Bryan Rodriguez-Andrade

CS 325 Fall 2020

Portfolio Project

Project Description:

Puzzle: Sudoku

Rules: the Sudoku board is a 9x9 multi-dimensional matrix that is partially filled in with numbers 0-9. Empty boxes are considered boxes that have the number 0 filled in. The board is broken up into smaller 3x3 sub-sections and the goal of the puzzle is to have only 1 instance of the integer 1-9 per 3x3 box, and only 1 instance of the integer 1-9 per row and column of the 9x9 board, respectively. In other words, you cannot repeat any integers for the entire board on a given row and column and 3x3 sub-section. This game provides randomly generated boards, a verification algorithm for your solution, a GUI to play the puzzle, and a backtracking algorithm to solve the puzzle for you.

Verification Algorithm:

```
def verify_solution(sudoku_board):
    rows = [row for row in sudoku_board if not verify_row(row)]
    board = list(zip(*sudoku_board))
    for i in board:
        if 0 in i:
            return False
    columns = [column for column in sudoku_board if not verify_row(column)]
    boxes = []
    for i in range(1, 9, 3):
        for j in range(1, 9, 3):
            box = list(itertools.chain(row[j:j+3]for row in board[i:i+3]))
            boxes.append(box)
    solved_boxes = [box for box in boxes if not verify_row(box)]
    return not rows or not columns or not solved_boxes
```

This brute-force, algorithm uses a double for loop to verify each row, column, and boxes. It then returns a Boolean value based on their validity. Despite a few list comprehensions within the algorithm, the double for loop portion of the algorithm persists during run time and yields an O(n^2) complexity.

This solution is NP-Complete because it runs in polynomial time, we can then reduce the problem. If we go through each of the boundaries and check solutions for each row, we can revert the puzzle to its original problem. Reducing problems shows that this Sudoku problem is in polynomial time and thus is NP-Complete.

Solution Algorithm:

The solution algorithm uses a backtracking method to recursively check for correct solutions within the board. The time complexity for this solution algorithm is O(n^2) as it must also iterate through nxn boxes to verify the correct integer to be placed through the legal_move_check() method. This solution algorithm works in conjunction with the methods from the game to check that the assigning number

within the squares are valid. If the assignments are valid it then recurses through the grid and attempts to complete the puzzle.

Random Board Generator:

The puzzle also includes a random board generator that will produce a, solvable, random Sudoku board everytime the “Reset” button is pressed within the GUI.

README and Hosting:

README & Repl.it hosting located at: https://repl.it/@BryanRodriguez5/SUHDoku

Github Repo: https://github.com/brod26/CS325SUHdoku

Sudoku

Objective

There are 81 squares, divided into 9 blocks, each containing 9 squares. To win the game, each nine block section must contain all number 1-9 within its squares, without any duplicates in any one row, column or 9 block sub-section.

Game Rules

```
-Use your mouse to click on the block you'd like to select.
-Once your block is selected, you may type using the numpad, any number from 1-9.
-Your initial number will be "sketched" in, meaning it will not be registered until you hit the "ENTER" button. Doing so will
-You may sketch as many numbers as you like, however, you'll need to go back and manually enter every number once you're read
-The "Solve" button will use a backtracking, recursive algorithm to solve the board for you.
-The "Verify" button uses a brute-force, algorithm to verify your board solution, or the solution provided from the "Solve" b
-The "Reset" button generates a random new board.
-Once you've filled in all squares, you may hit the "Verify" button which will then verify your solution, if the solution is
```

How to play

Repl.it

The game is hosted on repl.it here: https://repl.it/@BryanRodriguez5/SUHDoku

Terminal

The alternative approach has a better UI experience but requires some set-up:

1. Install the latest version of Python
2. Install pip and Pygame

   To install pygame, type: python -m pip install pygame in your terminal
3. On GitHub, click the Code button, download the zip file to the desktop and in your termial:

   -cd Desktop

   then type:

   -cd CS325SUHdoku-main
4. In your terminal type python3 sudoku_GUI.py

Thanks for playing!