UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

COMPUTING FOR LIFE
COP3502

Final Project: Sudoku

Overview

In this project, **students will work in groups of 4** to create a simplified version of the Sudoku game. Students will implement random board generation, state checking, and visuals using PyGame. The project is designed to be a playful and practical opportunity to practice the programming skills we have learned this semester, as well as working with external libraries.

Getting Started

The provided code and instructions for working with Replit are available at the following GitHub link:

https://github.com/zhoulisha/Sudoku-Project

To get started, it is recommended for students to fork the repository (this can be done by clicking "fork" in the upper-right corner).

Rules of the Road

Note: The phrase 'single-digit numbers' will be used throughout this project to refer to numbers 1-9, inclusive. For the purposes of this project, we will not consider 0 a single-digit number.

A Sudoku is a puzzle consisting of a 9x9 grid which is partially filled with single-digit numbers. The objective is to fill in the rest of the grid with single-digit numbers so that certain rules are satisfied:

1. In each of the nine rows of the board, no digit is repeated (and each digit occurs once).
2. In each of the nine columns of the board, no digit is repeated (and each digit occurs once).
3. In each of the nine 3x3 outlined 'boxes' of the board, no digit is repeated (and each digit occurs once).

Example Sudoku boards. When the board satisfies the above rules and each cell is filled, it is solved.

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

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

Image reference: https://sandiway.arizona.edu/sudoku/examples.html

Your task for this project is split into two main parts:

1. Generate a random solvable Sudoku board.
2. Create visuals for your program using PyGame.

Generating a Sudoku board involves a more complex algorithm than we can cover in this class, so some of this code is provided to you.

UI Requirements

Many of the visuals in this project do not have a hard requirement. We will mainly be checking that your visuals satisfy the requirements laid out below.

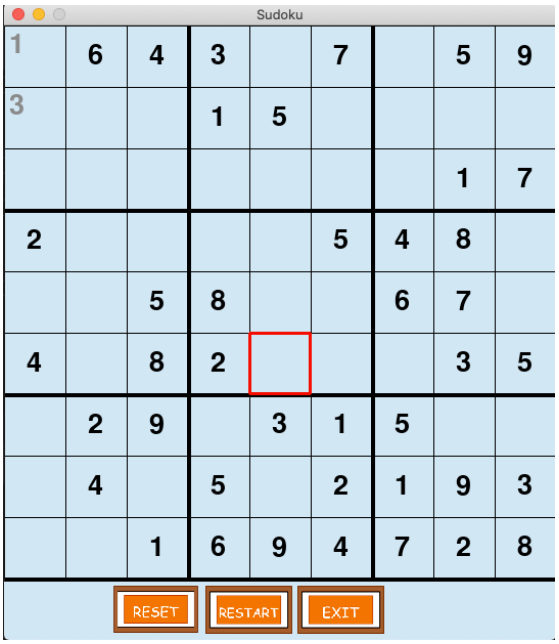
When the program starts, it should display a Game Start screen. This will have buttons for the user to choose a difficulty between easy, medium, or hard.

Once the user clicks a button to select a difficulty, you will generate a Sudoku board (more details on how to do this later). The difficulty level will determine how many squares on the board are initially empty:

Difficulty	Number of empty cells
easy	30
medium	40
hard	50

Once a difficulty is chosen, your program should generate a Sudoku board, make the appropriate number of cells empty, and display the partially filled board as shown in the image below.

Game Start screen



Game-In-Progress Screen

Note: 1 and 3 in the top-left squares are sketched in by a user as input to the game.

Sudoku successfully completed screen

Sudoku unsuccessfully completed screen

Once the player has chosen a difficulty, a partially filled Sudoku board is shown. There are a few different operations the user can use to interact with the game:

- The user can select a cell by clicking on it. Please note that the user can select any cell; however, the user can only edit the cells that are not randomly generated.

- Once a cell is selected, the user can type a single-digit number to 'sketch' it in the cell (see the game-in-progress screenshot for an example).
- If the selected cell has a sketched value, the user can press the return (enter) key to submit their guess.
- The user can select a different cell by clicking on it, or by using the arrow keys.

If the board is full and solved completely correctly, the game win screen is displayed. If the board is full but not solved correctly, the game over screen should be displayed.

During the game, there are three buttons below the board:

- The Reset button will reset the board to its initial state.
- The Restart button will take the user back to the Game Start screen.
- The Exit button will end the program.

Class Structure

Students should have the following classes in their code. Class methods/attributes other than those listed may be added as necessary.

SudokuGenerator (Required)

This class generates a Sudoku – the puzzle as well as the solution.

`__init__(self, row_length, removed_cells)`

Constructor for the SudokuGenerator class.
For the purposes of this project, row_length is always 9.
removed_cells could vary depending on the difficulty level chosen (see "UI Requirements").

