Sudoku Solver

Team Number: 34
Team Members:
Andy Corliss
Max Corliss
Phillip Ionkov
Ming-Yuan Lo
Team Mentor:
Li-Ta Lo
Latchesar Ionkov

Problem Definition:
Sudoku is a logic-based puzzle that is solved by placing numbers in a grid. For each row, column, and box, all numbers in a range must be placed only once. For example, an original 9x9 grid (the most common form, found in newspapers and books) has 9 rows, 9 columns and 9 boxes, and the numbers 1 to 9 are used. Each box is a square root of the total board size (for example, a 9x9 grid has nine 3x3 boxes, and a 16x16 grid has sixteen 4x4 boxes). We want to be able to make the computer solve Sudoku puzzles of any size, especially puzzles that humans can not solve easily. We will also investigate the relationship between board size and computer's speed in solving the puzzle.

Problem Solution:
We will solve this problem by creating a program in Python[1] (using the repl.it[2] web-based programming environment) that will systematically try different number combinations using a technique called "backtracking"[3, 4]. In this technique, the computer tries a potential partial solution and checks to see if it works. If the partial solution works, the computer will try to add another number to the solution. If the solution does not work, it will go back and try an alternative potential solution. Recursion[4] is a computing technique in which a function repeatedly calls itself, and is used to implement backtracking.

Our Progress:
We began this year by learning some very basic Python coding. We made simple programs such as drawing graphics in Python with the Turtle library. Because Python is a new coding language for most of us, we needed to get a grasp on Python's syntax. We have been looking into recursion through drawing Sierpinski's Triangle[5] and other fractals. More recently, we started working on some more relevant codes, such as the "N Queen Puzzle"[6] where the solver attempts to place 'n' queens on a chessboard of 'n' x 'n' size without any of the queens threatening each

other. To solve the N Queen problem, we used backtracking. Both of these methods will be used for our Sudoku solver.

What we expect:

We expect to make a Sudoku solver that can solve any Sudoku puzzle. We will start with relatively simple puzzles, such as 4x4 and 9x9 Sudokus with increasing difficulty. Then, we will continue to the harder and more complicated puzzles, including 16x16, 25x25, and greater. Additionally, we expect our solver to generate Sudoku puzzles and determine if a puzzle has one or multiple solutions.

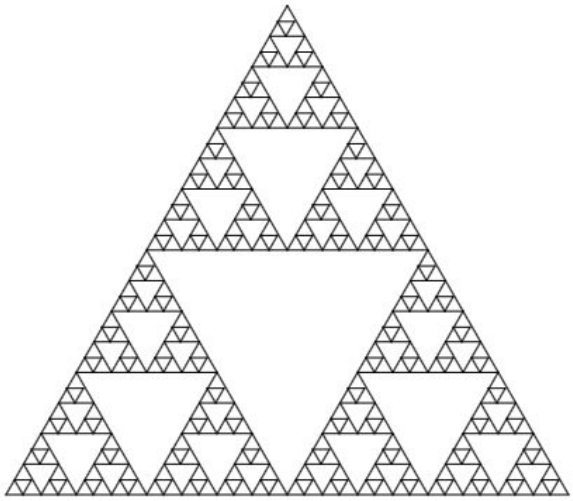
Citations:

1. The Python programming language, https://www.python.org/
2. The repl.it web-based python programming environment, https://repl.it/
3. Wikipedia page on algorithms for solving Sudoku, https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
4. Lecture note from Stanford on backtracking and recursion, the algorithm to solve some "searching" problems https://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf
5. Sierpinski triangle, the motivating exercise for us to understand recursion better, https://en.wikipedia.org/wiki/Sierpinski_triangle
6. NQueen problem, where we actually put "backtracking" into solving a puzzle. https://en.wikipedia.org/wiki/Eight_queens_puzzle

Appendix:

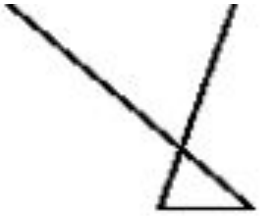
Sierpinski Triangle Code by Phillip Ionkov:

```
import turtle
import math

t = turtle.Turtle()
t.speed(0)

def draw_triangle(size, x,
y):
  t.penup()
  t.goto(x, y)
  t.pendown()
  for i in range(3):
    t.forward(size)
    t.left(120)

def draw_spks(size, x, y,
level):
```



```
  if (level >= 0):
    draw_triangle(size/2, x + size/2, y)
    draw_spks(size/2, x, y, level - 1)
    draw_spks(size/2, x + size/2, y, level - 1)
    draw_spks(size/2, x + size/4, y + size*math.sqrt(3)/4,
level - 1)

draw_triangle(400, -200, -200)
t.left(60)
draw_spks(400, -200, -200, 4)
t.ht()
```

Triangle Generator by Max Corliss



```
import turtle

t = turtle.Turtle()

t.speed(3)

def chill(s,l,n):
  for i in range(n):
      t.forward(2*s)
      t.left(l)

def tri(a,b):
  t.left(180 - a)
  t.forward(60)
  t.left(180 - b)
  t.forward(60)
  t.goto(0,0)

def test(s):
  t.right(s)
  t.left(s)

def triangle(a,b):
  if(a > b):
      t.forward(30)
      t.goto(0,0)
      t.left(a)
      t.forward(500)
      t.penup()
      t.goto(30,0)
      t.pendown()
      t.left(180 - a)
      t.right(b)
```

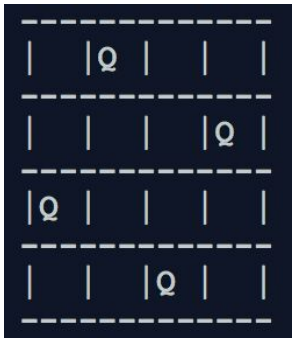
```
        t.forward(500)
    else:
        t.forward(30)
        t.goto(0,0)
        t.left(b)
        t.forward(500)
        t.penup()
        t.goto(30,0)
        t.pendown()
        t.left(180 - b)
        t.right(a)
        t.forward(500)

triangle(70,40)
```

N Queen Solver By Ming-Yuan Lo:

```
N = 4

board = [[' ' for i in range(N)] for i in range(N)]

def printboard():
  for i in board:
    print('-' * (N * 3 + 1))
    print('|', end="")
    for j in i:
      print(j, '|', end="")
    print("")
  print('-' * (N * 3 + 1))


def notOnRow(row):
  for i in range(N):
    if board[row][i] == "Q":
      return False
  return True

def notOnCol(col):
  for i in range(N):
    if board[i][col] == "Q":
      return False
  return True
```

```
def notOnDiagonal(row, col):
  for i in range(N):
    for j in range(N):
      if board[i][j] == 'Q' and abs(row-i) == abs(col-j):
        return False
  return True

def isSafe(row, col):
  if notOnRow(row) and notOnCol(col) and
notOnDiagonal(row, col):
    return True
  return False

def NQueen(row):
  if row == N:
    return True
  for col in range(N):
    if isSafe(row, col):
      board[row][col] = "Q"
      if NQueen(row + 1):
        return True
      else:
        board[row][col] = " "
  return False

NQueen(0)
printboard()
```

Output:



Square in a Square by Andy Corliss:

```
import turtle
import math
```

```
t = turtle.Turtle()
t.speed(0)
def square(size):
  for i in range(4):
    t.forward(size)
    t.left(90)

def sqrthing(size,reps):
  t.penup()
  t.goto(-200,-200)
  t.pendown()
  for i in range(reps):
    square(size)
    t.forward(size/2)
    t.left(45)
    size = math.sqrt(2)*size/2
sqrthing(400, 15)
t.ht()
```