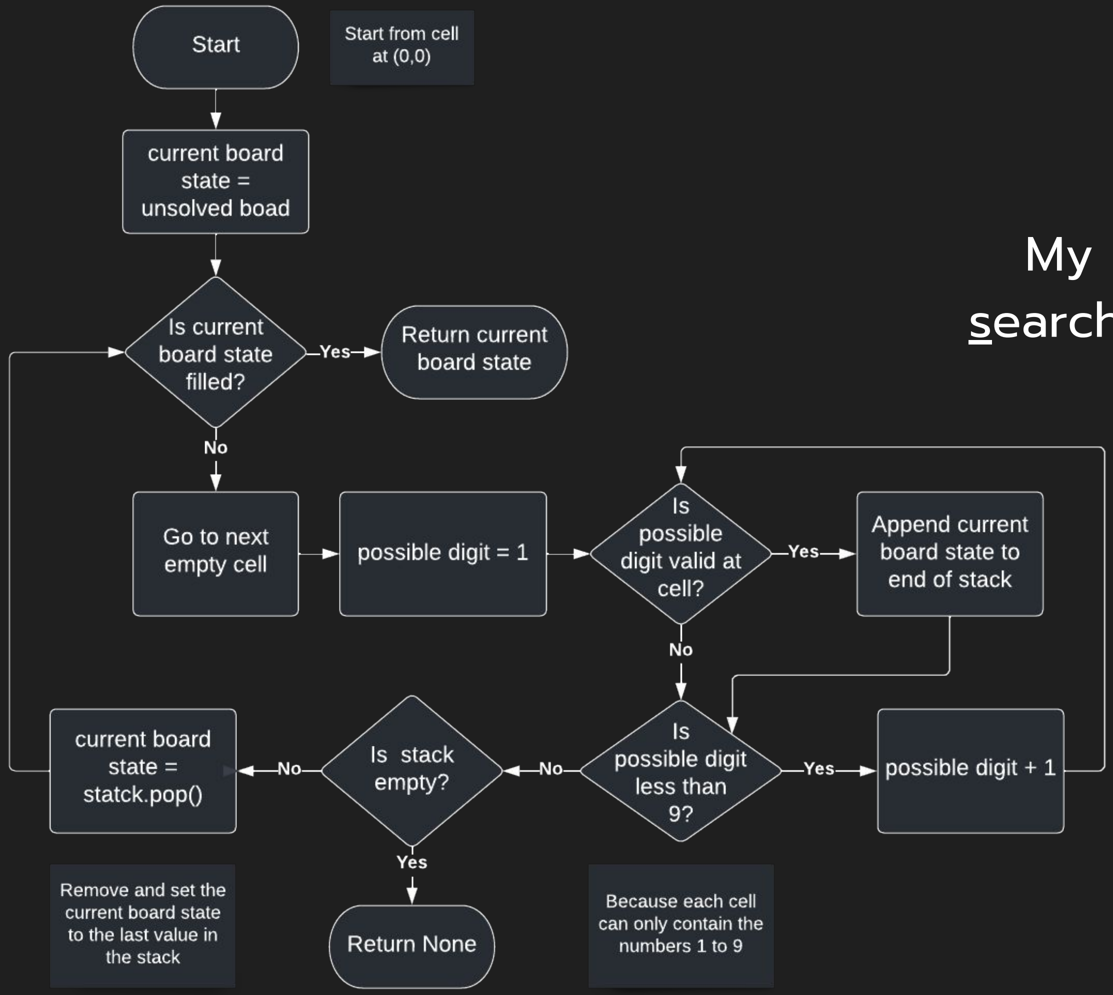


```
puzzle to be solved:
 [[0 8 0 0 0 0 7 0 0]
 [0 0 5 0 0 1 8 0 2]
 [1 7 0 8 0 0 0 5 0]
 [0 9 0 1 0 5 2 0 0]
 [0 0 0 0 3 0 0 0 0]
 [0 0 1 6 0 4 0 8 0]
 [0 3 0 0 0 8 0 6 4]
 [4 0 8 5 0 0 9 0 0]
 [0 0 9 0 0 0 0 2 0]]

solved puzzle:
 [[9 8 2 4 5 6 7 3 1]
 [6 4 5 3 7 1 8 9 2]
 [1 7 3 8 2 9 4 5 6]
 [3 9 6 1 8 5 2 4 7]
 [8 5 4 2 3 7 6 1 9]
 [7 2 1 6 9 4 3 8 5]
 [2 3 7 9 1 8 5 6 4]
 [4 1 8 5 6 2 9 7 3]
 [5 6 9 7 4 3 1 2 8]]

time: 0.21404790878295898 seconds
```

