Project 2 soduko Report

Phase1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ID | Academic Level | | Specialization |
| محمد علي ابراهيم احمد | 201901007 | Level 3 | | IS |
| فاطمه محمد فاروق | 202000642 | Level 3 | | IS |
| محمد حمدي عبد الرازق | 202000747 | |
| يوسف محمود قطب | 202001108 | Level 3 | | IS |
| ريم ايمن محمد محمود | 202000347 | Level 3 | | IS |
| ميار ماهر رشاد | 202000966 | Level 3 | | IS |
| ادهم على ابو حسيبة سالم | 202000100 | Level 3 | | IS |

***Contents***

:project idea in detail

:similar applications in the market

:A literature review of Academic publications

:Details of the main functionalities algorithm(s)/approach(es) used and the results of the experiments

:Development platform

[project idea in details](#project_idea_in_details)

**Sudoku is a logic puzzle**

**played on a 9x9 grid**

**subdivided into 9 3x3**

**subgrids. The player is**

**presented a partially**

**filled puzzle, and must**

**complete it following these rules:**

**Each column must contain**

**the numbers 1-9 (no**

**repeats!)**

**Each row must contain the numbers 1-9 (no repeats!)**

**Each 3x3 subgrid must contain the numbers 1-9 (no repeats!)**

**This program tests if a**

**Sudoku puzzle follows all**

**these rules and compares**

**two puzzles to see if they**

**are the same or different**

**:similar applications in the market**

**1-Bubble pop**

**2-Nonogram**

**3-sudoko master**

***:A literature review of Academic publications***

***1)***

**https://www.researchgate.net/profile/Amy-Langville/publication/228615106\_An\_integer\_programming\_model\_for\_the\_sudoku\_problem/links/5c0883e7a6fdcc494fdca89d/An-integer-programming-model-for-the-sudoku-problem.pdf**

### 3) [An integer programming model for the Sudoku problem](https://www.researchgate.net/profile/Amy-Langville/publication/228615106_An_integer_programming_model_for_the_sudoku_problem/links/5c0883e7a6fdcc494fdca89d/An-integer-programming-model-for-the-sudoku-problem.pdf)

Sudoku is a logic-based puzzle that first appeared in the U.S. under the title “Number Place” in 1979 in the magazine Dell Pencil Puzzles & Word Games [6]. The game was designed by Howard Garns, an architect who, upon retirement, turned to puzzle creation. In the 1980s, the game grew in popularity in Japan and was renamed by publisher Nikoli to “suji wa dokushin ni kagiru,” which translates as the “the digits must remain single.” This was eventually shortened to “sudoku” or “single number.” By 1997 an entrepreneur named Wayne Gould saw the financial potential available in the game. Gould spent six years refining his computer program so that it could quickly generate puzzles of varying levels of difficulty. In November 2004, Gould convinced The Times of London to print a puzzle. From there the popularity of the puzzle spread until now they commonly appear in a wide variety of newspapers and magazines. Interestingly, Gould does not charge newspapers for his puzzles, but they must include the Web address http://www.sudoku.com where his Sudoku program can be downloaded (for a free trial version and a fee for permanent use). Sudoku most commonly appears in its 9 × 9 matrix form. The rules are simple: fill in the matrix so that every row, column, and 3 × 3 submatrix contains the digits 1 through 9 exactly once. Each puzzle appears with a certain number of givens. The number and location of these determine the game’s level of difficulty. Figure 1 is an example of a 9 × 9 Sudoku puzzle.

INTEGER PROGRAMMING MODEL FOR SUDOKU

Description: A picture containing text, crossword puzzle

Description automatically generated

Figure 1: An example Sudoku puzzle This puzzle idea can accommodate games of other sizes. Of course, a 4 × 4 puzzle would be easier and a 16 × 16 puzzle harder. In general, any n × n game can be created, where n = m2 and m is any positive integer. There are numerous other variants of the game; see [9, 2, 8]. Sudoku puzzles elicit the following two interesting mathematical questions: • How can these puzzles be solved mathematically? • What mathematical techniques can be used to create these puzzles? In the following sections, we explore these questions. We present a binary integer linear program to solve this feasibility problem. Further, such an approach is extended to variations on the traditional Sudoku puzzle. In addition, we speculate as to how Sudoku puzzles are created, and provide several theorems for generating many new puzzles from one given original puzzle. Exercises and challenge problems that use principles from optimization, combinatorics, linear algebra, and computer science are presented for students. Answers to the exercises are contained at the conclusion of the article.

