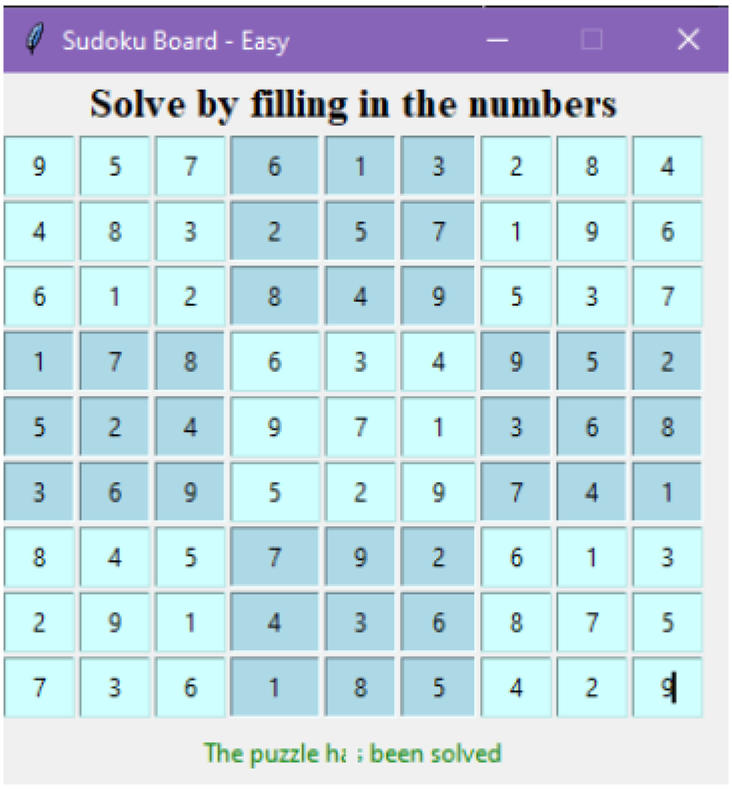
SUDOKU GENERATOR

Networking & Cybersecurity

Phebe Le

Progress Report

Session 1.5: 6/5/2023-23/5/23

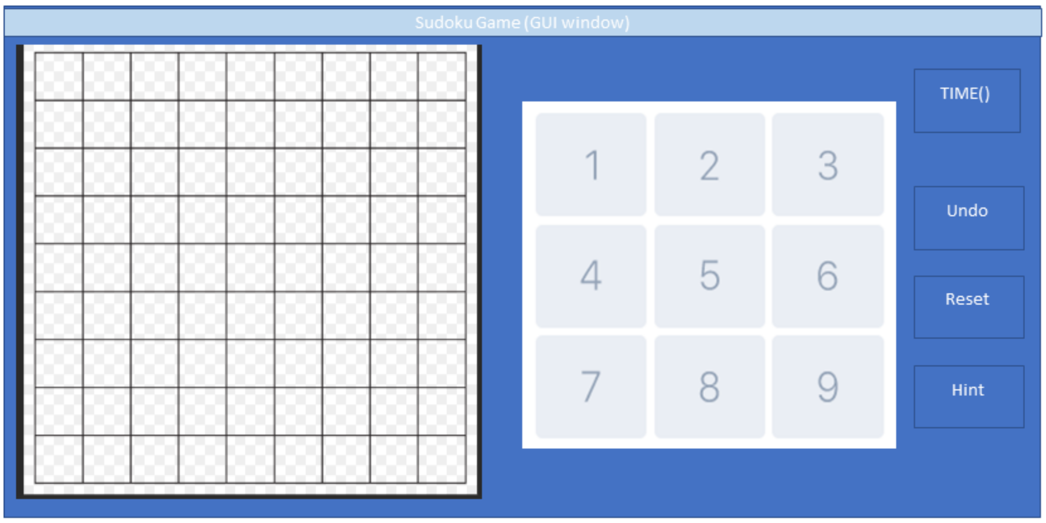


Solve by filling in the numbers

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

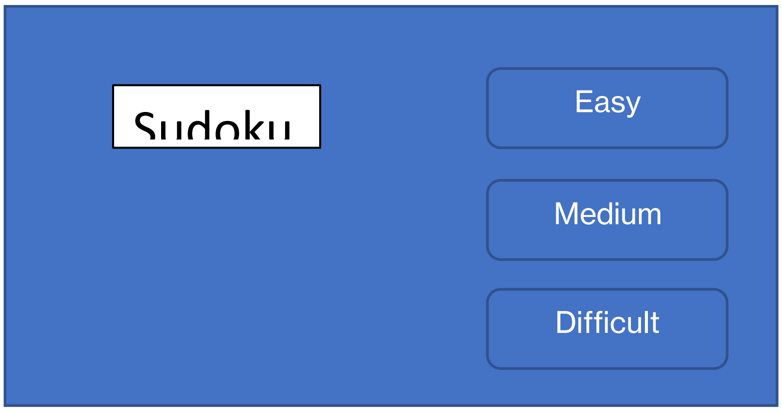
The puzzle h: ; been solved

1. GUI Window



Aim: For the GUI window, I intend to segregate the GUI window into three: the sudoku board, the number pad and the 4 button/widgets. The number pad will input a number into the given square. The time widget will begin once a user begins inputting numbers, the undo button will undo the previous action, the reset will reset to the beginning puzzle the user begun with. The hint button will input one number into the sudoku puzzle.

2. Menu



Aim: The menu will be created on the 'main screen' of the GUI window. This will then lead into the different sudoku modes and randomised designs. The easy mode

3. Sudoku square

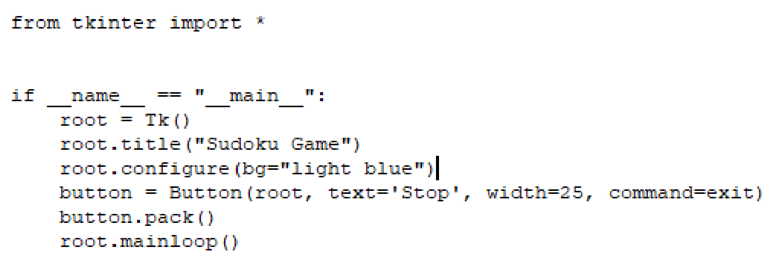
To create the sudoku code, the use of a list will be used as a prototype. This will initially be a list with 81 numbers to replicate a 9x9 square.

11/6/23

