Sudoku Solver

Sai Keerthi Doma

Department of Computer Science

Bridgewater State University

COMP 545 -001 - Analysis of Algorithms

Dr. Haleh Khojasteh

December 23, 2022

Table of Contents

Introduction

Sudoku is a logic-based number-placement puzzle. The term "sudoku" is abbreviated from the Japanese, where su means number and doku means single. It consists of a 9x9 grid(81 total cells) with nine 3x3 subgrids. The goal is to fill the empty cells in the grid in such a way that each number from 1 to 9 appears only once in each row, column, and 3x3 subgrid.

Sudoku puzzles are currently growing more and more well-liked among people all around the world. As the game has grown in popularity throughout many nations, numerous developers have worked to create puzzles that are both more challenging and interesting. Nowadays, the game may be found in almost every newspaper, book, and online.

Rules:

Solve the sudoku in such a way that it satisfies all of the following rules:

1. In every row, the numbers 1 to 9 must appear exactly once.
2. In every column, the numbers 1 to 9 must appear exactly once.
3. In each of the grid's 3x3 sub-boxes, the numbers 1 to 9 must appear exactly once.

Project Approach

In this project, I implemented python code to generate sudoku based on the user-selected level of difficulty, such as easy or hard, and I also implemented python code to solve the generated sudoku by following to the 3 main constraints, such as each row, col, and 3x3 subgrid should have 1-9 exactly once.

I have used the Pygame module for user interface and Created a window with a box of 10 Horizontal & 10 Vertical lines. Next, it will ask the user to choose the difficulty level,after choosing the difficulty level, a sudoku is generated. Then it solve the generated sudoku using Backtracking algorithm.

For solving sudoku we have used Backtracking Algorithm. Backtracking is an algorithm whose goal is to use brute force to find the desired solution to a problem. where we start with one possible value out of many possibilities and try to solve the problem, if we are able to acieve the goal then print solution else we will backtrack and select some other possible value and try to solve it until reach the goal.So, the recursion is the key in Backtracking.

Algorithm

❖ For Sudoku Generator:

Generating Sudoku randomly based on the chosen level of difficulty. When the user chooses the easy level(Press ”e”), create a sudoku with 10 numbers in different locations. If the user chose the hard level(Press “h”), it will generate a sudoku with 20 numbers in different locations.

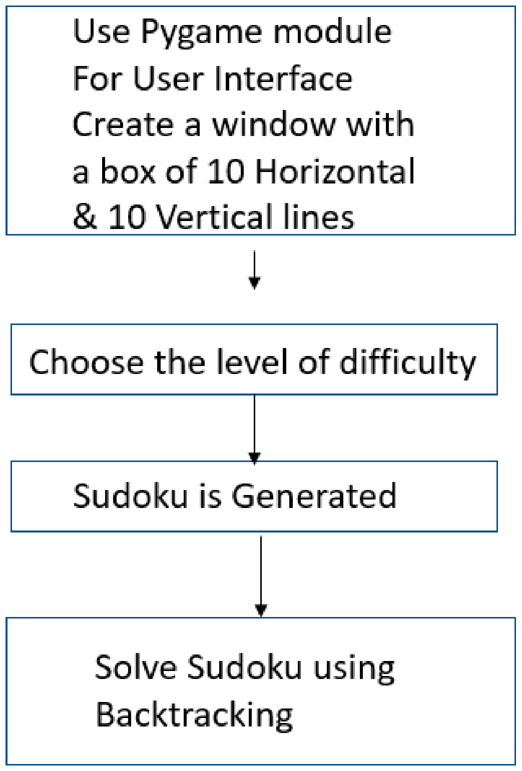
❖ For Sudoku Solver:

Find row, column of an empty cell.

For numbers from 1 to 9:

1. If there is no conflict for number at row, column and 3x3 sub-grid, then assign number to that cell
2. Recursively try fill all the empty cells
3. If recursion successful, return true
4. Else, remove the number and try with other possible number

Flow Chart


Use Pygame module
For User Interface
Create a window with
a box of 10 Horizontal
& 10 Vertical lines
Choose the level of difficulty
Sudoku is Generated
Solve Sudoku using
Backtracking