**2)**

<https://books.google.com.eg/books?hl=en&lr=&id=iK2ze0Lyy8sC&oi=fnd&pg=PR10&dq=article+about+sudoku+game&ots=9XEw8bEuPm&sig=cFxQ2ro-nruJURGUHsslaWUufqQ&redir_esc=y#v=onepage&q&f=false>

### [BOOK] [Programming Sudoku](https://books.google.com/books?hl=en&lr=&id=iK2ze0Lyy8sC&oi=fnd&pg=PR10&dq=article+about+sudoku+game&ots=9XEw8bEuPm&sig=cFxQ2ro-nruJURGUHsslaWUufqQ)

The Sudoku Problem Definition 1 A Sudoku puzzle is represented by a 9×9 grid, which comprises nine 3×3 sub-grids (also called boxes). Some of the entries in the grid are filled with numbers from 1 to 9, whereas other entries are left blank. Figure 1 is an example of a Sudoku puzzle. Puzzles are often also assigned a difficulty level, which usually depends on the number of initial non-blank entries provided. This number may be as few as 17 to test expert players. As we shall see, the numbers 1 through 9 are used solely for convenience; arithmetic relationships between them are completely irrelevant. Hence, any set of distinct symbols could have been used. Definition 2 A Sudoku puzzle is solved by assigning numbers from 1 to 9 to the blank entries such that every row, every column, and every 3×3 sub-grid contains each of the nine possible numbers. Interestingly, this rule explains why Sudoku means “single number” in Japanese. Although the definition above characterizes Sudoku, the puzzles available in the entertainment literature have two additional properties. In the remainder of this paper, we will only consider Sudoku puzzles that have these properties, i.e.: Property 1 Sudoku puzzles that have only one solution. Property 2 Sudoku puzzles that can be solved with only reasoning, i.e., with no search. Having only one solution means that all the numbers to be assigned to the blank entries are necessary assignments. The second property requires in addition that, at any stage in the course of solving the puzzle, there should always be at least one blank entry that can be assigned to merely by considering what is immediately implied by the set of non-blank entries. Hence, reasoning consists in using inference rules in such a way that all of the assignments are found. To illustrate how to solve a Sudoku puzzle, let us consider entry (a) in the left-hand grid of Figure 2. Considering the relevant 3×3 sub-grid, this is the only position where number 3 can be placed. Also, consider entry (b) in the right-hand grid of the same figure. It is clear that (b) is the only position where number 7 can be placed in the second row

:Details of the main functionalities algorithm(s)/approach(es) used and the results of the experiments

First in methods from 1:5

Prepare the environment of

Shape of sudoku

1)

def cross(a, b):

    return [s+t for s in a for t in b]

"""

The code is trying to create a list of squares.

The first line creates the list of rows and columns, which is then used in the next two lines to create lists of row units and column units.

These are then combined into a single list called square\_units that contains all possible combinations of rows, columns, and squares.

"""

2) def grid\_values(grid):

    """

    Convert grid into a dict of {square: char} with '123456789' for empties.

    Input: A grid in string form.

    Output: A grid in dictionary form

            Keys: The boxes, e.g., 'A1'

            Values: The value in each box, e.g., '8'. If the box has no value, then the value will be '123456789'.

    """

3)

def eliminate(values):

    """

    Go through all the boxes, and whenever there is a box with a value, eliminate this value from the values of all its peers.

    Input: A sudoku in dictionary form.

    Output: The resulting sudoku in dictionary form.

    """

4) def only\_choice(values):

    """

    Go through all the units, and whenever there is a unit with a value that only fits in one box, assign the value to this box.

