Analysis

Introduction

This project will be based around the Sudoku game and will delve into the foundation of this puzzle. We will look to implement the sudoku game as an interactive technical solution and provide additional features to enhance the user's experience.

Problem

The program being investigated is generating a puzzle that is challenging yet has a unique solution and solvable for the user to engage with. For the user to interact with the game, it must allow users to input values into the puzzle. After the grid is filled, verify whether the grid has been correctly filled out, otherwise, display the incorrect value in a way that could be differentiated from other values.

Background Information

Sudoku is a popular logic-based puzzle game that has gained the interest of millions worldwide. The goal is to complete the 9x9 grid with numbers from 1 to 9, while ensuring that each row, column and 3x3 sub grid has a unique number inside the given range. Moreover, there levels to choose ranging from easy to hard and requires users to utilise their deduction and problem-solving skills in order to conclude the game. The game attracted significant attention because of its challenging puzzles, offering players a sense of accomplishment upon completion. Furthermore, it aids user to relieve stress by shifting their focus to the game.

Target Audience

After consulting with family members and friends who have various levels of experiences with Sudoku, I gathered their feedback on the desired game features.

Question: What features would you to see incorporated into this game?

Response 1: provide various levels of difficulties so that their friends could also enjoy the game when using the same device regardless of their skills

Response 2: save the unfinished puzzle and include a timer that stops when the puzzle is saved and continues to run when the puzzle is accessed again

Based on the responses above and additional inputs, the common requests included the ability to save and load the previous game, an error checking feature that differentiate the incorrect inputs when displayed, a timer to time themselves for each puzzle alongside a history log that keep tracks of their time taken.

Objectives

1. Generate a fully solved sudoku grid
2. Remove cells from the generated grid
3. Ensure generated puzzle has a unique solution
4. Validate the user inputs
5. Save and load the puzzles
6. Timer continues to run when user loads the game

Techniques Research

- Researched on generating a full sudoku grid
- Keep the game running even with empty cells in the grid
- Converting a string in a text file back into the 2d array with desired data types

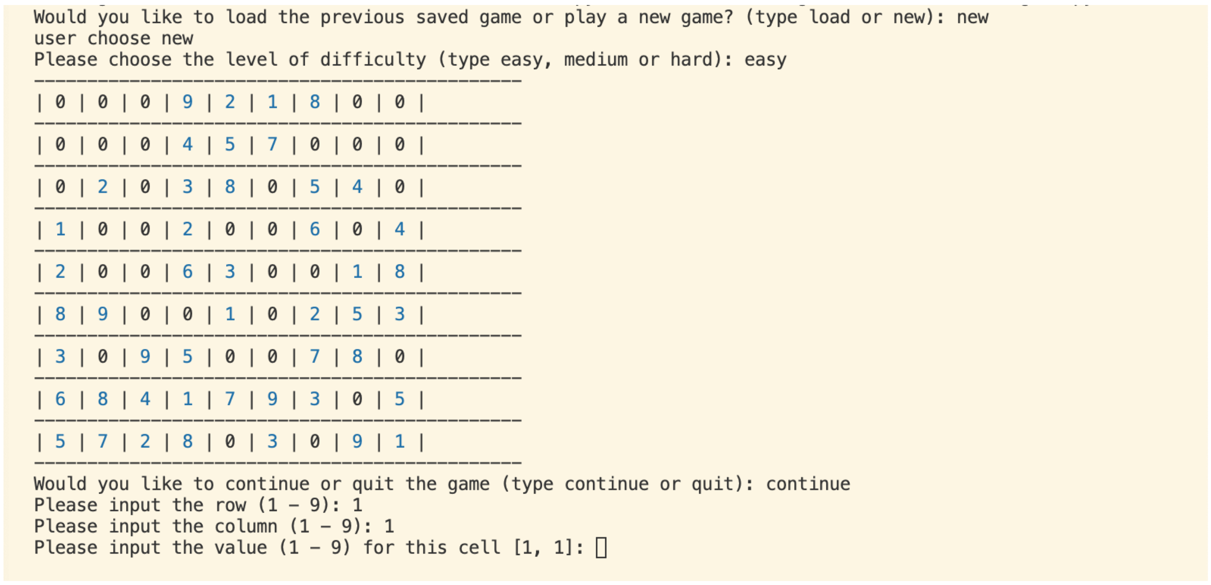
Design

Sudoku

- Random generator
  - Generate a complete valid sudoku
- Generate puzzle
  - Use generated puzzle
  - Randomly hide the number of cells according to the levels of difficulty
- Levels
  - Hard: 19 – 26 clues remove 55 - 62 cells
  - Medium: 27 – 36 clues; remove 45 - 54 cells
  - Easy: given cells 38 - 45 clues; remove 36 - 43 cells
  - There must be a minimum of 17 given cells for the puzzle to be solved
- Check input
  - Number in each element is in range of 1 – 9
  - Check there is no duplication in each row
  - Check for no duplication for each 3x3 grid
- Game Over
  - Check if all cells are filled in (each cell ≠ 0)
    - Check if all the inputs are correct
      - Game is finished
- Solution
  - Backtracking algorithm
- Save and load unfinished progress
  - If the previous game is unfinish, save it so user can return
  - Load the unfinished progress when user wishes to continue with their previous game
- Grid
  - 9x9 grid
  - Print grid nicely for user

Overview of the Game Design

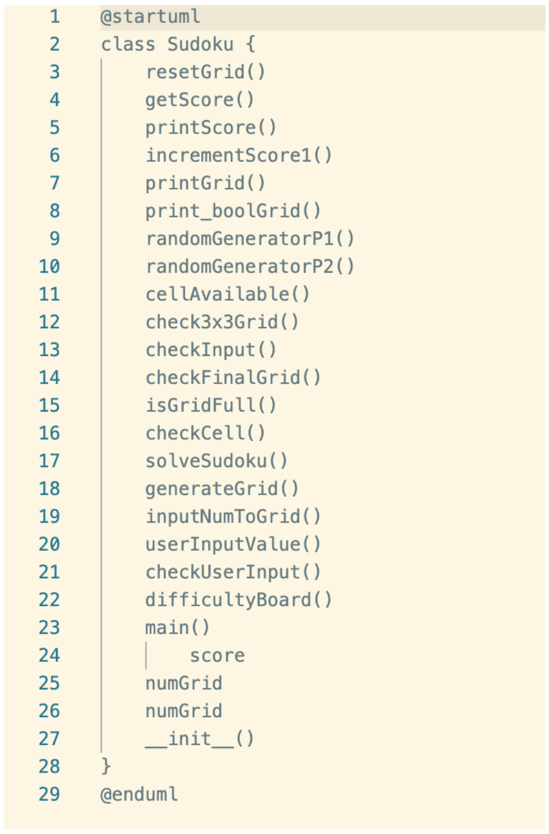
User’s View



```
Would you like to load the previous saved game or play a new game? (type load or new): new
user choose new
Please choose the level of difficulty (type easy, medium or hard): easy
---------------------------------------------
| 0 | 0 | 0 | 9 | 2 | 1 | 8 | 0 | 0 |
---------------------------------------------
| 0 | 0 | 0 | 4 | 5 | 7 | 0 | 0 | 0 |
---------------------------------------------
| 0 | 2 | 0 | 3 | 8 | 0 | 5 | 4 | 0 |
---------------------------------------------
| 1 | 0 | 0 | 2 | 0 | 0 | 6 | 0 | 4 |
---------------------------------------------
| 2 | 0 | 0 | 6 | 3 | 0 | 0 | 1 | 8 |
---------------------------------------------
| 8 | 9 | 0 | 0 | 1 | 0 | 2 | 5 | 3 |
---------------------------------------------
| 3 | 0 | 9 | 5 | 0 | 0 | 7 | 8 | 0 |
---------------------------------------------
| 6 | 8 | 4 | 1 | 7 | 9 | 3 | 0 | 5 |
---------------------------------------------
| 5 | 7 | 2 | 8 | 0 | 3 | 0 | 9 | 1 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit): continue
Please input the row (1 – 9): 1
Please input the column (1 – 9): 1
Please input the value (1 – 9) for this cell [1, 1]: 
```

Figure 1: an example of user input

Class Diagram



```
@startuml
class Sudoku {
    resetGrid()
    getScore()
    printScore()
    incrementScore1()
    printGrid()
    print_boolGrid()
    randomGeneratorP1()
    randomGeneratorP2()
    cellAvailable()
    check3x3Grid()
    checkInput()
    checkFinalGrid()
    isGridFull()
    checkCell()
    solveSudoku()
    generateGrid()
    inputNumToGrid()
    userInputValue()
    checkUserInput()
    difficultyBoard()
    main()
        score
    numGrid
    numGrid
    __init__()
}
@enduml
```

Figure 2: class diagram

What the project will allow user to do

1. Option to resume their saved game or start a new game
2. Allows user to select the level of difficulty
3. Allows user to continue playing or exit the game after changing the values in the sudoku
4. Upon deciding to exit, user can choose to save their progress
5. User input a number into the chosen cell
6. The program places the number into the designated cell and return to step 3 and repeats
7. When the grid is filled with all the correct inputs, the game ends
8. The game will continue if all the inputs are not correct

Finding which 3 x 3 table the chosen cell is in

	0	1	2	3	4	5	6	7	8
0	(0,0)			(3,0)			(6,0)		
1									
2			(2,2)			(5,2)			(8,2)
3	(0,3)			(3,3)			(6,3)		
4									
5			(2,5)			(5,5)			(8,5)
6	(0,6)			(3,6)			(6,6)		
7									
8			(2,8)			(5,8)			(8,8)

Table 1

Method 1: Find upper left corner: [(x , y) DIV 3) * 3]

e.g. (5 , 3) DIV 3 = (1 , 1) {assumes that the coordinates are not starting with index 0}

take the quotient only

(1 , 1) * 3 = (3 , 3) -> the cell is located in the 3x3 grid which starts at the coordinate (3 , 3)

Function: divmod (x , y)

e.g. print = divmod (5 , 3) -> 5/3

= (1, 2) -> gives the quotient and remainder

print = divmod (3 , 3) -> 3/3

= (1, 0) -> 1 is the quotient and there are no remainder

Method 2: Comparing the x and y coordinates to the coordinates of the 3x3 grids

e.g. (3 , 4)

Subtract 1 from the x and y coordinates as the array starts from the index 0

(3 , 4) -> (2 , 3)

Determine whether the x coordinate is smaller than 3, bigger than 5 or between 3 and 5

X coordinate = 2

$\Rightarrow$ $2 < 3$

This indicates that the cell is located within the first 3x3 column

Apply the same step for the y coordinate

Y coordinate = 3

$\Rightarrow$ $3 \nless 3$
$\Rightarrow$ $3 \ngtr 5$
$\Rightarrow$ 3 in range of 3 to 5

This suggests that the cell is contained within the second 3x3 row

Multidimensional array

I utilised a 2D array to depict the grid, storing in the variables '*self.numGrid*' and '*self.boolGrid*'. each element in the array corresponds to an individual cell, similar to a sudoku puzzle, with the selected cell is represented as 'self.numGrid[y][x]', where y signifies the row number and x denotes the column number.

e.g. user would like to access the value at (6 , 1)

x and y coordinates will be deduced by 1 in the program

(6 , 1) -> self.numGrid[5][0] {because the arrays start at an index of 0}

Backtracking algorithm

The algorithm operates similar to a trial-and-error problem, requiring a return to the previous step and try a new approach if the current step has a dead end. In this case, after the diagonal 3x3 grids has been randomly filled in, the algorithm will attempt to assign a number ranging from 1 to 9 for each empty cell and will determine whether the puzzle is completed successfully in accordance with the game regulations. Nevertheless, assuming that the solution of puzzle is unsuccessful, the algorithms will backtrack and try an alternative method.

