Documentation for the Sudoku Project

Created by Shashwat Saini, Dayananda Sagar University

1. Running the program:

To run the version of the code written in C, place all the associated files: game.py, generator.py, main.py, menu.py, & save.txt in the same folder, as they should be by default: To run the program, run menu.py using a suitable interpreter. *Note: The following libraries must be installed: pygame.*

2. The Python Program

2.1: Files:

2.1.1: *game.p*y: Contains functions for the sudoku puzzle, to check for win, check for fit, etc.

2.1.2: *generator.py:* Contains a backtracking algorithm to generate a sudoku puzzle, based on the difficulty level.

2.1.3: *main*.py: Contains the loop in *pygame* to handle graphics of the game, note-taking, etc.

2.1.4: *menu.py:* Contains the intro menu, to select a level, or to load the previously played saved game.

2.1.5: *data.py:* Contains the data handling code, to access high score, etc.

2.2: The Backtracking Algorithm:

This forms the backbone of the program. A Backtracking algorithm approaches a problem recursively to try and solve it. It builds a solution step by step, continuously eliminating those scenarios which fail.

The first step is to generate a solved sudoku board. This can be done by moving systematically from cell to cell, generating a random number using *random.randint(1,9).* If the condition is satisfied, the program enters the number onto the board and moves ahead.

The second step is to delete cells. If the chosen level by the user is 1, the program deletes 30 cells, if 2, then 40, and if 3, then 50. Obviously, the puzzle becomes tougher with fewer cells being available for the user in the puzzle for solving. Deleting more than 55 cells tends to take a lot more time, and has been found to not be feasible.

2.3: Pygame utilization:

2.3.1: In *menu.py:* A button class is created to enable clickable buttons in the program. The class has three different modes, normal, hover, and pressed. Different colours are used to indicate different states. Four buttons are created using this class, three for the different levels, and one for the ‘load’ option, to load the previously played game. Once a button has been clicked, the file calls the *gameloop()* function in *main.py.*

2.3.2: In *main.py:* This file is, of course, the main part of the entire program, as the name goes. We shall discuss the functions one by one.

2.3.2.1: *getboard():* This function gets the game board, either from the generator or the saved file.

2.3.2.2: *drawbg():* This function draws the board itself, with a white rectangle for the game, and lines to differentiate different boxes in the game.

2.3.2.3: *print\_notes():* This function prints the notes entered in the board so far, in each cell. Notes here refer to the scribbling of all the possible values a cell can take, and this technique is much in use in harder sudoku puzzles.

2.3.2.4: *addnum(num, col, row):* This function prints the values that have been entered into the puzzle board so far.

2.3.2.5: *solve(num, col, row):* This function checks if the entered value is compatible with the cell, in accordance with the *check\_fit()* function of the *game.py* class. It also takes care of removing the notes from the cell, once a suitable number has been entered, by blitting a white rectangle to ‘reset’ the box.

2.3.2.6: *addnotes(num, row, col):* This function adds the notes that the user enters to the notes.

2.3.2.7: *cursor(col, row, color):* This function controls the cursor, essentially a square that goes around the board to indicate the current square the user is solving. *color* here will tell the cursor to be green by default, otherwise red, if the user has entered a wrong value.

2.3.2.8: *mode\_solves():* This function sets the ‘mode’ of the puzzle to ‘solve,’ in which the user is entering the necessary values to solve the puzzle.

2.3.2.9: *mode\_notes():* This function sets the ‘mode’ of the puzzle to ‘notes,’ in which the user is entering tiny notes into each cell, in order to solve tougher puzzles.

2.3.2.10: *gameloop():* This function coordinates with all other functions, calling them when needed. It contains the *while* loop, that pygame needs. It sets the icon and the title of the pygame window, and has provisions to check for key clicks.

2.4: *game.py*

This file contains functions similar to the functions in *sudoku.c*, that check for fit, for a given value, check for a win, etc.

3. Credits:

All icons are from *flaticon.com.*