    Input: A sudoku in dictionary form.

    Output: The resulting sudoku in dictionary form.

    """

5) def reduce\_puzzle(values):

    """

    Iterate eliminate() and only\_choice(). If at some point, there is a box with no available values, return False.

    If the sudoku is solved, return the sudoku.

    If after an iteration of both functions, the sudoku remains the same, return the sudoku.

    Input: A sudoku in dictionary form.

    Output: The resulting sudoku in dictionary form.

    """

6)the method search use depth first search with backtracking and elimination methods

In the utilities,reduce, puzzle,only choice,

differential evaluation

def search(values):

    "Using depth-first search and propagation, try all possible values."

    """The code is a function that uses depth-first search and propagation to try all possible values.

    The first step is to reduce the puzzle using the previous function, which takes in an array of integers as input and returns False if there are no solutions.

    If all of the boxes have only one value, then it's solved!

    Otherwise, choose one of the unfilled squares with the fewest possibilities (the min() function) by choosing n=min(len(boxes), s).

    Now use recurrence to solve each one of these resulting sudokus by calling this function on each element in new\_sudoku.

    The code starts off by creating a list called values that contains all possible values for our puzzle.

    It then reduces this list down into smaller lists based on how many boxes we have left over from earlier reductions: values = reduce\_puzzle(values) If there are still more than two boxes left over after reducing them down, then we'll need to find another solution for those remaining boxes.

    We do so by finding out what box has fewer numbers in it than any other box (using len()): if len(boxes) > 1: n, s = min((len(boxes

The code has a recursive function that will search through all the values in the list of boxes, and if it finds a solution, return True."""

    # First, reduce the puzzle using the previous function

7)prepare the gui from 7:10 using python tkinter

 def createEntries(self):

        """The code creates a grid of 9 rows and columns.

        The first row is numbered 0, the second row is numbered 1, etc.

        Each entry in the grid has a width of 3 pixels.

        The code creates an Entry object for each column and then appends it to the root node with borderwidth=2 so that it will be on top of any other entries in that column.

        Then it sets its width to 3 pixels so that all three sides are equal size.

        The code creates nine entries in a grid.

        The first line of the code assigns a value to the variable i.

        The next line of the code assigns a value to the variable j.

        The third line of the code appends an entry to the list of entries in self.gridEntry with an ID of root and borderwidth=2, width=3.

        The fourth line iterates through all nine lines and places each iteration's row number and column number into self.gridEntry[-1].

        This is done because there are nine iterations and -1 is used as a placeholder for iteration 0 (the last iteration).

        """

8) def getGrid(self):

        """he code is trying to get the grid values.

        If an item is not found in the list, then it will return a '.'.

        Otherwise, it will return the value of that item.

        The code returns a list with all of the items in it and their respective values if they are found or else just a '.'

        for any other case.

        The code returns a list of strings, each string representing the value of the grid entry.

        The code is used to return a list of values from the grid.

        """

9)

  def solve\_sudoku(self):

        """The code starts by creating a grid of 9x9 squares.

        The next line creates a canvas and sets it to the size of 300x300 pixels.

        Next, we create a text object that will display the values in our grid.

        Finally, we use search() to find all possible solutions for each row and column in our grid.

        The code creates a canvas with the dimensions of 300x300.

        The code then creates a grid on the canvas with 9 rows and 9 columns.

        The code then places text on the canvas to display the solved grid.

        """

10) def perform(self):

        """The code starts by creating a new tk.Button object called "button" and setting its text to "Solve".

        The code then creates a function that will be executed when the button is clicked, which calls the solve\_sudoku() function.

        This function sets up an empty grid with nine rows and columns in it, then adds the button to this grid at row 9, column 0.

        The main loop of the program starts here because we have not yet added any other code to it.

        The code attempts to create a button that when clicked, will call the function solve\_sudoku() which will be defined in the following code.

        def solve\_sudoku(): print('Solve') return True"""

:Development platform

windows 10 64bit,

microsoft vscode code editor,

16core cpu,12gb ram