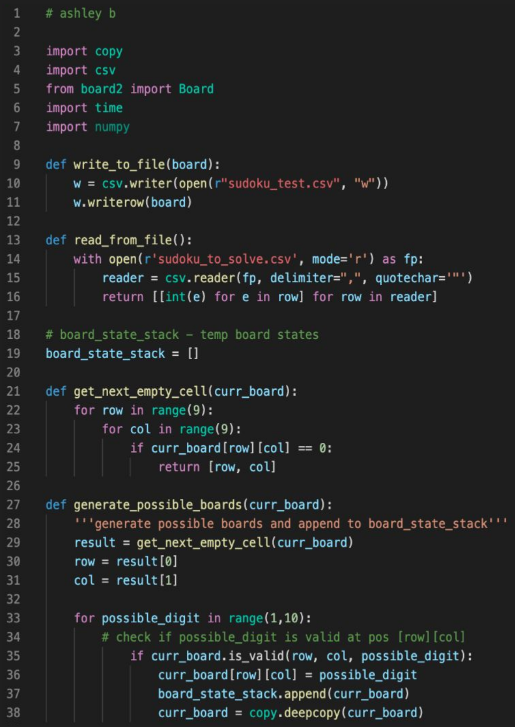
Step 1

Code Flowchart

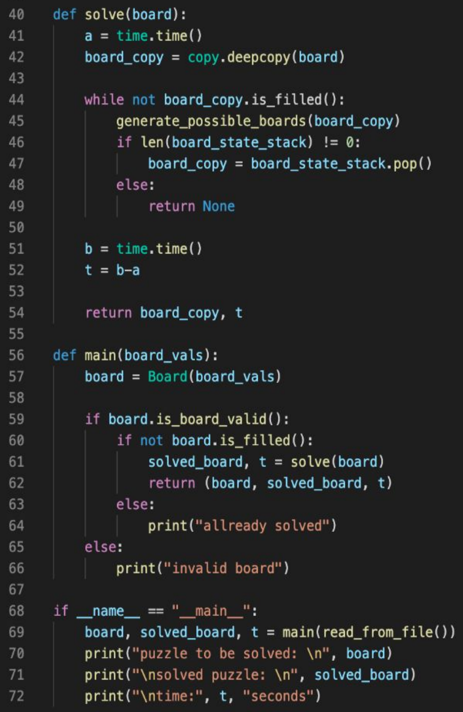
My program uses the depth-first search (DFS) algorithm to generate a solution


Start
Start from cell at (0,0)
current board state = unsolved boad
Is current board state filled?
Yes
Return current board state
No
Go to next empty cell
possible digit = 1
Is possible digit valid at cell?
Yes
Append current board state to end of stack
No
Is possible digit less than 9?
Yes
possible digit + 1
No
Is stack empty?
Yes
Return None
No
current board state = statck.pop()
Remove and set the current board state to the last value in the stack
Because each cell can only contain the numbers 1 to 9