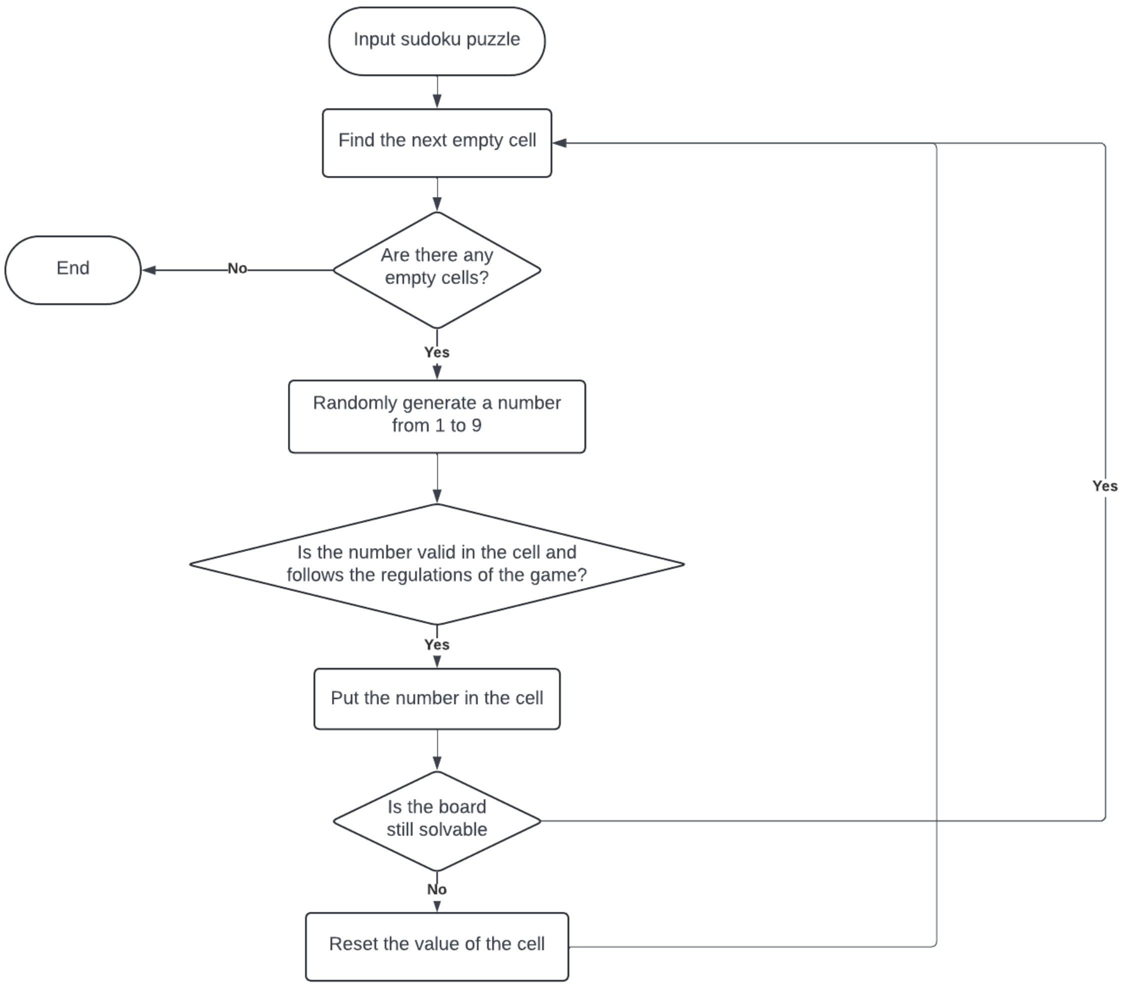
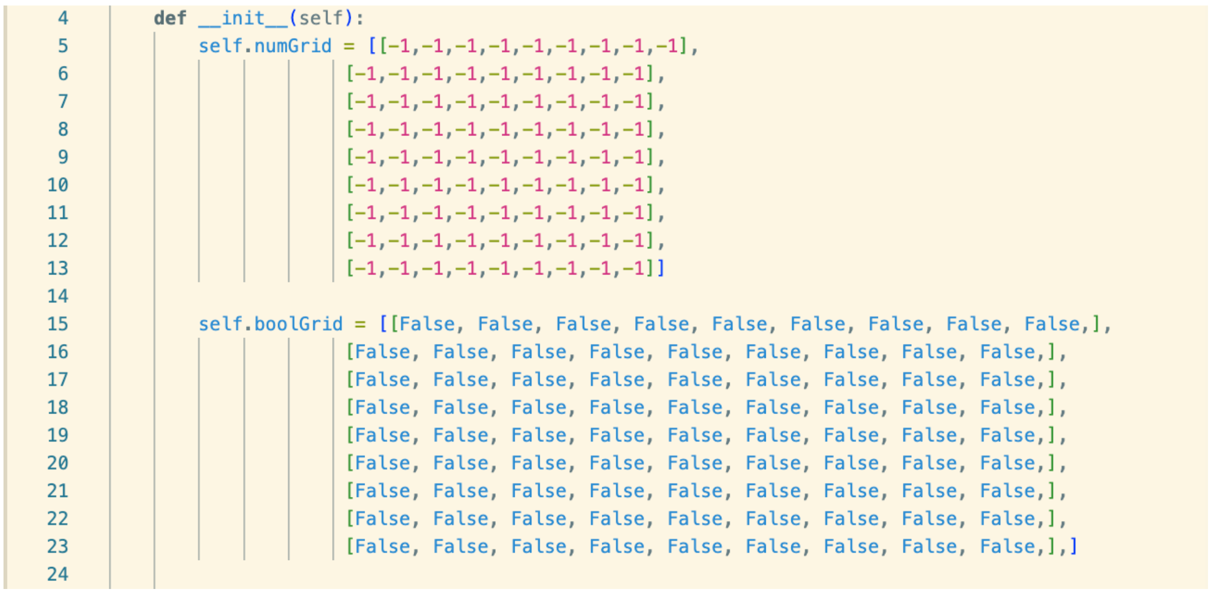

Input sudoku puzzle
Find the next empty cell
Are there any empty cells?
No
End
Yes
Randomly generate a number from 1 to 9
Is the number valid in the cell and follows the regulations of the game?
Yes
Put the number in the cell
Is the board still solvable
Yes
No
Reset the value of the cell

Diagram 1

Technical Solutions

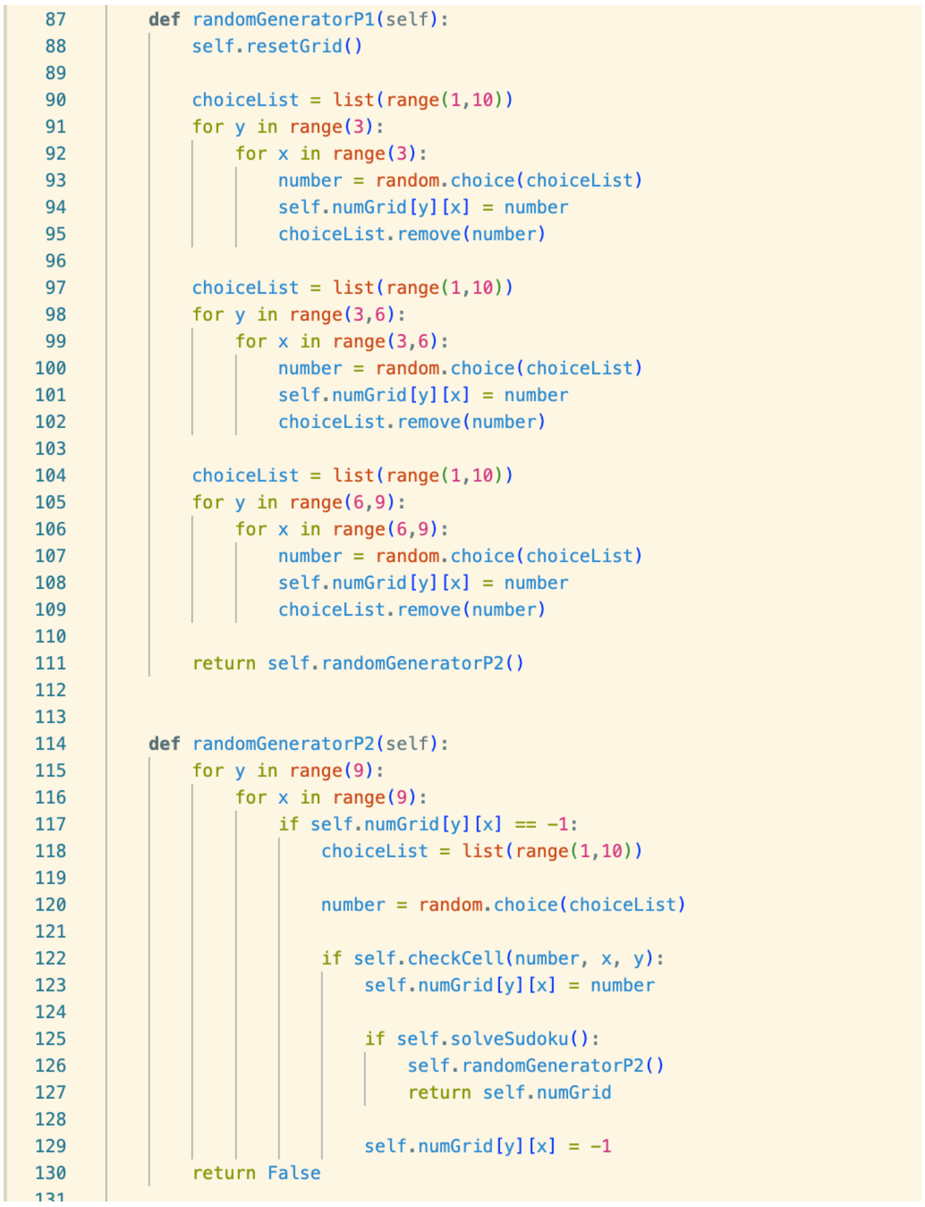
Please find full code listing in the appendix at the end of this document



```
    def __init__(self):
        self.numGrid = [[-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1]]

        self.boolGrid = [[False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],
                     [False, False, False, False, False, False, False, False, False,],]

```

Figure 3: 2d array is carried out in these two variables

Figure 3 illustrated above shows the two variables applying multidimensional array more specifically 2d arrays. ‘*self.numGrid*’ represents the value of each cell in the sudoku grid and is set as -1 as the initial value. This is done so to avoid confusion because this distinct the empty cell within the program and the empty cell when program is run to obtain the user’s input. Utilising a 2d array to represent the grid provides a structure to match a sudoku grid, where each cell in the 2d arrays holds the corresponding value in the puzzle. The ‘*self.numGrid*’ stores of value of each cell in the grid into, whereas the ‘*self.boolGrid*’ store the Boolean value for whether the value stored in the cell is valid or not.

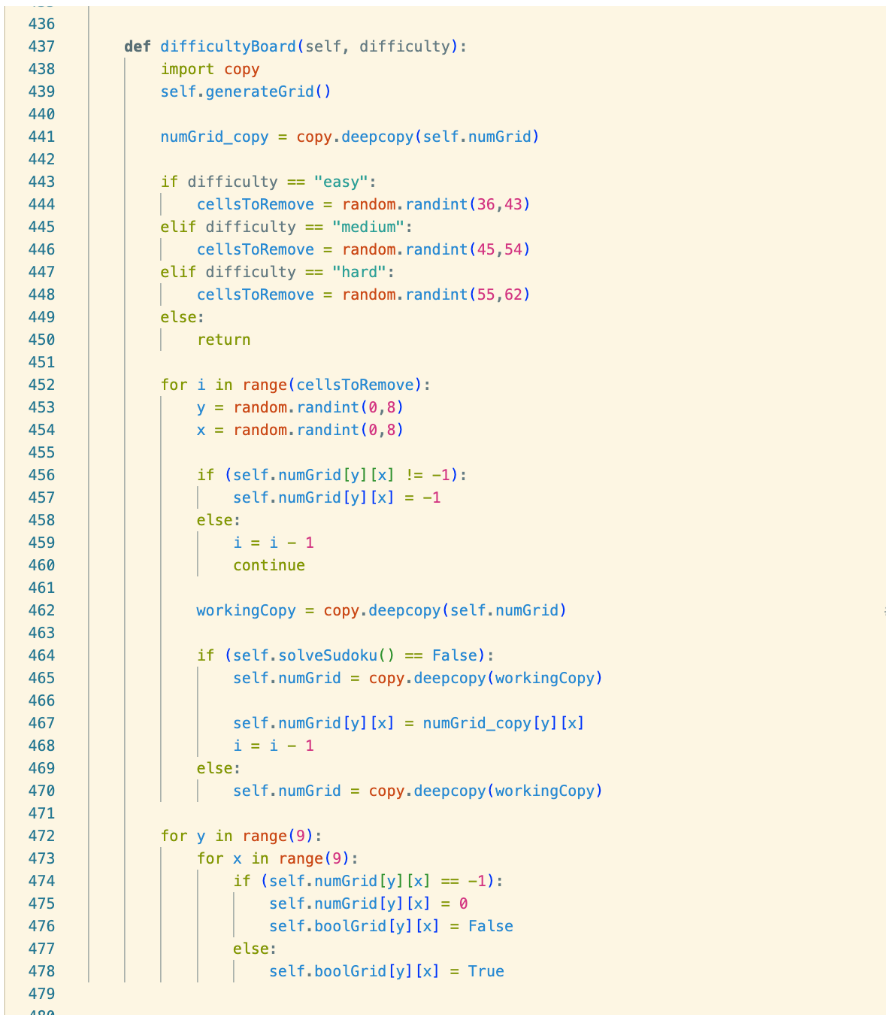


```
    def randomGeneratorP1(self):
        self.resetGrid()

        choiceList = list(range(1,10))
        for y in range(3):
            for x in range(3):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        choiceList = list(range(1,10))
        for y in range(3,6):
            for x in range(3,6):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        choiceList = list(range(1,10))
        for y in range(6,9):
            for x in range(6,9):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        return self.randomGeneratorP2()


    def randomGeneratorP2(self):
        for y in range(9):
            for x in range(9):
                if self.numGrid[y][x] == -1:
                    choiceList = list(range(1,10))

                    number = random.choice(choiceList)

                    if self.checkCell(number, x, y):
                        self.numGrid[y][x] = number

                        if self.solveSudoku():
                            self.randomGeneratorP2()
                            return self.numGrid

                        self.numGrid[y][x] = -1
        return False
```