`get_board(self)`

Returns a 2D python list of numbers, which represents the board

`print_board(self)`

Displays the board to the console.
This is not strictly required, but it may be useful for debugging purposes.

`valid_in_row(self, row, num)`

Returns a Boolean value.
Determines if num is contained in the given row of the board.

`valid_in_col(self, col, num)`

Returns a Boolean value.
Determines if num is contained in the given column of the board.

`valid_in_box(self, row_start, col_start, num)`

Returns a Boolean value.
Determines if num is contained in the 3x3 box from (row_start, col_start) to (row_start+2, col_start+2)

`is_valid(self, row, col, num)`

Returns if it is valid to enter num at (row, col) in the board.
This is done by checking the appropriate row, column, and box.

`fill_box(self, row_start, col_start)`

Randomly fills in values in the 3x3 box from (row_start, col_start) to (row_start+2, col_start+2)
Uses unused_in_box to ensure no value occurs in the box more than once.

`fill_diagonal(self)`

Fills the three boxes along the main diagonal of the board.
This is the first major step in generating a Sudoku.
See the Step 1 picture in Sudoku Generation for further explanation.

`fill_remaining(self, row, col)`

This method is provided for students.
It is the second major step in generating a Sudoku.
This will return a completely filled board (the Sudoku solution).

`fill_values(self)`

This method is provided for students.
It constructs a solution by calling fill_diagonal and fill_remaining.

`remove_cells(self)`

This method removes the appropriate number of cells from the board.
It does so by randomly generating (row, col) coordinates of the board and setting the value to 0.
Note: Be careful not to remove the same cell multiple times. A cell can only be removed once.
This method should be called after generating the Sudoku solution.

`generate_sudoku(size, removed)`

Note: This is a function outside of the SudokuGenerator class.
This function should also be implemented in sudoku_generator.py as it interacts with the class.
Given size and removed, this function generates and returns a size-by-size sudoku board.
The board has cleared removed number of cells.
This function should just call the constructor and appropriate methods from the SudokuGenerator class.

Cell (Recommended)

This class represents a single cell in the Sudoku board. There are 81 Cells in a Board.

`__init__(self, value, row, col, screen)`

Constructor for the Cell class

`set_cell_value(self, value)`

Setter for this cell's value

`set_sketched_value(self, value)`

Setter for this cell's sketched value

`draw(self)`

Draws this cell, along with the value inside it.
If this cell has a nonzero value, that value is displayed.
Otherwise, no value is displayed in the cell.
The cell is outlined red if it is currently selected.

Board (Recommended)

This class represents an entire Sudoku board. A Board object has 81 Cell objects.

`__init__(self, width, height, screen, difficulty)`

Constructor for the Board class.
screen is a window from PyGame.
difficulty is a variable to indicate if the user chose easy, medium, or hard.

`draw(self)`

Draws an outline of the Sudoku grid, with bold lines to delineate the 3x3 boxes.
Draws every cell on this board.

`select(self, row, col)`

Marks the cell at (row, col) in the board as the current selected cell.
Once a cell has been selected, the user can edit its value or sketched value.

`click(self, x, y)`

If a tuple of (x, y) coordinates is within the displayed board, this function returns a tuple of the (row, col) of the cell which was clicked. Otherwise, this function returns None.

`clear(self)`

Clears the value cell. Note that the user can only remove the cell values and sketched value that are filled by themselves.

`sketch(self, value)`

Sets the sketched value of the current selected cell equal to user entered value.
It will be displayed at the top left corner of the cell using the draw() function.

`place_number(self, value)`

Sets the value of the current selected cell equal to user entered value.
Called when the user presses the Enter key.

`reset_to_original(self)`

Reset all cells in the board to their original values (0 if cleared, otherwise the corresponding digit).

`is_full(self)`

Returns a Boolean value indicating whether the board is full or not.

`update_board(self)`

Updates the underlying 2D board with the values in all cells.

`find_empty(self)`

Finds an empty cell and returns its row and col as a tuple (x, y).

`check_board(self)`

Check whether the Sudoku board is solved correctly.