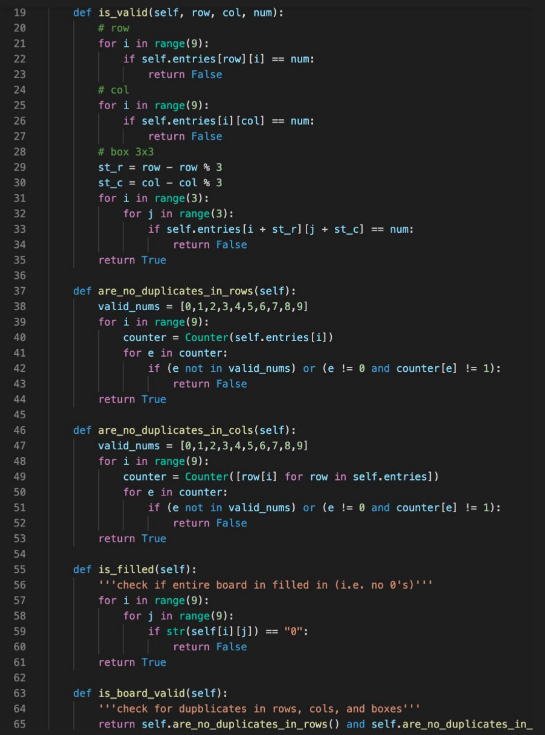
Step 1



```python
# ashley b

import copy
import csv
from board2 import Board
import time
import numpy

def write_to_file(board):
    w = csv.writer(open(r"sudoku_test.csv", "w"))
    w.writerow(board)

def read_from_file():
    with open(r'sudoku_to_solve.csv', mode='r') as fp:
        reader = csv.reader(fp, delimiter=",", quotechar='"')
        return [[int(e) for e in row] for row in reader]

# board_state_stack - temp board states
board_state_stack = []

def get_next_empty_cell(curr_board):
    for row in range(9):
        for col in range(9):
            if curr_board[row][col] == 0:
                return [row, col]

def generate_possible_boards(curr_board):
    '''generate possible boards and append to board_state_stack'''
    result = get_next_empty_cell(curr_board)
    row = result[0]
    col = result[1]

    for possible_digit in range(1,10):
        # check if possible_digit is valid at pos [row][col]
            if curr_board.is_valid(row, col, possible_digit):
                curr_board[row][col] = possible_digit
                board_state_stack.append(curr_board)
                curr_board = copy.deepcopy(curr_board)
```



```python
def solve(board):
    a = time.time()
    board_copy = copy.deepcopy(board)

    while not board_copy.is_filled():
        generate_possible_boards(board_copy)
        if len(board_state_stack) != 0:
            board_copy = board_state_stack.pop()
        else:
            return None

    b = time.time()
    t = b-a

    return board_copy, t

def main(board_vals):
    board = Board(board_vals)

    if board.is_board_valid():
        if not board.is_filled():
            solved_board, t = solve(board)
            return (board, solved_board, t)
        else:
            print("allready solved")
    else:
        print("invalid board")

if __name__ == "__main__":
    board, solved_board, t = main(read_from_file())
    print("puzzle to be solved: \n", board)
    print("\nsolved puzzle: \n", solved_board)
    print("\ntime:", t, "seconds")
```



```python
    def is_valid(self, row, col, num):
        # row
        for i in range(9):
            if self.entries[row][i] == num:
                return False
        # col
        for i in range(9):
            if self.entries[i][col] == num:
                return False
        # box 3x3
        st_r = row - row % 3
        st_c = col - col % 3
        for i in range(3):
            for j in range(3):
                if self.entries[i + st_r][j + st_c] == num:
                    return False
        return True

    def are_no_duplicates_in_rows(self):
        valid_nums = [0,1,2,3,4,5,6,7,8,9]
        for i in range(9):
            counter = Counter(self.entries[i])
            for e in counter:
                if (e not in valid_nums) or (e != 0 and counter[e] != 1):
                    return False
        return True

    def are_no_duplicates_in_cols(self):
        valid_nums = [0,1,2,3,4,5,6,7,8,9]
        for i in range(9):
            counter = Counter([row[i] for row in self.entries])
            for e in counter:
                if (e not in valid_nums) or (e != 0 and counter[e] != 1):
                    return False
        return True

    def is_filled(self):
        '''check if entire board in filled in (i.e. no 0's)'''
        for i in range(9):
            for j in range(9):
                if str(self[i][j]) == "0":
                    return False
        return True

    def is_board_valid(self):
        '''check for dupblicates in rows, cols, and boxes'''
        return self.are_no_duplicates_in_rows() and self.are_no_duplicates_in_
```

This code generates the solution to a Sudoku puzzle

Step 2

What are OpenCV & Tesseract?