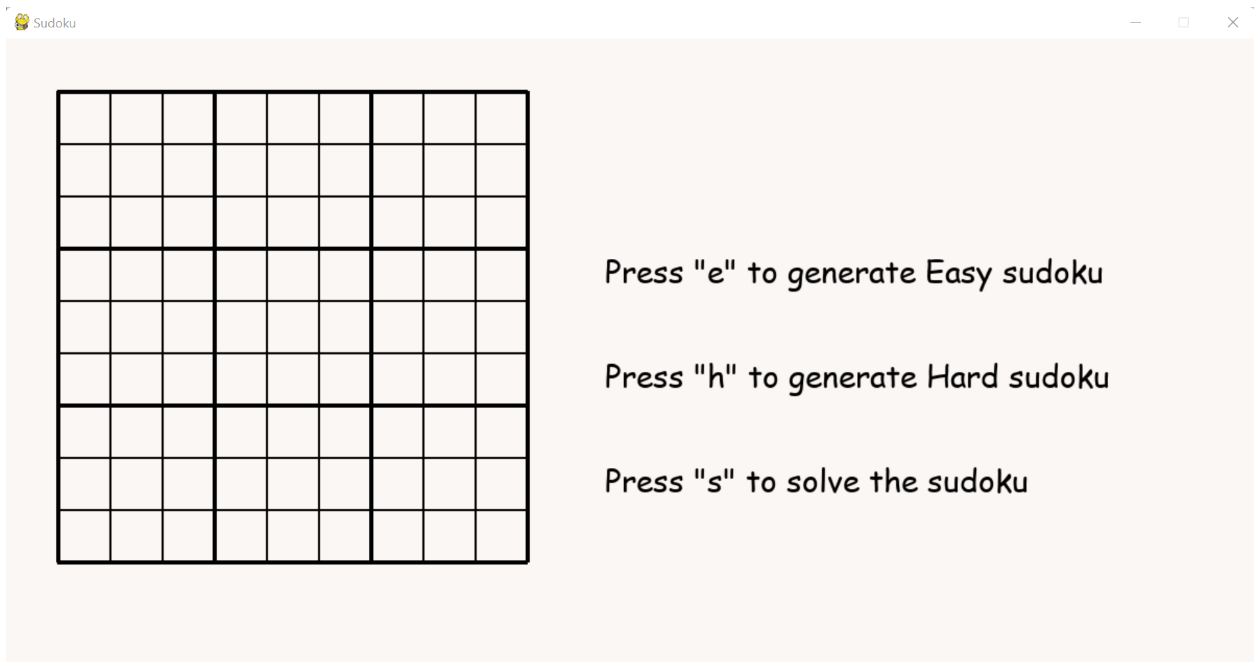
Code

```
import pygame

import random


clock = pygame.time.Clock()

display_width = 1200

display_height = 600


background_color = (251,247,245)

original_grid_element_color = (52, 31, 151)

buffer = 5


grid = [[0 for x in range(9)] for y in range(9)]


def isEmpty(num):

    if num == 0:

        return True

    return False


def isValid(row,col, num):

     #Check for Column, row and sub-grid

     #Checking row

    for i in range(0, len(grid[0])):
        if(grid[row][i] == num):
            return False
    #Checking column
    for i in range(0, len(grid[0])):
        if(grid[i][col] == num):
            return False
    #Check sub-grid
    x = row//3*3
    y = col//3*3
    #Gives us the box number
    for i in range(0,3):
        for j in range(0,3):
            if(grid[x+i][y+j]== num):
                return False
    return True


def Sudoku_Generator(n):          # Generator
    global grid
        # The range here is the amount
        # of numbers in the grid
    for i in range(n):
        #choose random numbers
        row = random.randrange(9)
        col = random.randrange(9)
        num = random.randrange(1,10)
        while(not isValid(row,col,num) or grid[row][col] != 0): #if taken
or not valid reroll
```

```
            row = random.randrange(9)
             col = random.randrange(9)
             num = random.randrange(1,10)
        grid[row][col]= num;
solved = 0


def sudoku_solver(win):
    global grid
    myfont = pygame.font.SysFont('Comic Sans MS', 35)
    for i in range(0,len(grid[0])):
        for j in range(0, len(grid[0])):
            if(isEmpty(grid[i][j])):
                for k in range(1,10):
                    if isValid(i,j, k):
                        grid[i][j] = k
                        pygame.draw.rect(win, background_color, ((j+1)*50
+ buffer, (i+1)*50+ buffer,50 -2*buffer , 50 - 2*buffer))
                        value = myfont.render(str(k), True, (0,0,0))
                        win.blit(value, ((j+1)*50 +15,(i+1)*50))
                        pygame.display.update()
                        pygame.time.delay(25)
                        sudoku_solver(win)
                       #Exit condition
                        global solved
                        if(solved == 1):
                            return 1
                        #if sudoku_solver returns, there's a mismatch
                        grid[i][j] = 0
                    pygame.draw.rect(win, background_color, ((j+1)*50
+ buffer, (i+1)*50+ buffer,50 -2*buffer , 50 - 2*buffer))

                    pygame.display.update()

                    return

    solved = 1



def main():

    pygame.init()                                   #initializing the
pygame

