INF-1400: Objektorientert programmering

ASSIGNMENT 1

Tora Kjetså Homme

February 1, 2024

Table of contents

1 Introduction 1
2 Technical background 2
2.1 Classes 2
2.1.1 Init 2
2.1.2 Subclass and superclass 2
2.2 Append 2
2.3 Remove 2
2.4 2D list 2
3 Implementation 2
3.1 Sudoku board 3
3.2 Elements 4
3.3 Squares 4
4 Testing 5
5 Discussion 5
5.1 Results 5
5.2 Strengths 5
5.3 Improvements 5
6 Conclusion 6
7 Acknowledgement 6
8 References 6

1 Introduction

This report will describe the design of a program which solves Sudoku puzzles, using classes. The report will look at the technical background and the implementation of the classes and the solving algorithm. It will also have a short look at some testing results. Lastly, the report contains a discussion and a conclusion at the end.

2 Technical background

2.1 Classes

A class in python is a template for different objects. All the information of that object is located in the class. This information can be attributes (variables) or different methods (functions). The attributes and methods can be accessed inside the class. [1]

2.1.1 Init

__init__ is a method used inside a class in python which is used to initialize that class. It needs a self-parameter, but it can also take other parameters. The method initializes the attributes, and it can configure an object with values which are specific for that object. [2]

2.1.2 Subclass and superclass

Since a class is like a template, one can create several objects using the same class. These objects are called subclasses of that class. The subclass is a class itself, and it inherits and can access the attributes and methods of that other class. The class which a subclass inherits from is called a superclass. In addition to have inherited attributes or methods, the subclasses can have their own as well. [3]

2.2 Append

Append is a module in python which is used to add an element to a list or array. [4]

2.3 Remove

The remove() method is used to remove the first occurrence of a defined item from a list or array. [5]

2.4 2D list

A 2D-list is simply a list of lists, where the data is organized in rows and columns. A 2D-list can be created by looping through a list inside another list. [6]

3 Implementation

In this application, there are added three classes which are used to represent and solve a game of sudoku.

3.1 Sudoku board

One of the classes is a subclass. The superclass of this subclass is used to represent a square board that can be used to play games. The subclass is used to represent a game of sudoku. Every time this class is called, it sends up the numbers already on the sudoku board to the constructor. Then, each time the class is called, it creates three different empty lists, which all contains nine different element-lists; one list of nine row-lists, one list of nine column-lists and one list of nine box-lists. These lists are used to represent the different rows, columns and boxes on the sudoku board. Then the methods to make squares and assigns them to their correct row, column and box are called.

Moreover, the class has a method which prints out the board in a sudoku board style, which makes it easy to check whether the board is correct. This method is inspired by a similar method in the superclass.

The method that makes the squares, does this by making the numbers(integers) from the sudoku file into square class objects. As will be explained more thoroughly later in the report, this square class represent a square on the board. Setting up the squares on the board is done by looping through the nine columns, row by row, and making each number from the board into an object of the square class. This way each number is made into a square object, and each square is assigned their row, column, box and value. This method makes a 2D list, where each element in the lists is a square object. And the row and column indices for the squares represents the indices in the 2D list.

Another method in this class is used to determine which rows and columns make up which box. This is done by using if statements, to easily subdivide the rows into three by three, and assign each of these segments their own three columns. Moreover, each of these three-by-three boxes get their own box index. Furthermore, the method handles any errors if they should occur.

To set up links between the squares and elements (row, column, box), there is a method that adds each square to their assigned row list, column list and box list by looping through the rows and columns.

A method which is used in the solving method, finds each of the empty squares on the board. This is done by looping through each of the rows and columns, and checking if the value is zero (which represents an empty square). If the value is zero, the position of the square is returned. If there are no empty squares left, the method returns None, which is later used in the solve method to check if the puzzle is solved. Both this method and the next are inspired by Kylie Ying's solver [7].

The last method in this class is the one that solves the sudoku boards. The solving algorithm is the brute-force method. The first thing that happens in this method is that the empty squares are found, by using the method explained earlier. If that method returns None, that means all the squares are filled in, and the sudoku must be solved. And it will return True. However, if there are empty squares, the row and column indices for these squares are then used to iterate through all the empty squares. For each empty square, the method loops through the numbers 1-9, as these are the values that can be placed on the sudoku boards.

Inside this loop, there is used different methods which will be explained more later in the report. One of these is a method to check if a number can be placed in the square. If the value is not legal to be placed there, the loop will try a different number on that square. If the number is legal in that square, that squares value is updated to be that legal number. That is done by using another method. The new value is also added to the right row, column and box lists. But before the value is updated, the value that is already in that square is removed from their row-list, column-list and box-list using a different method. The method then calls itself, and if every square has been successfully filled, the sudoku is solved.