OpenCV is an open source computer vision software that can process images.

Tesseract OCR (Optical Character Recognition) is also an open source software library that recognises digits and letters in images.

OpenCV

Learn more at https://opencv.org/

Tesseract OCR

Learn more at https://github.com/tesseract-ocr/tesseract

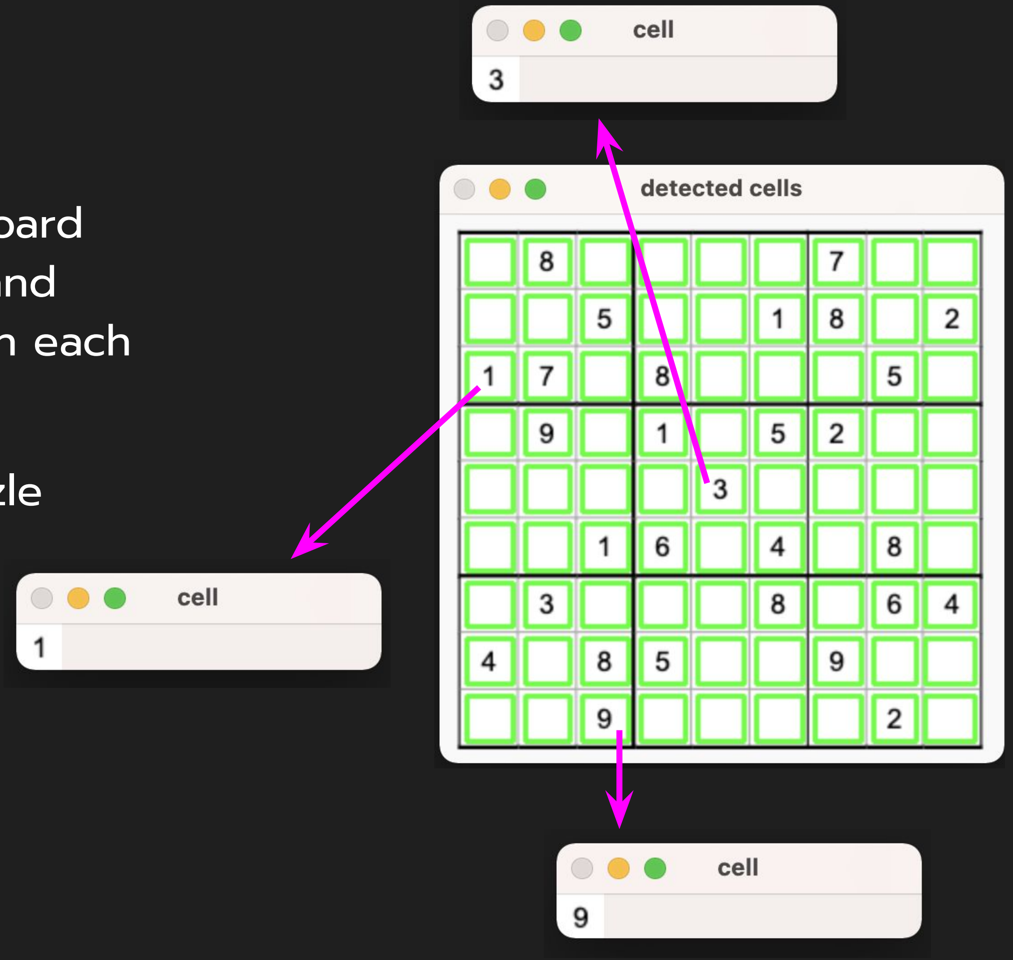
Step 2

Code Explanation

My program breaks the Sudoku board image into 81 mini-images (cells) and uses Tesseract digit recognition on each cell to identify the digit.