    win = pygame.display.set_mode((display_width , display_height))
#creating a window

    pygame.display.set_caption("Sudoku")            #setting a caption as
sudoku

    win.fill(background_color)                      #fill the background
color

    myfont = pygame.font.SysFont('Comic Sans MS', 30)

    myfont1 = pygame.font.SysFont('Comic Sans MS', 40, (255,0,0))

   for i in range(0,10):

        if(i%3 == 0):   #for bold subboxes

            #(color,starting coordinate,ending coordinate,line thickness)

            pygame.draw.line(win, (0,0,0), (50 + 50*i, 50), (50 + 50*i
,500 ), 4 )

            pygame.draw.line(win, (0,0,0), (50, 50 + 50*i), (500, 50 +
50*i), 4 )



        pygame.draw.line(win, (0,0,0), (50 + 50*i, 50), (50 + 50*i ,500 ),
2 )  #vertical lines

        pygame.draw.line(win, (0,0,0), (50, 50 + 50*i), (500, 50 + 50*i),
2 )   #horizontal lines

    pygame.display.update()
    e= myfont.render('Press "e" to generate Easy sudoku' , True , (0,0,0))

    h= myfont.render('Press "h" to generate Hard sudoku' , True , (0,0,0))

    s= myfont.render('Press "s" to solve the sudoku' , True , (0,0,0))


    win.blit(e, (575,200))

    win.blit(h, (575,300))

    win.blit(s, (575,400))

    pygame.display.update()


    flag=False

    while not flag:

        for event in pygame.event.get():        #If we press the quit in
window the window will quit

            if event.type == pygame.QUIT:

                flag=True

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K_e:

                    Sudoku_Generator(20)

                    for i in range(0, len(grid[0])):

                        for j in range(0, len(grid[0])):

                            if(0<grid[i][j]<10):

                                value = myfont.render(str(grid[i][j]),
True, original_grid_element_color)

                                win.blit(value, ((j+1)*50 + 15, (i+1)*50
))

                                pygame.display.update()


                 if event.key == pygame.K_h:

                    Sudoku_Generator(10)
                    for i in range(0, len(grid[0])):
                        for j in range(0, len(grid[0])):
                            if(0<grid[i][j]<10):
                                value = myfont.render(str(grid[i][j]), 
True, original_grid_element_color)
                                win.blit(value, ((j+1)*50 + 15, (i+1)*50 
))
                                pygame.display.update()
                if event.key == pygame.K_s:
                    if(sudoku_solver(win)==1):
                        c= myfont1.render('Solved Successfully!!!' , True 
, (0,0,0))
                        win.blit(c, (520,20))
                        pygame.display.update()


main()
pygame.quit()
quit()
```