Figure 4: two functions that is combined to generate the solved grid

In *Figure 4*, the two functions shown above have been implemented to generate a solved grid. First, the ‘*randomGeneratorP1*’ function is reset to the default value of -1, which then randomly fills in the diagonal of 3x3 grids (the top left, middle and bottom middle). This process ensure that each puzzle has a unique starting point resulting to a variety of puzzles that could be generated. Therefore, this minimises chance of repetition each time a new puzzle is generated. Thus, it improves the user’s experienced by providing new challenges with each new puzzle.

Diagram 1 shows the algorithm used by the ‘*randomGeneratorP2*’. It utilises a backtracking algorithm to find the next empty cell. Consequently, generates a random number between 1 and 9 for that cell, which is then validated against the game’s regulations. If the appropriate number is generated, it will be place in the grid, otherwise, the cell’s value is reset and moves onto the next empty cell. This process is repeated until there are no empty cells left in the grid.



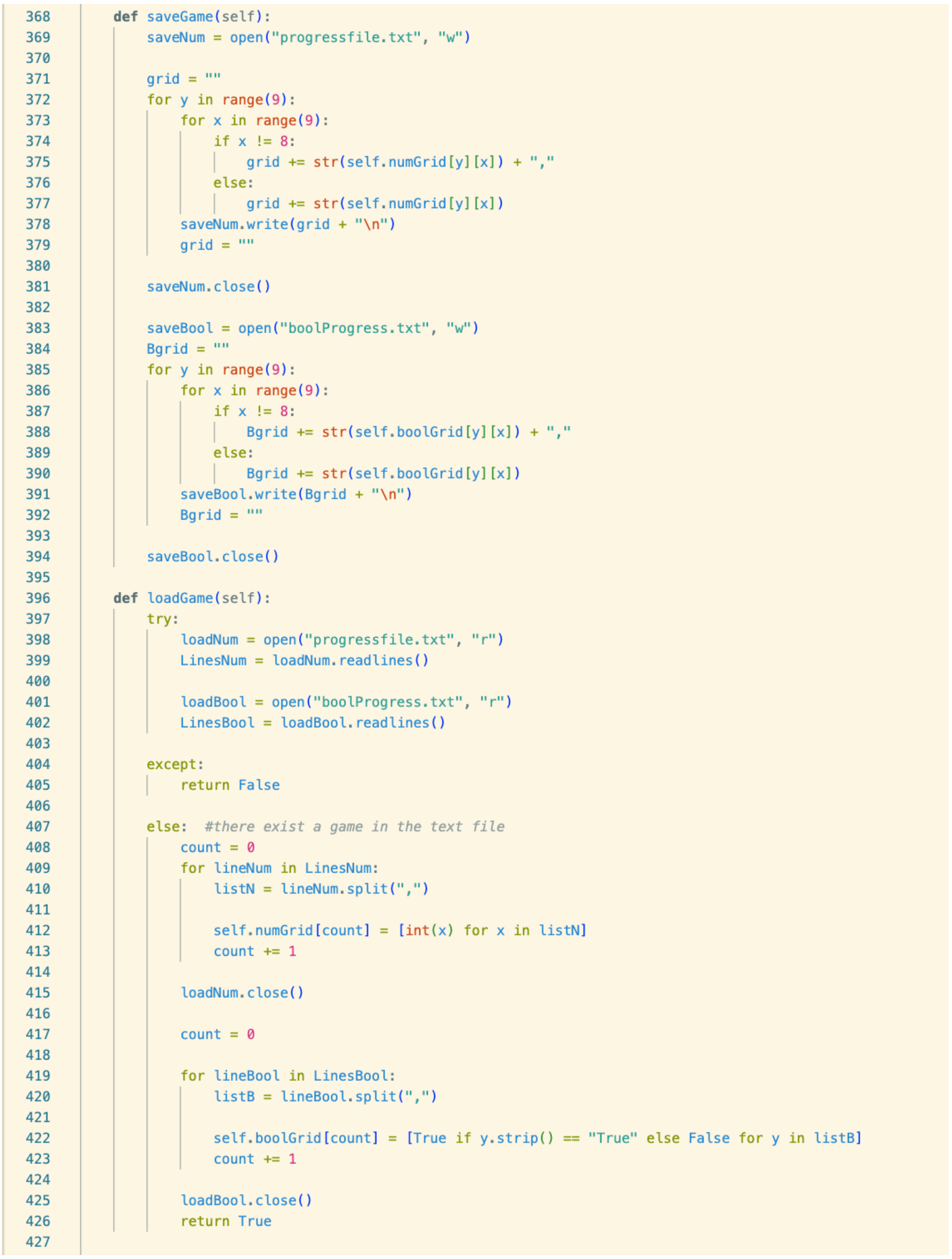
```python
    def difficultyBoard(self, difficulty):
        import copy
        self.generateGrid()

        numGrid_copy = copy.deepcopy(self.numGrid)

        if difficulty == "easy":
            cellsToRemove = random.randint(36,43)
        elif difficulty == "medium":
            cellsToRemove = random.randint(45,54)
        elif difficulty == "hard":
            cellsToRemove = random.randint(55,62)
        else:
            return

        for i in range(cellsToRemove):
            y = random.randint(0,8)
            x = random.randint(0,8)

            if (self.numGrid[y][x] != -1):
                self.numGrid[y][x] = -1
            else:
                i = i - 1
                continue

            workingCopy = copy.deepcopy(self.numGrid)

            if (self.solveSudoku() == False):
                self.numGrid = copy.deepcopy(workingCopy)

                self.numGrid[y][x] = numGrid_copy[y][x]
                i = i - 1
            else:
                self.numGrid = copy.deepcopy(workingCopy)

        for y in range(9):
            for x in range(9):
                if (self.numGrid[y][x] == -1):
                    self.numGrid[y][x] = 0
                    self.boolGrid[y][x] = False
                else:
                    self.boolGrid[y][x] = True
```

Figure 5: function that removes cells to create puzzle

As shown in *Figure 5*, the '*difficulty*' parameter is derived from the user's input in the main function and is used to determine the range of cells that should be removed from the grid in order to generate a varied puzzle for each difficulty level. Initially, a duplicated of the generated solved grid named '*numGrid_copy*' is created. To ensure that the grid is solvable, the number of cells that should be removed should not exceed 64.

The process of removing random cells is performed by selecting random cells and confirming whether they contain a number. If a number is present, it is replaced with an empty cell, otherwise, the loop will be deduced by 1. A copy of the modified grid '*workingCopy*' is produced. The modified grid is then analysed to determine whether it is solvable. If it is unsolvable, the cell's value from '*numGrid_copy*' is restored and the loop count is decremented, elsewise, '*workingCopy*' replaces the current grid. This loop is repeated until the desired number of cells is removed.

Finally, a loop is implemented to covert cells that contains -1 to 0, along with setting the cell at '*self.boolGrid*' to False, the remaining cells are set to True.



```python
    def saveGame(self):
        saveNum = open("progressfile.txt", "w")

        grid = ""
        for y in range(9):
            for x in range(9):
                if x != 8:
                    grid += str(self.numGrid[y][x]) + ","
                else:
                    grid += str(self.numGrid[y][x])
            saveNum.write(grid + "\n")
            grid = ""

        saveNum.close()

        saveBool = open("boolProgress.txt", "w")
        Bgrid = ""
        for y in range(9):
            for x in range(9):
                if x != 8:
                    Bgrid += str(self.boolGrid[y][x]) + ","
                else:
                    Bgrid += str(self.boolGrid[y][x])
            saveBool.write(Bgrid + "\n")
            Bgrid = ""

        saveBool.close()

    def loadGame(self):
        try:
            loadNum = open("progressfile.txt", "r")
            LinesNum = loadNum.readlines()

            loadBool = open("boolProgress.txt", "r")
            LinesBool = loadBool.readlines()

        except:
            return False

        else:  #there exist a game in the text file
            count = 0
            for lineNum in LinesNum:
                listN = lineNum.split(",")

                self.numGrid[count] = [int(x) for x in listN]
                count += 1

            loadNum.close()

            count = 0

            for lineBool in LinesBool:
                listB = lineBool.split(",")

                self.boolGrid[count] = [True if y.strip() == "True" else False for y in listB]
                count += 1

            loadBool.close()
            return True

```

Figure 6: functions that saves and load the puzzle

As illustrated in *Figure 6*, there are two functions which allows the user to store and retrieve their puzzle if incomplete. The '*saveGame*' function opens the file '*progressfile.txt*' if it exists, otherwise, it creates the file '*progressfile.txt*'. It then loops through the row and column of the '*self.numGrid*' converts the values in it into a string to write into the file and closes it after it is done. The same thing is done to the '*self.boolGrid*', storing it in the file '*boolProgress.txt*'

The '*loadGame*' function first try to see whether the files exist. If there both files exist, the grid stored will be transformed to the 2d array with the correct data types and returns True. For instance, grid stored in '*progressfile.txt*' turns back to an integer in '*self.numGrid*' and '*boolProgress.txt*' changes to a Boolean in '*self.boolGrid*'. In the case that the files do not exists, the function will return False.

Testing

While the program is running, some tests are performed to check various aspects of the game. These include validating the solved grid's uniqueness, the existence of a previous saved puzzle, user interactions and whether the generated puzzle matches the one encountered before with the same level of difficulty.