If the sudoku is not solved, and if the loop has tried every number, and none of them works, the number that already was on the square is removed from the element-lists. The value in that square is reset to zero, and zero is added to the element-lists. Then the method goes to the square before and tries a bigger number. If none of these numbers work either, it removes the last value of the square, resets the value to zero and adds zero to the element-lists. This keeps going until every square is filled.

3.2 Elements

The second class represents the row, column or box. This is done by creating an empty list which has a type. The type is “row”, “column” or “box” and is determined each time this class is used. In the sudoku board class, these are used to create the three different lists.

The class has a method to check if a given value exists in the given list, and returns true if it does, and false if it does not. This method is used in the square class later.

The other methods this class contains are methods are one that removes and one that adds a given value from the given element-list.

3.3 Squares

The last class represents a square on the board. It keeps track of the row, column, box and value of that square.

It also has a method to print the value of the square. This is important, as it gives a readable representation of the squares.

The third method checks if a number is legal in that square, using the method from the element class to check if an element-list has that value. In other words, it checks if that specific number exists in its row, column or box already. If it does, the value is not legal, and it returns false. If the value is legal, it returns true.

The last method is a way to place the legal value in the square. This method uses the method from the element class that adds a value to its elements-lists. The square method sets the value of the square equal to the legal number. And it adds the legal number to the square’s row, column and box list.

4 Testing

The application has been tested on files with 10 sudokus, 100 sudokus and 1 million, and it can solve every sudoku in these files. When solving one million sudokus, it took 18 minutes and 52 seconds to solve them all. Also, to check if the solving algorithm worked, some specific boards have been printed.

5 Discussion

5.1 Results

The result from the testing reveals that the application works as intended, as it can solve every solvable sudoku.

5.2 Strengths

The application has several strengths. First and foremost, the application can solve the solvable sudokus, which is what was the intention of this application. Since the solving method is recursive it can go back and change a value if it does not belong at that square. And the brute force algorithm ensures that every solvable sudoku will be solved.

Another strength is that it is structured in a clear and easy way, by using classes. Because of the use of classes, each similar object and the different methods are only defined once, which makes the code compact. For example, one does not need to define 81 different squares, as one uses a square class as a blueprint for every square. In addition, the sudoku board inherits methods and attributes from its superclass, which means one does not have to define these methods and attributes more than once. And as the different operations are defined in a method, it makes it easy to do the same operation several places without having to write these operations every time.

By sending in the different class-objects as parameters for methods in other classes, one ensures that there are no circular import issues.

5.3 Improvements

One improvement of the application is to do more error handling. Currently, there is not much error handling, so more print statements and checks could be implemented to make the application more robust against possible errors. A way to handle unsolvable sudoku puzzles should also be added.

Another thing that could be done to improve the application is to implement a different solving algorithm that may be faster. This application only finds one solution, but there can be several solutions to a sudoku board. Therefore, an improvement could be to make a solving algorithm that finds every solution.

Moreover, another way to improve the application is to add a way to check more thoroughly if the "solved" board actually is solved. Now, one has to check oneself if the board is a valid solution to the puzzle.

6 Conclusion

This report has explored the implementation of an application which solves sudoku puzzles. It uses different classes to set up the board with square objects that are assigned to a row, a column and a box. By using recursion that brute forces a solution, the application successfully finds one solution to solvable sudokus. Positives with the application is that it is simple and clear and has no circular import issues. However, the error handling and solving algorithm could be improved.

7 Acknowledgement

I thank my teaching assistant for providing some of the code as well as guidance. I am also grateful for Kylie Ying's video on how to solve a Sudoku puzzle using recursion, as my solving method is inspired by her video.

8 References

[1] javaTpoint. (n.d). javaTpoint. *Classes and Objects in Python* https://www.javatpoint.com/python-objects-classes

[2] T. Schenia. (2023, August 29). Copahost. *Init python: Learn how to use __init__ to initialize objects in python* https://www.copahost.com/blog/init-python/

[3] Codesdope. (n.d.). Codesdope. *Python Subclass of a class.* https://www.codesdope.com/course/python-subclass-of-a-class/

[4] T. Schenia. (2023, August 23). Copahost. *Append python: Learn to add elements efficiently* https://www.copahost.com/blog/append-python/

[5] W3schools. (n.d). W3schools. *Python – Remove List Items* https://www.w3schools.com/python/python_lists_remove.asp

[6] W3schools. (n.d). W3schools. *Python | Using 2D arrays/lists the right way* https://www.geeksforgeeks.org/python-using-2d-arrays-lists-the-right-way/

[7] Kylie Ying. (2020, October 26). *Coding a Sudoku solver in Python using recursion/backtracking*. [Video]. https://www.youtube.com/watch?v=tvP_FZ-D9Ng