Then, my program solves the puzzle using the recognized digits to get the digits in the solution.

Finlay, it overlays the solution on the original image. *(see next slide)*



cell

3

detected cells

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

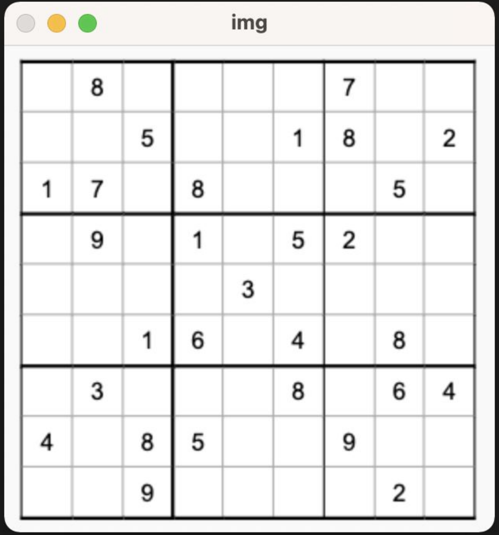
cell

1

cell

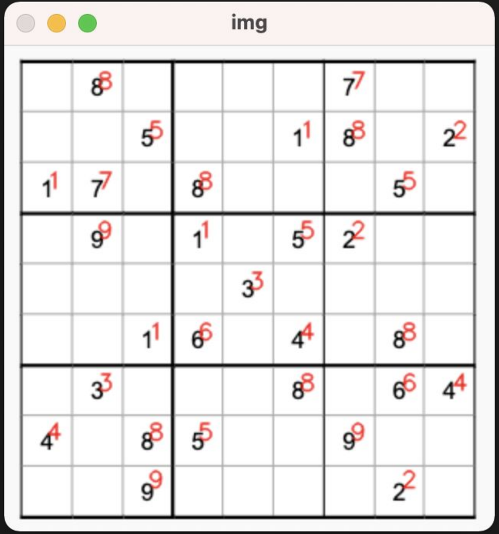
9

Overlaying Solution on Image



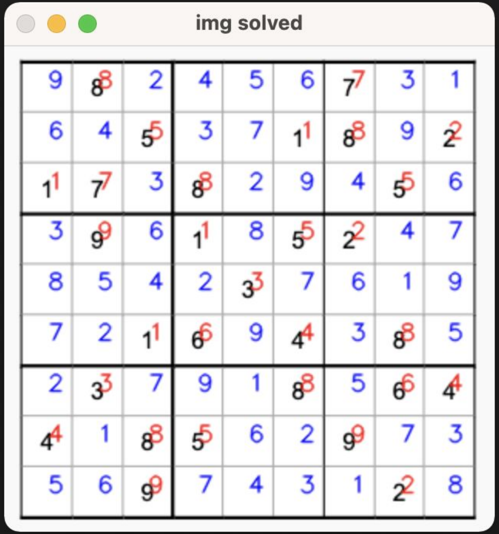
	8					7		
		5			1	8		2
1	7		8				5	
	9		1		5	2		
				3				
		1	6		4		8	
	3				8		6	4
4		8	5			9		
		9					2	