User Validation

Objective: validate the user inputs (row, column, number, difficulty, load or new input and continue or quit input)

New Puzzle Input

Test steps (new puzzle input, difficulty)

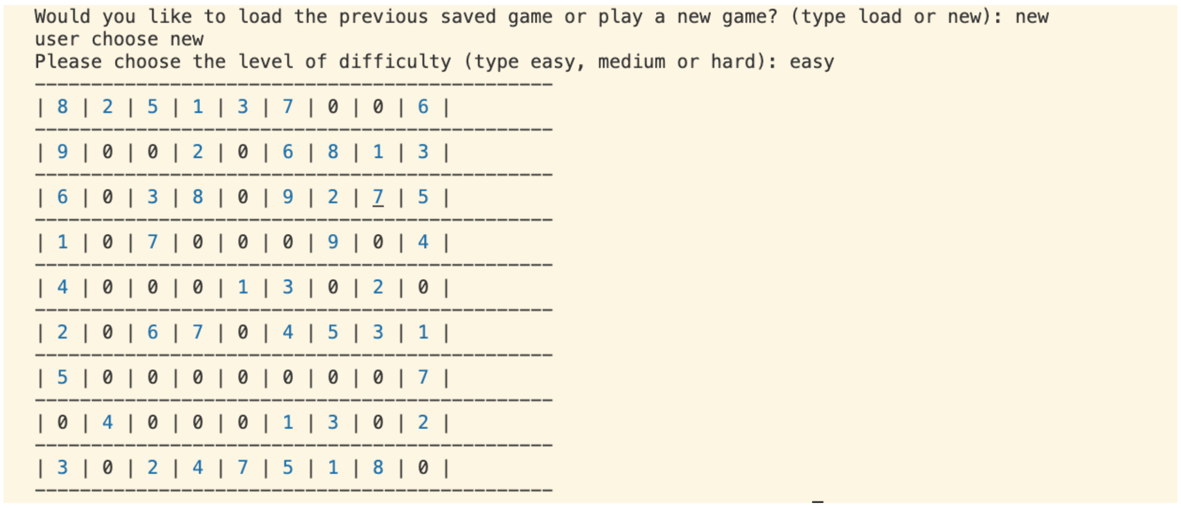
1. Input new
2. Input difficulty level (easy, medium or hard)

Expected Result:

- Game should display 9x9 grid where the number of prefilled cells should align with the corresponding difficulty level
- Digits that are displayed in each row should be unique
- Digits that are displayed in each column should be unique
- Digits displayed in each 3x3 sub grid should be unique
- Digits that are displayed should be in blue text
- All digits in the grid should be from 1 to 9

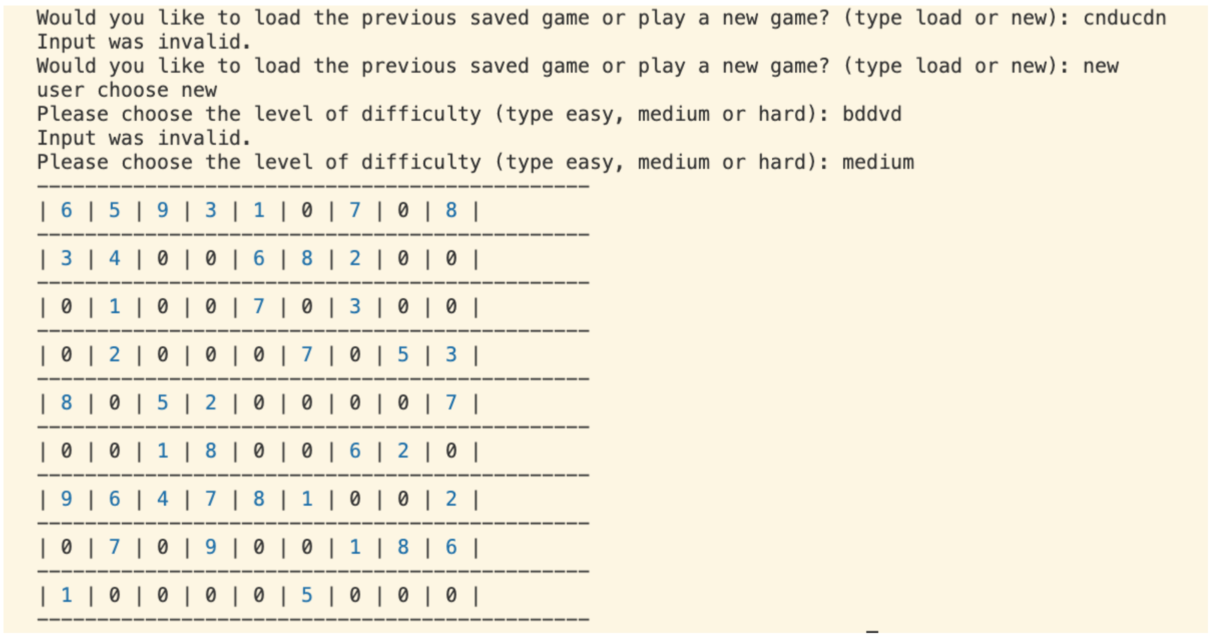
Test outcome:

- If user input "new", program will ask for level of difficulty
- If user input other than "new" or "load", program will send message that input is invalid and will continue to ask for an input (new or load)
- If user input level of difficulty ("easy", "medium" or "hard"), grid will be print out
- If user input level of difficulty other than ("easy", "medium" or "hard"), program will send message that input is invalid and continue to ask for input ("easy", "medium" or "hard")



```
Would you like to load the previous saved game or play a new game? (type load or new): new
user choose new
Please choose the level of difficulty (type easy, medium or hard): easy
---------------------------------------------
| 8 | 2 | 5 | 1 | 3 | 7 | 0 | 0 | 6 |
---------------------------------------------
| 9 | 0 | 0 | 2 | 0 | 6 | 8 | 1 | 3 |
---------------------------------------------
| 6 | 0 | 3 | 8 | 0 | 9 | 2 | 7 | 5 |
---------------------------------------------
| 1 | 0 | 7 | 0 | 0 | 0 | 9 | 0 | 4 |
---------------------------------------------
| 4 | 0 | 0 | 0 | 1 | 3 | 0 | 2 | 0 |
---------------------------------------------
| 2 | 0 | 6 | 7 | 0 | 4 | 5 | 3 | 1 |
---------------------------------------------
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |
---------------------------------------------
| 0 | 4 | 0 | 0 | 0 | 1 | 3 | 0 | 2 |
---------------------------------------------
| 3 | 0 | 2 | 4 | 7 | 5 | 1 | 8 | 0 |
---------------------------------------------
```

Figure 7: outcome when input is correct



```
Would you like to load the previous saved game or play a new game? (type load or new): cnducdn
Input was invalid.
Would you like to load the previous saved game or play a new game? (type load or new): new
user choose new
Please choose the level of difficulty (type easy, medium or hard): bddvd
Input was invalid.
Please choose the level of difficulty (type easy, medium or hard): medium
---------------------------------------------
| 6 | 5 | 9 | 3 | 1 | 0 | 7 | 0 | 8 |
---------------------------------------------
| 3 | 4 | 0 | 0 | 6 | 8 | 2 | 0 | 0 |
---------------------------------------------
| 0 | 1 | 0 | 0 | 7 | 0 | 3 | 0 | 0 |
---------------------------------------------
| 0 | 2 | 0 | 0 | 0 | 7 | 0 | 5 | 3 |
---------------------------------------------
| 8 | 0 | 5 | 2 | 0 | 0 | 0 | 0 | 7 |
---------------------------------------------
| 0 | 0 | 1 | 8 | 0 | 0 | 6 | 2 | 0 |
---------------------------------------------
| 9 | 6 | 4 | 7 | 8 | 1 | 0 | 0 | 2 |
---------------------------------------------
| 0 | 7 | 0 | 9 | 0 | 0 | 1 | 8 | 6 |
---------------------------------------------
| 1 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
---------------------------------------------
```

Figure 8: outcome when there is an invalid input

Load Puzzle Input

Test steps (load puzzle input, difficulty)

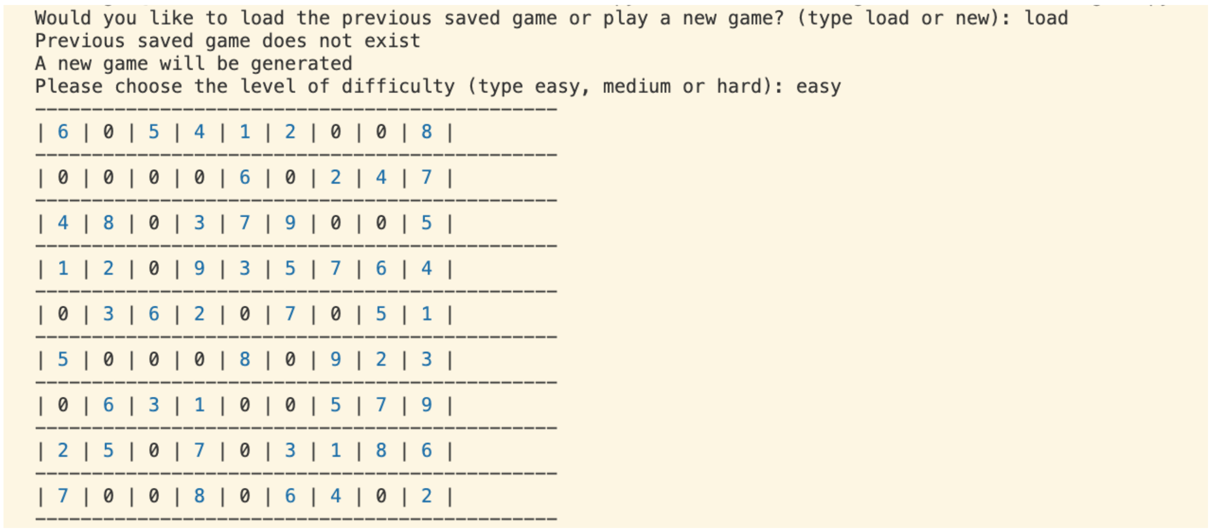
1. Input load
2. Input difficulty level (easy, medium or hard)

Expected Result:

- When load is chosen, program should display saved grid
- When load is failed, new puzzle will be generated

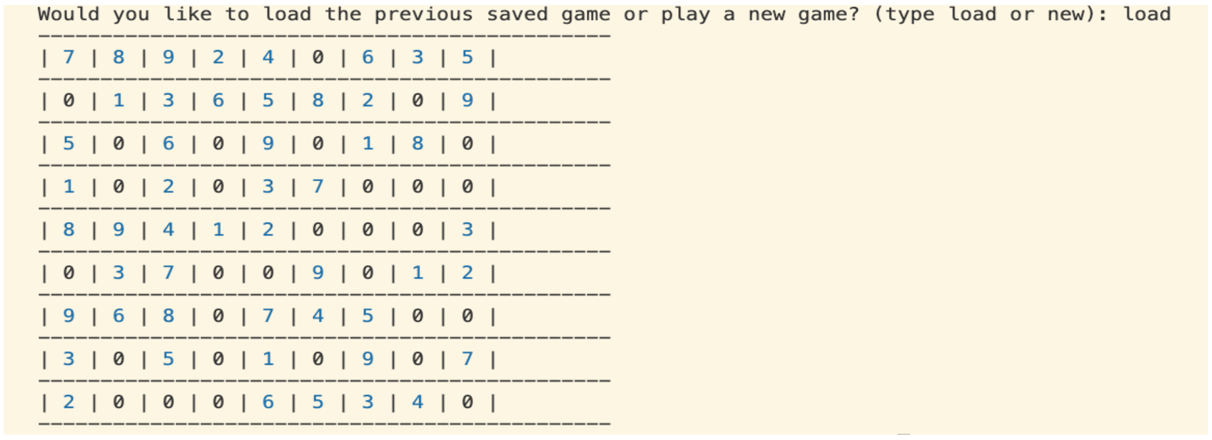
Test outcome:

- If user input “load”, program will check if there is a previous puzzle saved
- If user input other than “new” or “load”, program will send message that input is invalid and will continue to ask for an input (new or load)
- If saved previous puzzle does not exist, program will send message that previous game does not exist, a new game will be generated and will ask for level of difficulty for the new puzzle
- If user input level of difficulty (“easy”, “medium” or “hard”), grid will be print out
- If user input level of difficulty other than (“easy”, “medium” or “hard”), program will send message that input is invalid and continue to ask for input (“easy”, “medium” or “hard”)



```
Would you like to load the previous saved game or play a new game? (type load or new): load
Previous saved game does not exist
A new game will be generated
Please choose the level of difficulty (type easy, medium or hard): easy
----------------------------------------------
| 6 | 0 | 5 | 4 | 1 | 2 | 0 | 0 | 8 |
----------------------------------------------
| 0 | 0 | 0 | 0 | 6 | 0 | 2 | 4 | 7 |
----------------------------------------------
| 4 | 8 | 0 | 3 | 7 | 9 | 0 | 0 | 5 |
----------------------------------------------
| 1 | 2 | 0 | 9 | 3 | 5 | 7 | 6 | 4 |
----------------------------------------------
| 0 | 3 | 6 | 2 | 0 | 7 | 0 | 5 | 1 |
----------------------------------------------
| 5 | 0 | 0 | 0 | 8 | 0 | 9 | 2 | 3 |
----------------------------------------------
| 0 | 6 | 3 | 1 | 0 | 0 | 5 | 7 | 9 |
----------------------------------------------
| 2 | 5 | 0 | 7 | 0 | 3 | 1 | 8 | 6 |
----------------------------------------------
| 7 | 0 | 0 | 8 | 0 | 6 | 4 | 0 | 2 |
----------------------------------------------
```