Aim: The aim will be to create the main template for the design. This being the menu with the different level buttons.

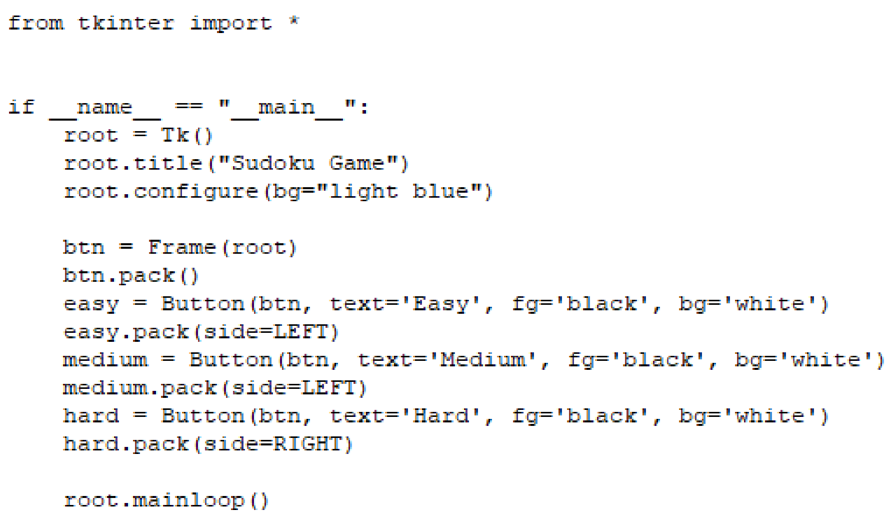
Implementation:

The first implementation was creating the main screen by using the '__main__' function and setting up the GUI screen, and main elements.



```
from tkinter import *


if __name__ == "__main__":
    root = Tk()
    root.title("Sudoku Game")
    root.configure(bg="light blue")
    button = Button(root, text='Stop', width=25, command=exit)
    button.pack()
    root.mainloop()
```

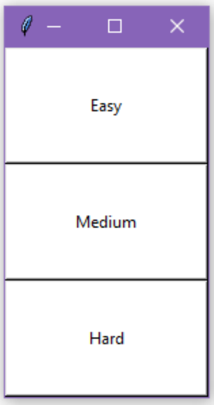
The second implementation is creating the button for the levels of difficulty and position it using a button pack. This may be changed in the future to suit other elements.



```
from tkinter import *


if __name__ == "__main__":
    root = Tk()
    root.title("Sudoku Game")
    root.configure(bg="light blue")

    btn = Frame(root)
    btn.pack()
    easy = Button(btn, text='Easy', fg='black', bg='white')
    easy.pack(side=LEFT)
    medium = Button(btn, text='Medium', fg='black', bg='white')
    medium.pack(side=LEFT)
    hard = Button(btn, text='Hard', fg='black', bg='white')
    hard.pack(side=RIGHT)

    root.mainloop()
```