Original image



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

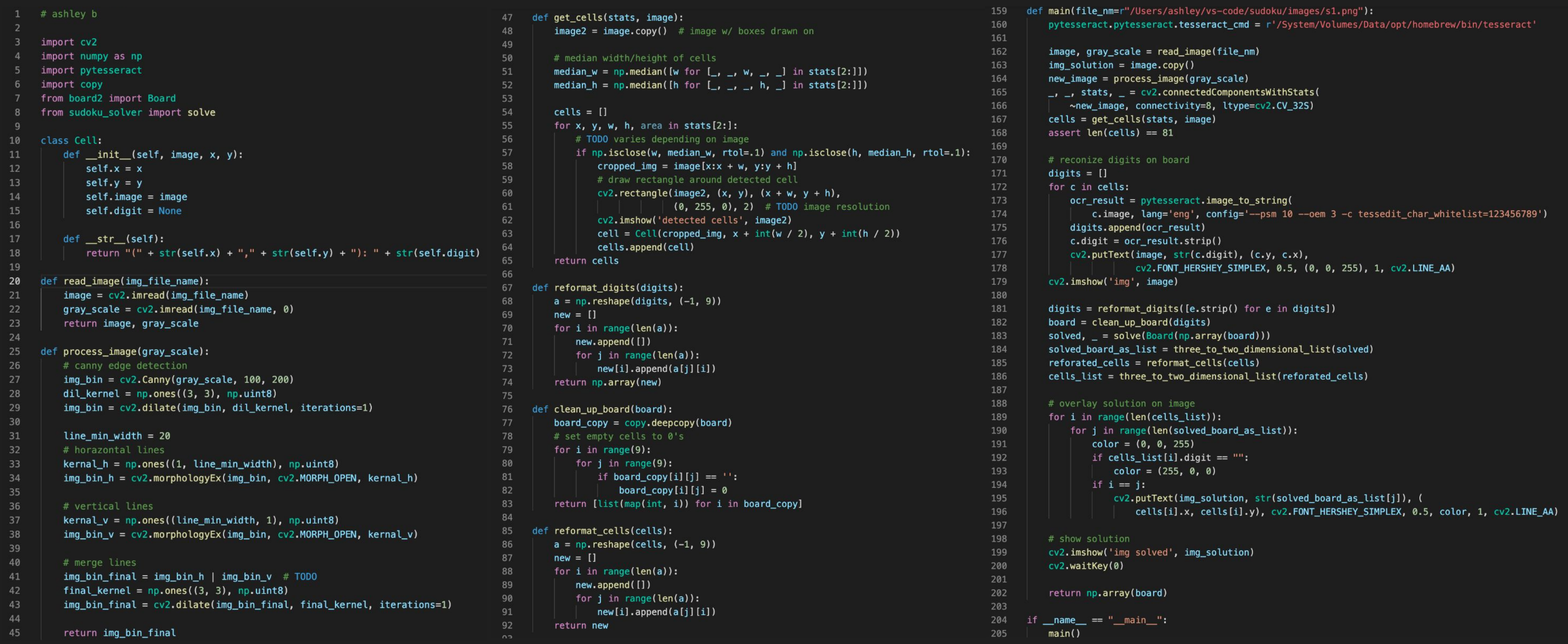
Recognized digits
(in red)



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

Solved board
(in blue)