Figure 9: outcome when user chooses load and there is no saved puzzle



```
Would you like to load the previous saved game or play a new game? (type load or new): load
----------------------------------------------
| 7 | 8 | 9 | 2 | 4 | 0 | 6 | 3 | 5 |
----------------------------------------------
| 0 | 1 | 3 | 6 | 5 | 8 | 2 | 0 | 9 |
----------------------------------------------
| 5 | 0 | 6 | 0 | 9 | 0 | 1 | 8 | 0 |
----------------------------------------------
| 1 | 0 | 2 | 0 | 3 | 7 | 0 | 0 | 0 |
----------------------------------------------
| 8 | 9 | 4 | 1 | 2 | 0 | 0 | 0 | 3 |
----------------------------------------------
| 0 | 3 | 7 | 0 | 0 | 9 | 0 | 1 | 2 |
----------------------------------------------
| 9 | 6 | 8 | 0 | 7 | 4 | 5 | 0 | 0 |
----------------------------------------------
| 3 | 0 | 5 | 0 | 1 | 0 | 9 | 0 | 7 |
----------------------------------------------
| 2 | 0 | 0 | 0 | 6 | 5 | 3 | 4 | 0 |
----------------------------------------------
```

Figure 10: outcome when user chooses load and there is exist a saved puzzle

Continue Input

Preconditions:

1. User has chosen either new or load
2. If new was chosen, user has chosen level of difficulty

Test steps

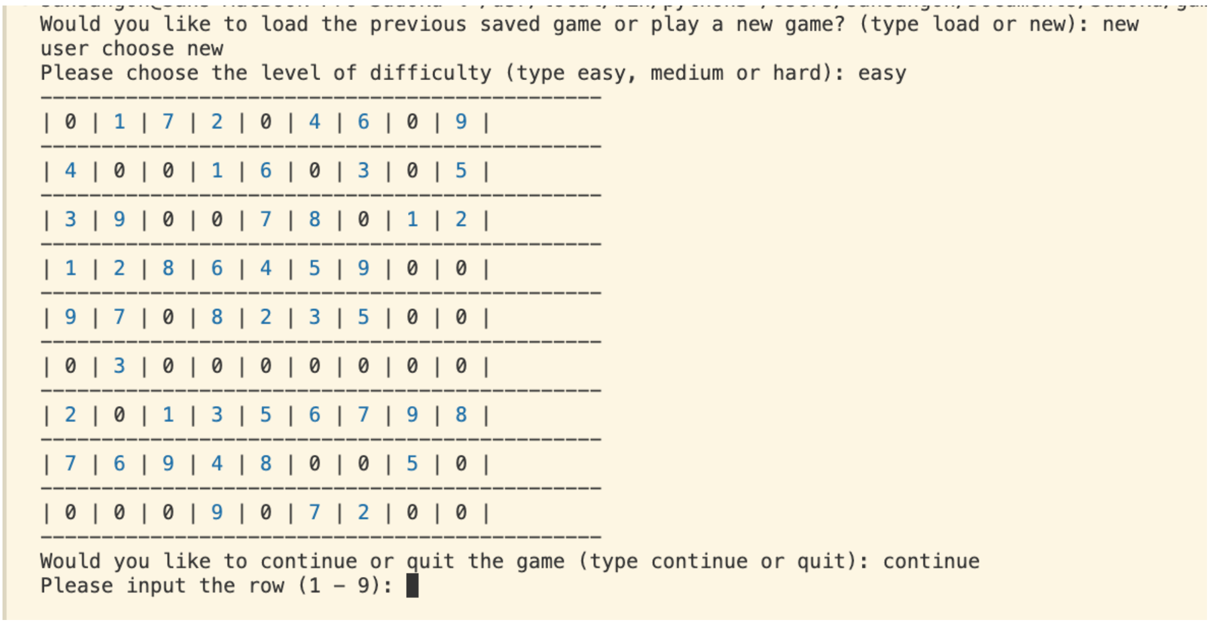
1. Input whether user would like to continue or quit the game

Expected Result:

- The program asks for row input

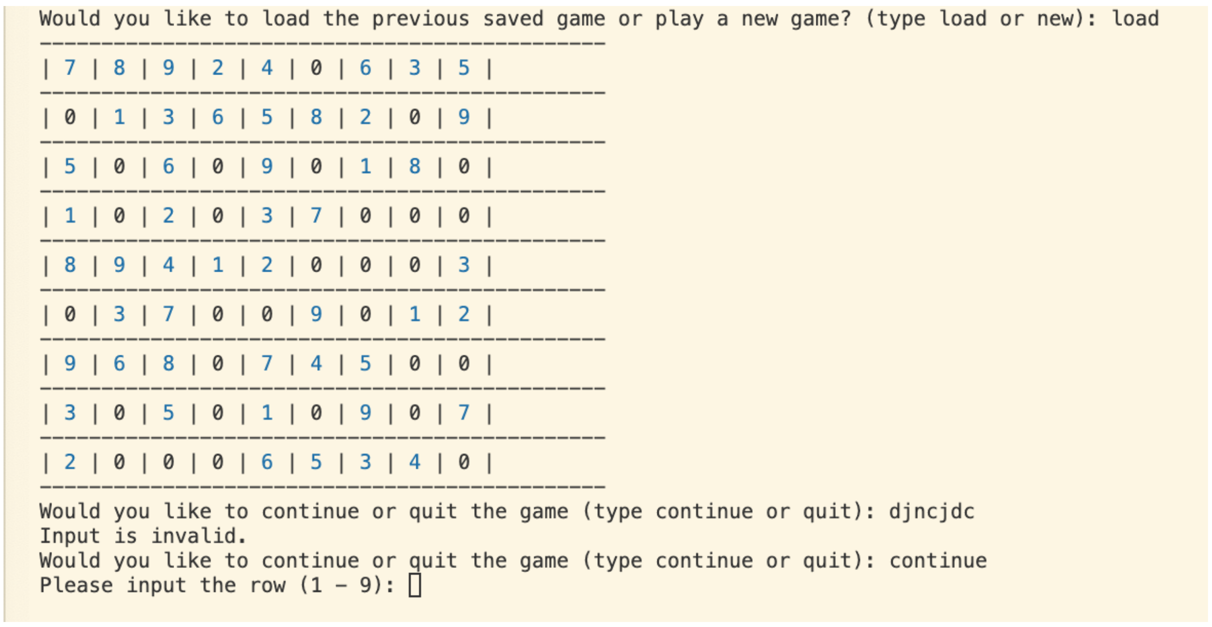
Test outcome:

- If user input 'continue' for continue or quit input, program will ask for row input
- If user input neither "continue" or "quit" for continue or quit input, program will send message that input is invalid and ask for input again



```
Would you like to load the previous saved game or play a new game? (type load or new): new
user choose new
Please choose the level of difficulty (type easy, medium or hard): easy
---------------------------------------------
| 0 | 1 | 7 | 2 | 0 | 4 | 6 | 0 | 9 |
---------------------------------------------
| 4 | 0 | 0 | 1 | 6 | 0 | 3 | 0 | 5 |
---------------------------------------------
| 3 | 9 | 0 | 0 | 7 | 8 | 0 | 1 | 2 |
---------------------------------------------
| 1 | 2 | 8 | 6 | 4 | 5 | 9 | 0 | 0 |
---------------------------------------------
| 9 | 7 | 0 | 8 | 2 | 3 | 5 | 0 | 0 |
---------------------------------------------
| 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
---------------------------------------------
| 2 | 0 | 1 | 3 | 5 | 6 | 7 | 9 | 8 |
---------------------------------------------
| 7 | 6 | 9 | 4 | 8 | 0 | 0 | 5 | 0 |
---------------------------------------------
| 0 | 0 | 0 | 9 | 0 | 7 | 2 | 0 | 0 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit): continue
Please input the row (1 - 9): 
```

Figure 11: outcome when user chooses to play new puzzle and continue playing



```
Would you like to load the previous saved game or play a new game? (type load or new): load
---------------------------------------------
| 7 | 8 | 9 | 2 | 4 | 0 | 6 | 3 | 5 |
---------------------------------------------
| 0 | 1 | 3 | 6 | 5 | 8 | 2 | 0 | 9 |
---------------------------------------------
| 5 | 0 | 6 | 0 | 9 | 0 | 1 | 8 | 0 |
---------------------------------------------
| 1 | 0 | 2 | 0 | 3 | 7 | 0 | 0 | 0 |
---------------------------------------------
| 8 | 9 | 4 | 1 | 2 | 0 | 0 | 0 | 3 |
---------------------------------------------
| 0 | 3 | 7 | 0 | 0 | 9 | 0 | 1 | 2 |
---------------------------------------------
| 9 | 6 | 8 | 0 | 7 | 4 | 5 | 0 | 0 |
---------------------------------------------
| 3 | 0 | 5 | 0 | 1 | 0 | 9 | 0 | 7 |
---------------------------------------------
| 2 | 0 | 0 | 0 | 6 | 5 | 3 | 4 | 0 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit): djncjdc
Input is invalid.
Would you like to continue or quit the game (type continue or quit): continue
Please input the row (1 - 9): 
```

Figure 12: outcome when user chooses to play the saved game and there is an invalid input

Quit and Save Puzzle Input

Preconditions:

1. User has chosen either new or load
2. If new was chosen, user has chosen level of difficulty

Test steps (quit, save)

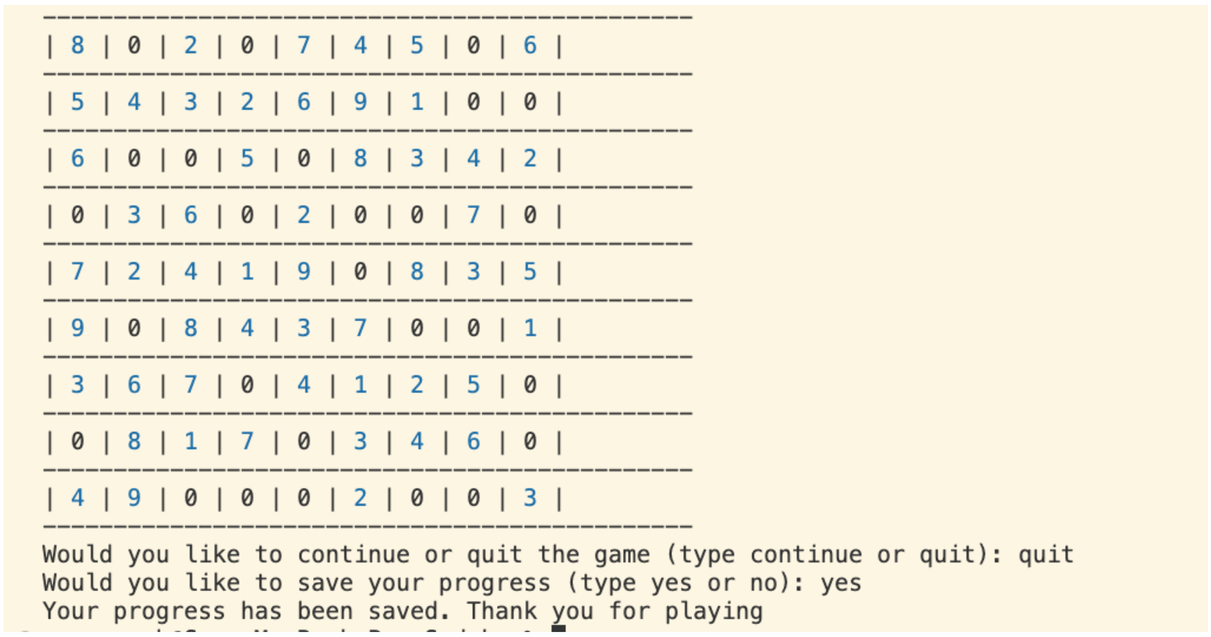
1. Input whether user would like to continue or quit the game
2. Input yes or no if the user would like to save the game

Expected Result:

- Program ends

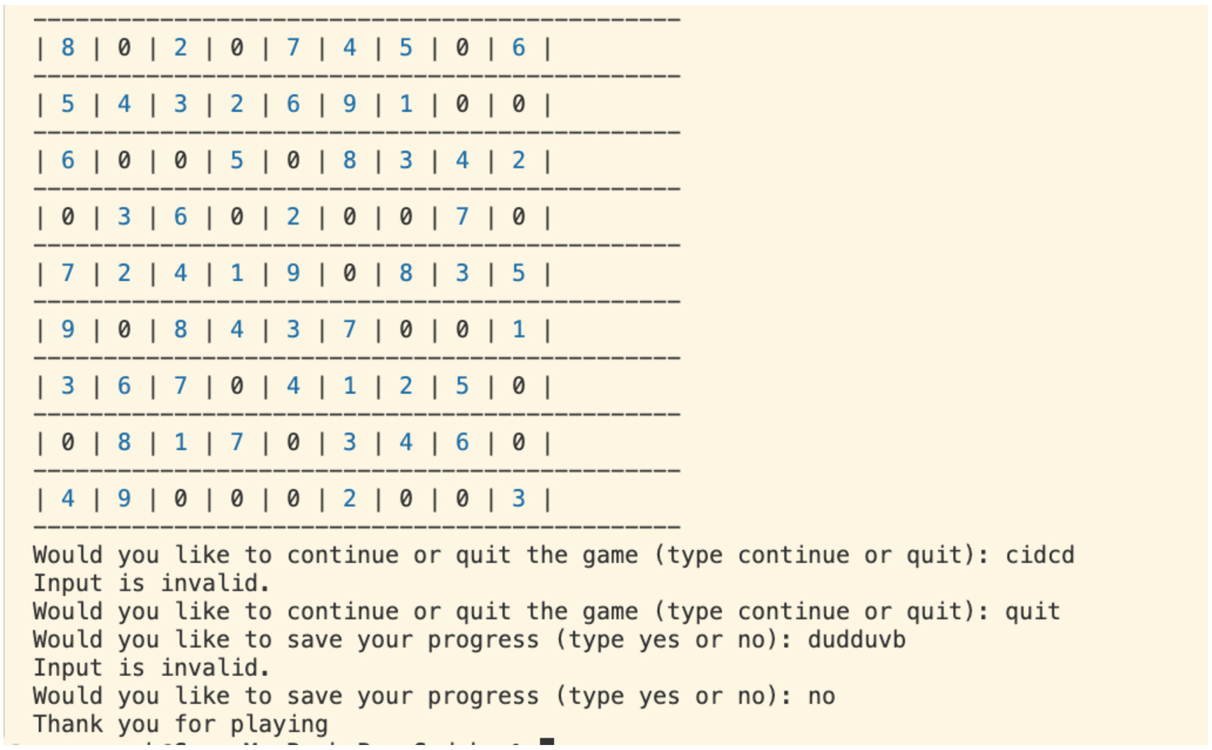
Test outcome:

- If user input “quit” for continue or quit input, program will ask for save input
- If user input neither “continue” or “quit” for continue or quit input, program will send message that input is invalid and ask for input again
- If user input “yes” for save input, program will send a message that progress has been saved, thank you for playing and game will end
- If user input “no” for save input, program will send a message saying thank you for playing and quits
- If user input neither “yes” or “no” for save input, program will send message saying that input is invalid and continues to ask for input



```
---------------------------------------------
| 8 | 0 | 2 | 0 | 7 | 4 | 5 | 0 | 6 |
---------------------------------------------
| 5 | 4 | 3 | 2 | 6 | 9 | 1 | 0 | 0 |
---------------------------------------------
| 6 | 0 | 0 | 5 | 0 | 8 | 3 | 4 | 2 |
---------------------------------------------
| 0 | 3 | 6 | 0 | 2 | 0 | 0 | 7 | 0 |
---------------------------------------------
| 7 | 2 | 4 | 1 | 9 | 0 | 8 | 3 | 5 |
---------------------------------------------
| 9 | 0 | 8 | 4 | 3 | 7 | 0 | 0 | 1 |
---------------------------------------------
| 3 | 6 | 7 | 0 | 4 | 1 | 2 | 5 | 0 |
---------------------------------------------
| 0 | 8 | 1 | 7 | 0 | 3 | 4 | 6 | 0 |
---------------------------------------------
| 4 | 9 | 0 | 0 | 0 | 2 | 0 | 0 | 3 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit): quit
Would you like to save your progress (type yes or no): yes
Your progress has been saved. Thank you for playing
```

Figure 13: outcome when inputs are valid, and user chooses to save their progress



```
 -----------------------------------------
 | 8 | 0 | 2 | 0 | 7 | 4 | 5 | 0 | 6 |
 -----------------------------------------
 | 5 | 4 | 3 | 2 | 6 | 9 | 1 | 0 | 0 |
 -----------------------------------------
 | 6 | 0 | 0 | 5 | 0 | 8 | 3 | 4 | 2 |
 -----------------------------------------
 | 0 | 3 | 6 | 0 | 2 | 0 | 0 | 7 | 0 |
 -----------------------------------------
 | 7 | 2 | 4 | 1 | 9 | 0 | 8 | 3 | 5 |
 -----------------------------------------
 | 9 | 0 | 8 | 4 | 3 | 7 | 0 | 0 | 1 |
 -----------------------------------------
 | 3 | 6 | 7 | 0 | 4 | 1 | 2 | 5 | 0 |
 -----------------------------------------
 | 0 | 8 | 1 | 7 | 0 | 3 | 4 | 6 | 0 |
 -----------------------------------------
 | 4 | 9 | 0 | 0 | 0 | 2 | 0 | 0 | 3 |
 -----------------------------------------
Would you like to continue or quit the game (type continue or quit): cidcd
Input is invalid.
Would you like to continue or quit the game (type continue or quit): quit
Would you like to save your progress (type yes or no): dudduvb
Input is invalid.
Would you like to save your progress (type yes or no): no
Thank you for playing
```

Figure 14: outcome when inputs are invalid, and user does not want to save their puzzle

Row, Column, Number Input

Preconditions:

1. User has chosen either new or load
2. If new was chosen, user has chosen level of difficulty
3. User has chosen to continue playing the game

Test steps (continue, row, column, number)

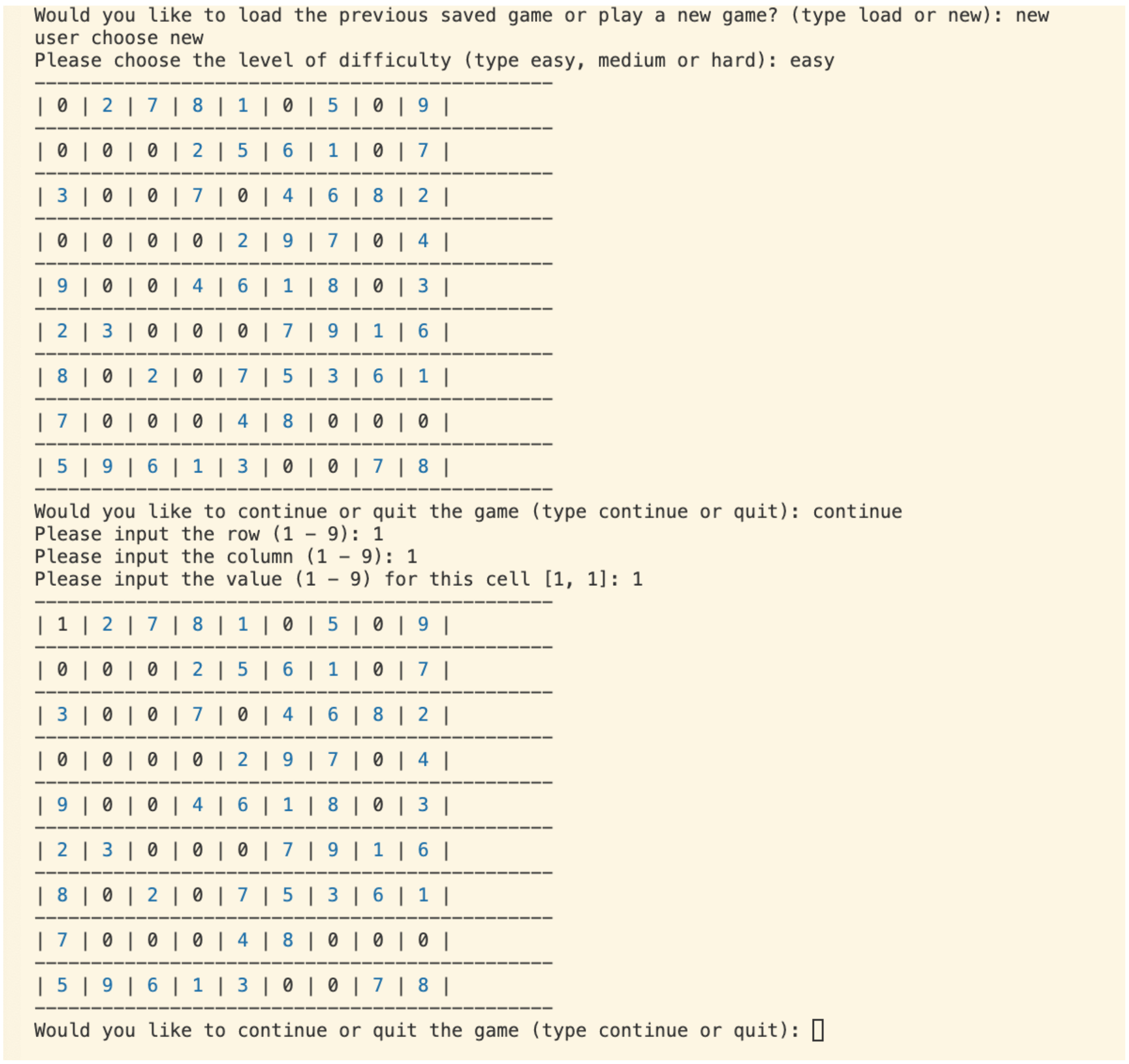
3. Input whether user would like to continue or quit the game
4. Input the row number of the cell user wishes to change
5. Input the column number of the cell user wishes to change
6. Input the number user would like the change the cell's value to

Expected Result:

- The value of the selected cell is changed

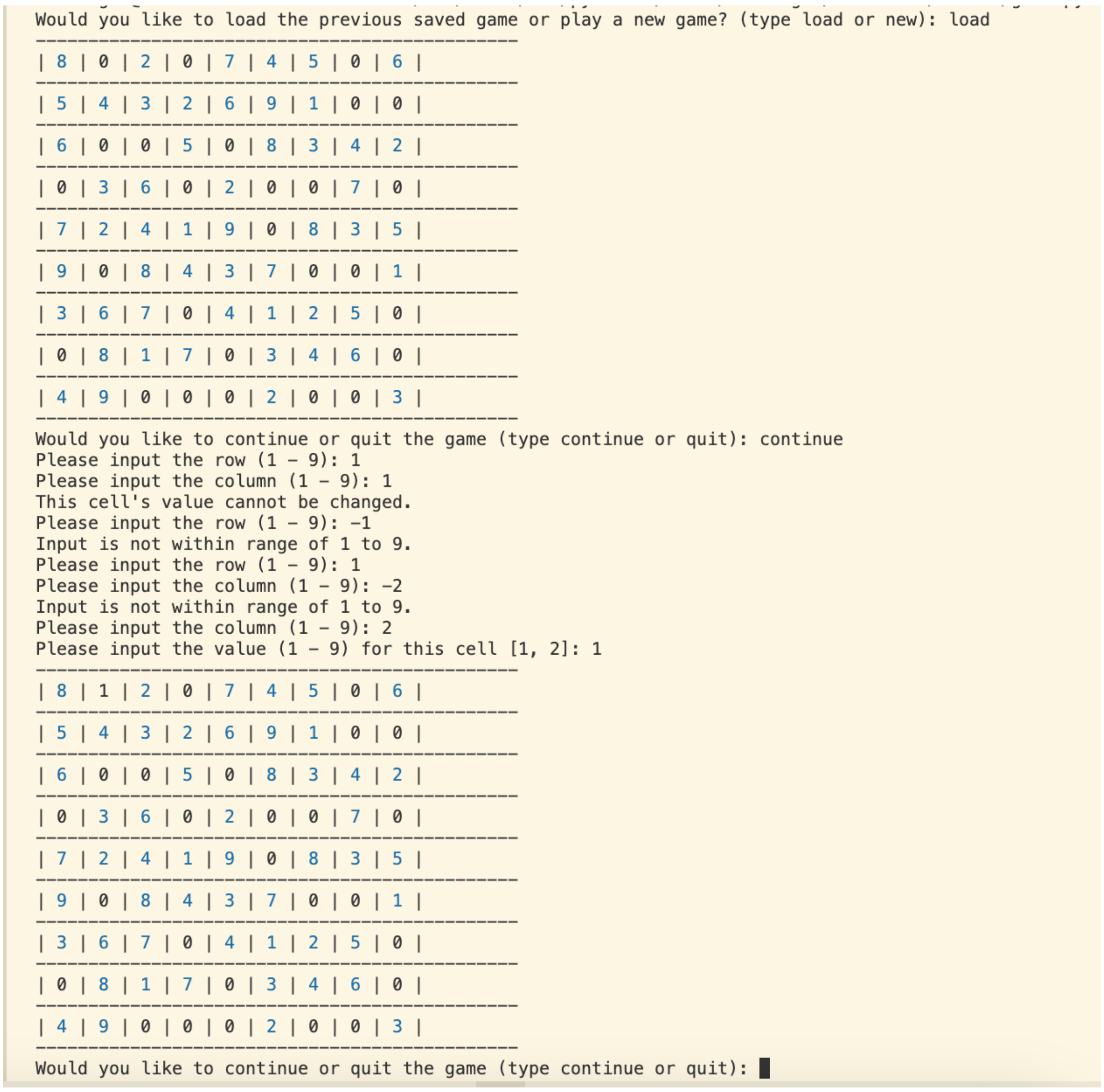
Test outcome:

- If user input a number from 1 to 9 for row input, program will ask for column input
- If user input a number from 1 to 9 for column input, program will ask for selected cell's value input
- If user input a number from 1 to 9 for selected cell's value input, program will change the value of the selected cell
- If row input, column input and selected cell's value input is not within the range of 1 to 9, program will send message that input is invalid and continue to ask for input
- If the select cell that user has chosen has a value display in blue text, program will send message that input is invalid and ask user to input the row and column again



```
Would you like to load the previous saved game or play a new game? (type load or new): new
user choose new
Please choose the level of difficulty (type easy, medium or hard): easy
-----------------------------------------
| 0 | 2 | 7 | 8 | 1 | 0 | 5 | 0 | 9 |
-----------------------------------------
| 0 | 0 | 0 | 2 | 5 | 6 | 1 | 0 | 7 |
-----------------------------------------
| 3 | 0 | 0 | 7 | 0 | 4 | 6 | 8 | 2 |
-----------------------------------------
| 0 | 0 | 0 | 0 | 2 | 9 | 7 | 0 | 4 |
-----------------------------------------
| 9 | 0 | 0 | 4 | 6 | 1 | 8 | 0 | 3 |
-----------------------------------------
| 2 | 3 | 0 | 0 | 0 | 7 | 9 | 1 | 6 |
-----------------------------------------
| 8 | 0 | 2 | 0 | 7 | 5 | 3 | 6 | 1 |
-----------------------------------------
| 7 | 0 | 0 | 0 | 4 | 8 | 0 | 0 | 0 |
-----------------------------------------
| 5 | 9 | 6 | 1 | 3 | 0 | 0 | 7 | 8 |
-----------------------------------------
Would you like to continue or quit the game (type continue or quit): continue
Please input the row (1 - 9): 1
Please input the column (1 - 9): 1
Please input the value (1 - 9) for this cell [1, 1]: 1
-----------------------------------------
| 1 | 2 | 7 | 8 | 1 | 0 | 5 | 0 | 9 |
-----------------------------------------
| 0 | 0 | 0 | 2 | 5 | 6 | 1 | 0 | 7 |
-----------------------------------------
| 3 | 0 | 0 | 7 | 0 | 4 | 6 | 8 | 2 |
-----------------------------------------
| 0 | 0 | 0 | 0 | 2 | 9 | 7 | 0 | 4 |
-----------------------------------------
| 9 | 0 | 0 | 4 | 6 | 1 | 8 | 0 | 3 |
-----------------------------------------
| 2 | 3 | 0 | 0 | 0 | 7 | 9 | 1 | 6 |
-----------------------------------------
| 8 | 0 | 2 | 0 | 7 | 5 | 3 | 6 | 1 |
-----------------------------------------
| 7 | 0 | 0 | 0 | 4 | 8 | 0 | 0 | 0 |
-----------------------------------------
| 5 | 9 | 6 | 1 | 3 | 0 | 0 | 7 | 8 |
-----------------------------------------
Would you like to continue or quit the game (type continue or quit): 
```

Figure 15: outcome when inputs are valid, value of the selected cell is changed



```
Would you like to load the previous saved game or play a new game? (type load or new): load
---------------------------------------------
| 8 | 0 | 2 | 0 | 7 | 4 | 5 | 0 | 6 |
---------------------------------------------
| 5 | 4 | 3 | 2 | 6 | 9 | 1 | 0 | 0 |
---------------------------------------------
| 6 | 0 | 0 | 5 | 0 | 8 | 3 | 4 | 2 |
---------------------------------------------
| 0 | 3 | 6 | 0 | 2 | 0 | 0 | 7 | 0 |
---------------------------------------------
| 7 | 2 | 4 | 1 | 9 | 0 | 8 | 3 | 5 |
---------------------------------------------
| 9 | 0 | 8 | 4 | 3 | 7 | 0 | 0 | 1 |
---------------------------------------------
| 3 | 6 | 7 | 0 | 4 | 1 | 2 | 5 | 0 |
---------------------------------------------
| 0 | 8 | 1 | 7 | 0 | 3 | 4 | 6 | 0 |
---------------------------------------------
| 4 | 9 | 0 | 0 | 0 | 2 | 0 | 0 | 3 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit): continue
Please input the row (1 - 9): 1
Please input the column (1 - 9): 1
This cell's value cannot be changed.
Please input the row (1 - 9): -1
Input is not within range of 1 to 9.
Please input the row (1 - 9): 1
Please input the column (1 - 9): -2
Input is not within range of 1 to 9.
Please input the column (1 - 9): 2
Please input the value (1 - 9) for this cell [1, 2]: 1
---------------------------------------------
| 8 | 1 | 2 | 0 | 7 | 4 | 5 | 0 | 6 |
---------------------------------------------
| 5 | 4 | 3 | 2 | 6 | 9 | 1 | 0 | 0 |
---------------------------------------------
| 6 | 0 | 0 | 5 | 0 | 8 | 3 | 4 | 2 |
---------------------------------------------
| 0 | 3 | 6 | 0 | 2 | 0 | 0 | 7 | 0 |
---------------------------------------------
| 7 | 2 | 4 | 1 | 9 | 0 | 8 | 3 | 5 |
---------------------------------------------
| 9 | 0 | 8 | 4 | 3 | 7 | 0 | 0 | 1 |
---------------------------------------------
| 3 | 6 | 7 | 0 | 4 | 1 | 2 | 5 | 0 |
---------------------------------------------
| 0 | 8 | 1 | 7 | 0 | 3 | 4 | 6 | 0 |
---------------------------------------------
| 4 | 9 | 0 | 0 | 0 | 2 | 0 | 0 | 3 |
---------------------------------------------
Would you like to continue or quit the game (type continue or quit):
```

Figure 16: outcome when inputs are invalid, program validates the inputs and value of selected cell is changed

Uniqueness of the grid created from random generator

Objective: run the '*generatedGrid*' function 10 times to ensure there are no repetition of the grid generated

Test steps:

1. Comment out the codes in the *main()*
2. Add this to the *main()*
```
        game.generateGrid()
        game.printGrid()
```
3. Run the program

Expected Results:

1. Fully solved grid is generated

Test outcome:

Testing Random Generator

File 1

File 1 has shown that during the sample size of 10 runs, there has been no duplication of the fully solved grid produced. Therefore, the solved grid created is unique

Uniqueness of the puzzle generated for each level of difficulty

Objective: guarantee that there is no recurrence of the puzzles created by running the program 5 times for each difficulty level

Preconditions:

1. Input new for load or new input

Tests steps:

1. Input the level of difficulty

Expect Results:

- A puzzle is generated

Test outcome:

Testing Difficulty Level

File 2

File 2 has shown that during the sample size of 30 runs in which each level was ran 5 times, there was no repetition of the puzzle generated. Hence, the puzzles developed are unique

Evaluation

Majority of the objectives was met apart from objective 6. The ones which were fulfilled were tested and had operated as planned when the program ran. Therefore, in my opinion, aside from objective 6, the remaining had performed fairly well.

The feedback was taken from users who tested the sudoku game. Some of the responses stated the game works as they had imagined and enjoy that the initial cells in the puzzle were in blue which helped them to identify which values were their inputs. However, they were not satisfied with the overall system as it has required them to input the values of the row and column and would have much preferred to have a user interface where they could have selected the cell and change the value alongside a timer to keep track to keep track of their progress.

Given additional time, I would have enhanced the '*difficultyBoard*' function to ensure that each puzzle generated has a unique solution. This modification would enable the comparison between the user input and the puzzle solution, returning True if they match and False if they differ. Furthermore, I would integrate an if statement within the for loop of the '*printGrid*' function. This condition would change the colour of the cell's number to red, indicating to the user that their input is incorrect.

Secondly, a user interface that lets user click on a cell instead of typing it would have been less time consuming, in addition, prevents the disturbance of user's concentration by removing the need to count the row and column of the cell they would like to change Furthermore, incorporating a timer would enable the user to monitor their progress for the purpose of determining whether their skills have improved.