Result

Pygame window with the caption "Sudoku" and a grid with 10 horizontal and 10 vertical lines, as well as some text instructions.

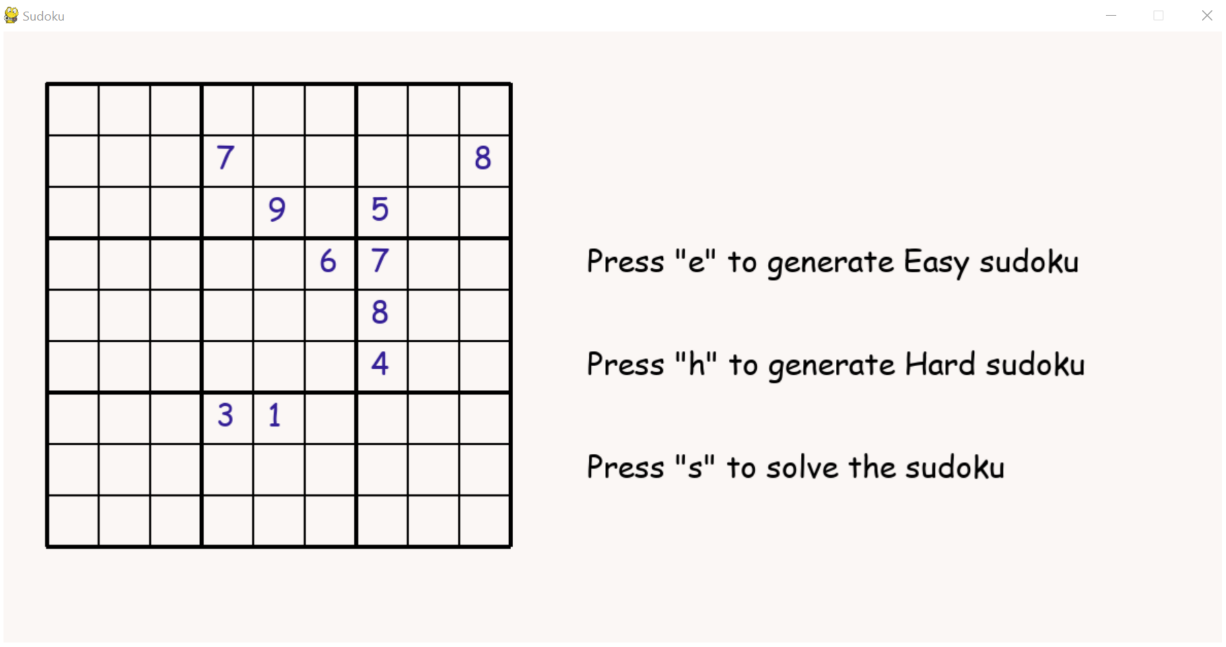


Press "e" to generate Easy sudoku

Press "h" to generate Hard sudoku

Press "s" to solve the sudoku

Sudoku Generated with 10 random numbers in different locations when we press "h"


Sudoku

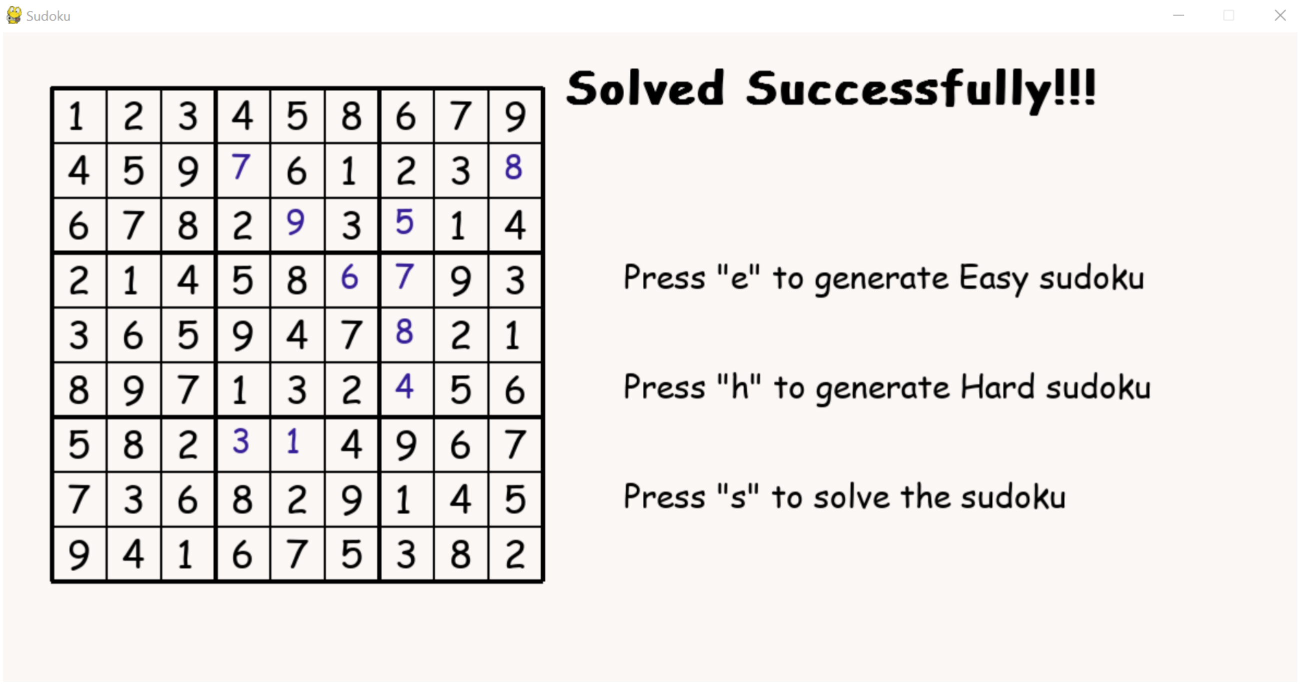
			7					8
				9		5		
					6	7		
						8		
						4		
			3	1				

Press "e" to generate Easy sudoku

Press "h" to generate Hard sudoku

Press "s" to solve the sudoku

Result after completion of Sudoku


Sudoku

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

Solved Successfully!!!

Press "e" to generate Easy sudoku

Press "h" to generate Hard sudoku

Press "s" to solve the sudoku

Bibliography

[1]. https://www.geeksforgeeks.org/sudoku-backtracking-7/

[2]. https://www.geeksforgeeks.org/building-and-visualizing-sudoku-game-using-pygame/

[3]. https://www.geeksforgeeks.org/program-sudoku-generator/

[4]. J. F. Crook(2009), A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles https://www.ams.org/notices/200904/tx090400460p.pdf

[5]. Michael Mepham (2005), Solving Sudoku, Crosswords Ltd., Frome, England. http://www. sudoku.org.uk/PDF/Solving_Sudoku.pdf.