Step 2



```python
# ashley b

import cv2
import numpy as np
import pytesseract
import copy
from board2 import Board
from sudoku_solver import solve

class Cell:
    def __init__(self, image, x, y):
        self.x = x
        self.y = y
        self.image = image
        self.digit = None

    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + "): " + str(self.digit)

def read_image(img_file_name):
    image = cv2.imread(img_file_name)
    gray_scale = cv2.imread(img_file_name, 0)
    return image, gray_scale

def process_image(gray_scale):
    # canny edge detection
    img_bin = cv2.Canny(gray_scale, 100, 200)
    dil_kernel = np.ones((3, 3), np.uint8)
    img_bin = cv2.dilate(img_bin, dil_kernel, iterations=1)

    line_min_width = 20
    # horazontal lines
    kernal_h = np.ones((1, line_min_width), np.uint8)
    img_bin_h = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN, kernal_h)

    # vertical lines
    kernal_v = np.ones((line_min_width, 1), np.uint8)
    img_bin_v = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN, kernal_v)

    # merge lines
    img_bin_final = img_bin_h | img_bin_v  # TODO
    final_kernel = np.ones((3, 3), np.uint8)
    img_bin_final = cv2.dilate(img_bin_final, final_kernel, iterations=1)

    return img_bin_final
```

```python
def get_cells(stats, image):
    image2 = image.copy()  # image w/ boxes drawn on

    # median width/height of cells
    median_w = np.median([w for [_, _, w, _, _] in stats[2:]])
    median_h = np.median([h for [_, _, _, h, _] in stats[2:]])

    cells = []
    for x, y, w, h, area in stats[2:]:
        # TODO varies depending on image
        if np.isclose(w, median_w, rtol=.1) and np.isclose(h, median_h, rtol=.1):
            cropped_img = image[x:x + w, y:y + h]
            # draw rectangle around detected cell
            cv2.rectangle(image2, (x, y), (x + w, y + h),
                          (0, 255, 0), 2)  # TODO image resolution
            cv2.imshow('detected cells', image2)
            cell = Cell(cropped_img, x + int(w / 2), y + int(h / 2))
            cells.append(cell)
    return cells

def reformat_digits(digits):
    a = np.reshape(digits, (-1, 9))
    new = []
    for i in range(len(a)):
        new.append([])
        for j in range(len(a)):
            new[i].append(a[j][i])
    return np.array(new)

def clean_up_board(board):
    board_copy = copy.deepcopy(board)
    # set empty cells to 0's
    for i in range(9):
        for j in range(9):
            if board_copy[i][j] == '':
                board_copy[i][j] = 0
    return [list(map(int, i)) for i in board_copy]

def reformat_cells(cells):
    a = np.reshape(cells, (-1, 9))
    new = []
    for i in range(len(a)):
        new.append([])
        for j in range(len(a)):
            new[i].append(a[j][i])
    return new
```

```python
def main(file_nm=r"/Users/ashley/vs-code/sudoku/images/s1.png"):
    pytesseract.pytesseract.tesseract_cmd = r'/System/Volumes/Data/opt/homebrew/bin/tesseract'

    image, gray_scale = read_image(file_nm)
    img_solution = image.copy()
    new_image = process_image(gray_scale)
    _, _, stats, _ = cv2.connectedComponentsWithStats(
        ~new_image, connectivity=8, ltype=cv2.CV_32S)
    cells = get_cells(stats, image)
    assert len(cells) == 81

    # reconize digits on board
    digits = []
    for c in cells:
        ocr_result = pytesseract.image_to_string(
            c.image, lang='eng', config='--psm 10 --oem 3 -c tessedit_char_whitelist=123456789')
        digits.append(ocr_result)
        c.digit = ocr_result.strip()
        cv2.putText(image, str(c.digit), (c.y, c.x),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
    cv2.imshow('img', image)

    digits = reformat_digits([e.strip() for e in digits])
    board = clean_up_board(digits)
    solved, _ = solve(Board(np.array(board)))
    solved_board_as_list = three_to_two_dimensional_list(solved)
    reforated_cells = reformat_cells(cells)
    cells_list = three_to_two_dimensional_list(reforated_cells)

    # overlay solution on image
    for i in range(len(cells_list)):
        for j in range(len(solved_board_as_list)):
            color = (0, 0, 255)
            if cells_list[i].digit == "":
                color = (255, 0, 0)
            if i == j:
                cv2.putText(img_solution, str(solved_board_as_list[j]), (
                    cells[i].x, cells[i].y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1, cv2.LINE_AA)

    # show solution
    cv2.imshow('img solved', img_solution)
    cv2.waitKey(0)

    return np.array(board)

if __name__ == "__main__":
    main()
```

This code uses OpenCV & Tesseract OCR to process and recognize digits in an image of a Sudoku board

What’s next?

Next, I would like to work on taking a live image from webcam or phone camera. Then, I would take my program and convert it to an app on phone.

THANK YOUS

Thank you to my IMPp mentor Mr. Thom

Also thank you to my CS teacher Mr. David L.

The End

:)