APPENDIX

```
import random
class Sudoku:
    def __init__(self):
        self.numGrid = [[-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                     [-1,-1,-1,-1,-1,-1,-1,-1,-1]]

        self.boolGrid = [[False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,]]

        self.score = 0

    def resetGrid(self):
        self.numGrid = [[-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1],
                        [-1,-1,-1,-1,-1,-1,-1,-1,-1]]

        self.boolGrid = [[False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,],
                     [False, False, False, False, False, False, False, False,
False,]]


    def getScore(self): # gets the score variable value
        return self.score

    def printScore(self):  # Prints score variable value
        print(self.getScore())

    def incrementScore1(self):
        self.score += 1

    def printGrid(self):
        red = "\033[91m"
        blue = "\033[34m"
        reset = "\033[0m"

        for y in range(9):
            print("----------------------------------------------")
            print("| ", end = "")

            for x in range(9):
                if (self.boolGrid[y][x] == True):
                    print(blue + str(self.numGrid[y][x]) + reset + " | ", end ="")
                else:
                    print(str(self.numGrid[y][x]) + " | ", end ="")

            print("")
        print("----------------------------------------------")


    def print_boolGrid(self):
        for row in self.boolGrid:
            print("----------------------------------------------")
            print("| ", end = "")
            for element in row:
                print(str(element) + " | ", end = "")
            print("")
        print("----------------------------------------------")


    def randomGeneratorP1(self):
        self.resetGrid()

        choiceList = list(range(1,10))
        for y in range(3):
            for x in range(3):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        choiceList = list(range(1,10))
        for y in range(3,6):
            for x in range(3,6):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        choiceList = list(range(1,10))
        for y in range(6,9):
            for x in range(6,9):
                number = random.choice(choiceList)
                self.numGrid[y][x] = number
                choiceList.remove(number)

        return self.randomGeneratorP2()


    def randomGeneratorP2(self):
        for y in range(9):
            for x in range(9):
                if self.numGrid[y][x] == -1:
                    choiceList = list(range(1,10))

                    number = random.choice(choiceList)

                    if self.checkCell(number, x, y):
                        self.numGrid[y][x] = number

                        if self.solveSudoku():
                            self.randomGeneratorP2()
                            return self.numGrid

                        self.numGrid[y][x] = -1
        return False


    def cellAvailable(self): # check if there are any empty cells
        for y in range(9):
            for x in range(9):
                if ((self.numGrid[y][x] == -1) or (self.numGrid[y][x] == 0)):
                    return (y, x)
        return False


    def check3x3Grid(self, number, x, y):
        if (x < 3): # determining which 3x3 x coord the cell is in
            checkX = 0
        elif (x > 5):
            checkX = 6
        else:
            checkX = 3

        if (y < 3): # determining which 3x3 y coord the cell is in
            checkY = 0
        elif (y > 5):
            checkY = 6
        else:
            checkY = 3

        for curX in range(checkX,checkX + 3): # checking if there are any
duplication in the 3x3 cell
            for curY in range(checkY,checkY + 3):
                if self.numGrid[curY][curX] == number:
                    return False

        return True


    def checkInput(self,x, y,number): # checks to see if the inputted value meets
the requirements of the game
        if not number in range(1,10):
            return False

        for value in self.numGrid[y]:
            if value == number:
                return False

        for row in self.numGrid:
            if row[x] == number:
                return False

        if (x < 3): # determining which 3x3 x coord the cell is in
            checkX = 0
        elif (x > 5):
            checkX = 6
        else:
            checkX = 3

        if (y < 3): # determining which 3x3 y coord the cell is in
```

```
            checkY = 0
        elif (y > 5):
            checkY = 6
        else:
            checkY = 3

        for curX in range(checkX,checkX + 3): #checking if there are any 
duplication in the 3x3 cell
            for curY in range(checkY,checkY + 3):
                if self.numGrid[curY][curX] == number:
                    return False

        return True


    def checkFinalGrid(self): # checks the correctness of the grid when it has been 
filled up by user
        for y in range(9):
            for x in range(9):
                tempCell = self.numGrid[y][x]
                self.numGrid[y][x] = 0

                for value in self.numGrid[y]:
                    if value == tempCell:
                        self.numGrid[y][x] = tempCell
                        return False

                for row in self.numGrid:
                    if row[x] == tempCell:
                        self.numGrid[y][x] = tempCell
                        return False

                if (x < 3): # determining which 3x3 x coord the cell is in
                    checkX = 0
                elif (x > 5):
                    checkX = 6
                else:
                    checkX = 3

                if (y < 3): # determining which 3x3 y coord the cell is in
                    checkY = 0
                elif (y > 5):
                    checkY = 6
                else:
                    checkY = 3

                for curX in range(checkX,checkX + 3): # checking if there are any 
duplication in the 3x3 cell
                    for curY in range(checkY,checkY + 3):
                        if self.numGrid[curY][curX] == tempCell:
                            return False
```

```python
                self.numGrid[y][x] = tempCell
        return True


    def isGridFull(self): # checking to see if the grid has been
        for y in self.numGrid:
            for x in y:
                if (x == 0):
                    return False

        return True


    def checkCell(self, number, x, y): # checks to see if a number can be fitted
into a specifc space; row, col
        if ((self.numGrid[y][x] != -1) & (self.numGrid[y][x] != 0)):
        # if self.numGrid[y][x] in range(1,10):
            return False

        for value in self.numGrid[y]: # check for duplication in row
            if value == number:
                return False

        for row in self.numGrid: # check for duplication in column
            if row[x] == number:
                return False

        if (x < 3): # determining which 3x3 x coord the cell is in
            checkX = 0
        elif (x > 5):
            checkX = 6
        else:
            checkX = 3

        if (y < 3): # determining which 3x3 y coord the cell is in
            checkY = 0
        elif (y > 5):
            checkY = 6
        else:
            checkY = 3

        for curX in range(checkX,checkX + 3): # checking if there are any
duplication in the 3x3 cell
            for curY in range(checkY,checkY + 3):
                if self.numGrid[curY][curX] == number:
                    return False

        return True
    def solveSudoku(self): # solves a board using recursion
        emptyCell = self.cellAvailable()
        if not emptyCell:
            return True
        else:
            y, x = emptyCell

        for number in range(1,10):
           if self.checkCell(number, x, y):
               self.numGrid[y][x] = number

               if self.solveSudoku():
                    return self.numGrid

               self.numGrid[y][x] = -1

        return False


    def generateGrid(self):
        self.randomGeneratorP1()
        self.randomGeneratorP2()

        for y in range(9):
            for x in range(9):
                self.boolGrid[y][x] = True


    def inputNumToGrid(self, inputY, inputX, inputNum): # input the number chosen
by user into the grid
        self.numGrid[inputY - 1][inputX - 1] = inputNum
        self.printGrid()

    def userInputValue(self): # validate user input and input into the grid
        import copy
        checkquit = str(input("Would you like to continue or quit the game (type
continue or quit): "))
        while ((checkquit != "continue") and (checkquit != "quit")):
            checkquit = str(input("Input is invalid. \nWould you like to continue
or quit the game (type continue or quit): "))

        if (checkquit == "continue"):
            inputY = int(input("Please input the row (1 - 9): "))
            while (not inputY in range (1,10)):
                inputY = int(input("Input is not within range of 1 to 9. \nPlease
input the row (1 - 9): "))

            inputX = int(input("Please input the column (1 - 9): "))
            while (not inputX in range (1,10)):
                inputX = int(input("Input is not within range of 1 to 9. \nPlease
input the column (1 - 9): "))
```

```
            while (self.boolGrid[inputY - 1][inputX - 1] == True): # prevents user
from accesing the original generated grid
                print("This cell's value cannot be changed.")

                inputY = int(input("Please input the row (1 - 9): "))
                while (not inputY in range (1,10)):
                    inputY = int(input("Input is not within range of 1 to 9.
\nPlease input the row (1 - 9): "))

                inputX = int(input("Please input the column (1 - 9): "))
                while (not inputX in range (1,10)):
                    inputX = int(input("Input is not within range of 1 to 9.
\nPlease input the column (1 - 9): "))

            inputNum = int(input("Please input the value (1 - 9) for this cell [" +
str(inputY) + ", " + str(inputX) + "]: "))
            while (not inputNum in range (1,10)):
                inputNum = int(input("Input is not within range of 1 to 9. \nPlease
input the value for this cell [" + str(inputY) + ", " + str(inputX) + "](1 - 9):
"))

            self.inputNumToGrid(inputY, inputX, inputNum) # input the entered value
to cell

            if (self.checkUserInput(inputX, inputY, inputNum) == True): # if user
input follows the regulations of the game, check whether the input is the same as
the correct solution of that cell

                if (self.numGrid[inputY - 1][inputX - 1] == [inputY][inputX]):
                    self.boolGrid[inputY - 1][inputX - 1] = True
                else:
                    self.boolGrid[inputY - 1][inputX - 1] = False
            else:
                    self.boolGrid[inputY - 1][inputX - 1] = False

        else:
            checksave = str(input("Would you like to save your progress (type yes
or no): "))
            while ((checksave != "yes") & (checksave != "no")):
                checksave = str(input("Input is invalid. \nWould you like to save
your progress (type yes or no): "))

            if (checksave == "no"):
                print("Thank you for playing")
                quit()
            else:
                self.saveGame()
                print('Your progress has been saved. Thank you for playing')
                quit()
```

```
    def saveGame(self):
        saveNum = open("progressfile.txt", "w")

        grid = ""
        for y in range(9): # saves numGrid
            for x in range(9):
                if x != 8:
                    grid += str(self.numGrid[y][x]) + ","
                else:
                    grid += str(self.numGrid[y][x])
            saveNum.write(grid + "\n")
            grid = ""

        saveNum.close()

        saveBool = open("boolProgress.txt", "w")
        Bgrid = ""
        for y in range(9): # saves boolGrid
            for x in range(9):
                if x != 8:
                    Bgrid += str(self.boolGrid[y][x]) + ","
                else:
                    Bgrid += str(self.boolGrid[y][x])
            saveBool.write(Bgrid + "\n")
            Bgrid = ""

        saveBool.close()

    def loadGame(self):
        try: # checks to see if the txt fiiles exists
            loadNum = open("progressfile.txt", "r")
            LinesNum = loadNum.readlines()

            loadBool = open("boolProgress.txt", "r")
            LinesBool = loadBool.readlines()

        except:
            return False

        else:  #the text files exists
            count = 0
            for lineNum in LinesNum: # converts numGrid back to a 2d array and the
values back to an integer
                listN = lineNum.split(",")

                self.numGrid[count] = [int(x) for x in listN]
                count += 1

            loadNum.close()

            count = 0
            for lineBool in LinesBool: # converts boolGrid back to a 2d array and
value back to boolean
                listB = lineBool.split(",")

                self.boolGrid[count] = [True if y.strip() == "True" else False for
y in listB]
                count += 1

            loadBool.close()
            return True


    def checkUserInput(self, inputX, inputY, inputNum):
        if (self.checkInput(inputX - 1,inputY - 1,inputNum) == True): # if there
are no duplication, number is placed in the grid
            self.numGrid[inputY - 1][inputX - 1] = inputNum
            return True
        else:
            return False


    def difficultyBoard(self, difficulty):
        import copy
        self.generateGrid()

        numGrid_copy = copy.deepcopy(self.numGrid)

        if difficulty == "easy":
            cellsToRemove = random.randint(36,43)
        elif difficulty == "medium":
            cellsToRemove = random.randint(45,54)
        elif difficulty == "hard":
            cellsToRemove = random.randint(55,62)
        else:
            return

        for i in range(cellsToRemove):
            y = random.randint(0,8)
            x = random.randint(0,8)

            if (self.numGrid[y][x] != -1):
                self.numGrid[y][x] = -1 # remove a random cell
            else:
                i = i - 1
                continue

            workingCopy = copy.deepcopy(self.numGrid)  # make a copy of the grid
with the removed cell
```

```
            if (self.solveSudoku() == False): # no posssible soluton with the
removed cell
                self.numGrid = copy.deepcopy(workingCopy) # replaced the solved
grid with the removed cell grid

                self.numGrid[y][x] = numGrid_copy[y][x] # assign the removed cell
back to the original value
                i = i - 1
            else:
                self.numGrid = copy.deepcopy(workingCopy) # the removed cell is
placed back into the grid

        for y in range(9):
            for x in range(9):
                if (self.numGrid[y][x] == -1): # the removed cells is displayed 0
instead of -1
                    self.numGrid[y][x] = 0
                    self.boolGrid[y][x] = False
                else:
                    self.boolGrid[y][x] = True # this will be the number that is
displayed when generated


def main():
    import copy
    game = Sudoku()

    userInput = str(input("Would you like to load the previous saved game or play a
new game? (type load or new): "))
    while ((userInput != "load") & (userInput != "new")):
        userInput = str(input("Input was invalid. \nWould you like to load the
previous saved game or play a new game? (type load or new): "))

    if (userInput == "load"):
        if (game.loadGame() == False):
            print("Previous saved game does not exist")
            print("A new game will be generated")

            userDifficulty = str(input("Please choose the level of difficulty (type
easy, medium or hard): "))
            while ((userDifficulty != "easy") & (userDifficulty != "medium") &
(userDifficulty != "hard")):
                userDifficulty = str(input("Input was invalid. \nPlease choose the
level of difficulty (type easy, medium or hard): "))

            game.difficultyBoard(userDifficulty)
            game.printGrid()

            while True:
                game.userInputValue()
                if (game.isGridFull() == True):
                    print("grid full: ")
                    print (game.isGridFull())

                    print ("checkfinalgrid: ")
                    print(game.checkFinalGrid())

                    if (game.checkFinalGrid() == True):
                        break

            print('Game Over. Thank you for playing ')

        else: # there exist a game in the text file
            game.loadGame()
            game.printGrid()

            while True:

                game.userInputValue()

                if (game.isGridFull() == True):
                    print("grid full: ")
                    print (game.isGridFull())

                    print ("checkfinalgrid: ")
                    print(game.checkFinalGrid())

                    if (game.checkFinalGrid() == True):
                        break

            print('Game Over. Thank you for playing ')


    else: # user wants to play a new game
        print("user choose new")
        userDifficulty = str(input("Please choose the level of difficulty (type 
easy, medium or hard): "))
        while ((userDifficulty != "easy") & (userDifficulty != "medium") & 
(userDifficulty != "hard")):
            userDifficulty = str(input("Input was invalid. \nPlease choose the 
level of difficulty (type easy, medium or hard): "))
        game.difficultyBoard(userDifficulty)
        game.printGrid()

        while True:
            game.userInputValue()

            if (game.isGridFull() == True):
                print("gri full: ")
                print (game.isGridFull())
```

```python
                print ("checkfinalgri: ")
                print(game.checkFinalGrid())

                if (game.checkFinalGrid() == True):
                    break

        print('Game Over. Thank you for playing ')


if __name__ == "__main__":
    main()
```