Main (Required)

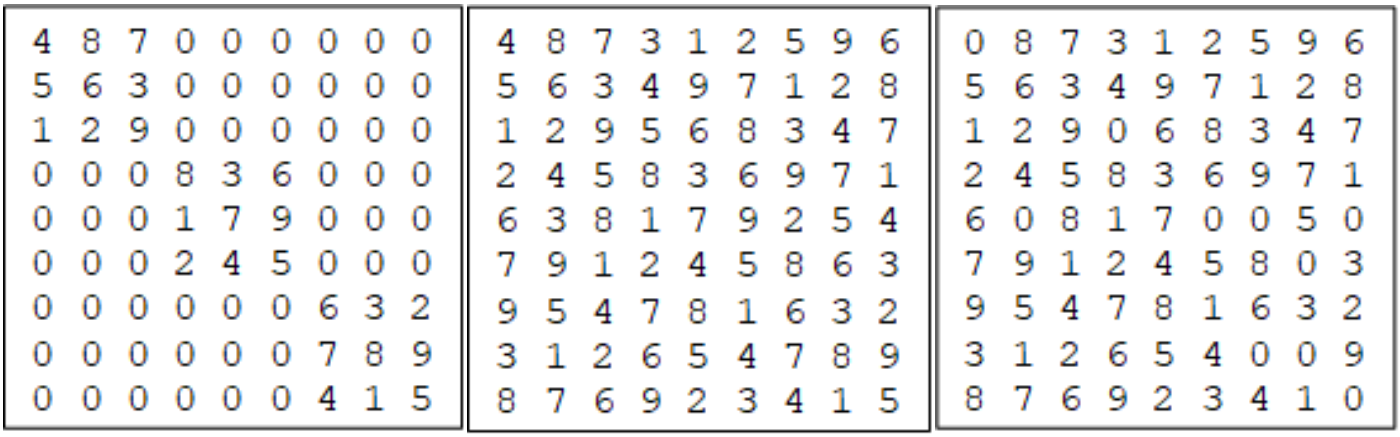
In addition to the above classes, students will have a sudoku.py file, where the main function will be run. This file will contain code to create the different screens of the project (game start, game over, and game in progress), and will form a cohesive project together with the rest of the code.

Sudoku Generation

Generating a Sudoku board involves a process known as backtracking, which is beyond the scope of this class. Students will implement some of the functions necessary to generate a Sudoku board, and the rest of the backtracking algorithm will be provided.

The overall process of generating a Sudoku board follows 3 main steps:

1. Generate values along boxes on the diagonal
2. Fill in the remaining values using backtracking
3. Make some of the cells blank



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

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

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

A Sudoku board after steps 1, 2, and 3 of generating the board. '0' digits represent empty spaces.

Notice that the board is filled after step 2 is completed. This is the solution to the Sudoku, so it should be stored in your program to verify if the solver's guess is correct or incorrect for a certain cell.

Deliverables

NOTE: Your output for this project may differ from the example. The outline suggests the minimum requirement and feel free to extend other functionality if you have time. Ensure you satisfy all the requirements in this document and on the rubric on Canvas in order to receive full credit.

The final project carries 10% weight of your final grade, and the deliverables have two components:

1. Group Deliverable (9% of grade) - One per group
2. Peer Feedback (1% of grade) - One for each team member (including yourself)

You must submit all deliverables. If you individually completed the project, you still must finish all the above

Group Deliverable (9% of your grade)

All submissions for this project should be submitted by any one member per group unless specified otherwise. You are required to submit three deliverables per group.

1. Report [2%]

- A maximum 3-page PDF document covering the following:
  - Administrative
    - Team Name
    - Team Members
    - GitHub URL (See below)
    - Link to Video (See below)
    - The name of the person who submits sudoku_generator.py on Canvas.
  - Reflection [Suggested 1-1.5 Page]
    - Distribution of Responsibility and Roles: Who did what?
    - As a group, how was the overall experience for the project?
    - Did you have any challenges? If so, describe.
    - If you were to start once again as a group, any changes you would make to the project and/or workflow?
    - Comment on what each of the members learned through this process.
  - References
- The font size must be at least 11 points.
- The report file must be named **Report_Group<*insert_group_number*>.pdf, e.g., Report_Group11.pdf**.

2. Source Code [5%]

- You will submit sudoku_generator.py file on Canvas per group. The name of the person who submits sudoku_generator.py on Canvas should be listed in the report [1.5%]
- You will be graded on the overall code quality for this deliverable. [3.5]
  - The group must include a link to the **PRIVATE** GitHub Repository where you collaborated in the report. A TA will be assigned to be added as a collaborator to evaluate your source code.

3. Video [2%]

- You will be submitting a URL to a YouTube video in the Report (feel free to make it Public or Unlisted after group discussion).
- The video will be used to evaluate the functionalities of your project. Thus, make sure you cover all features required in the video.
- The maximum time limit for the video is **5 minutes** and it will be ***strictly enforced.*** You will lose 20% points for each additional minute you use.
- You have the option to record it as a group or present the project in the video by a single member. We are flexible here and I would recommend you use Loom or Zoom to record the video.

Note: Although this is a group project and all members will have the same grade for the individual deliverables in most cases, the instructor reserves the right to accommodate individual grades on the Final Project based on feedback by your team members. If your peers' average score for you was less than 70%, students on average received a 10-50% penalty on their individual score.