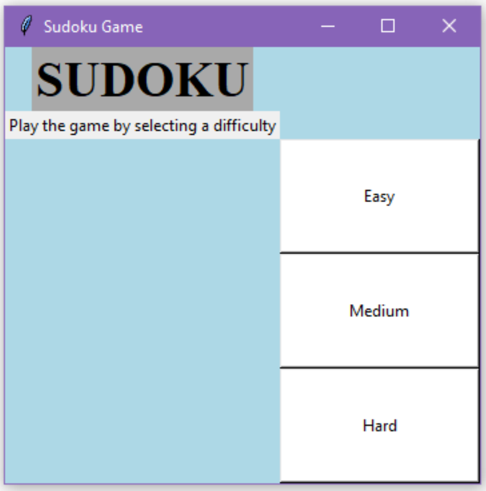
The third implementation, is eradicating the design style of the buttons to create a stacked button appearance. This is created using the row and column method.

```
from tkinter import *


if __name__ == "__main__":
    # Step 1.1
    root = Tk()
    root.title("Sudoku Game")
    root.configure(bg="light blue")

    # Step 1.2

    easy = Button(root, text='Easy', width = 20, height=5, fg='black', bg='white')
    medium = Button(root, text='Medium', width = 20, height=5,  fg='black', bg='white')
    hard = Button(root, text='Hard',width = 20, height=5, fg='black', bg='white')


    # Step 1.3

    easy.grid(row = 3, column = 2)
    medium.grid(row = 4, column = 2)
    hard.grid(row = 6, column = 2)
    root.mainloop()
```

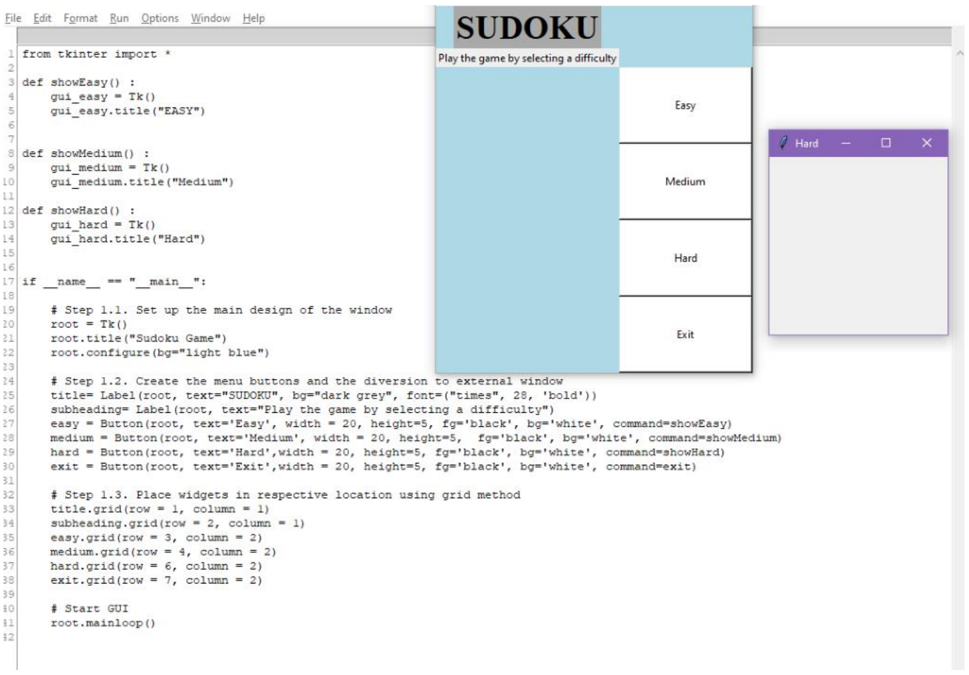


The fourth element is the further arrangement of the title, description and the buttons

```
from tkinter import *


if __name__ == "__main__":
    # Step 1.1
    root = Tk()
    root.title("Sudoku Game")
    root.configure(bg="light blue")

    # Step 1.2

    title= Label(root, text="SUDOKU", bg="dark grey", font=("times", 28, 'bold'))
    subheading= Label(root, text="Play the game by selecting a difficulty")
    easy = Button(root, text='Easy', width = 20, height=5, fg='black', bg='white')
    medium = Button(root, text='Medium', width = 20, height=5,  fg='black', bg='white')
    hard = Button(root, text='Hard',width = 20, height=5, fg='black', bg='white')


    # Step 1.3
    title.grid(row = 1, column = 1)
    subheading.grid(row = 2, column = 1)
    easy.grid(row = 3, column = 2)
    medium.grid(row = 4, column = 2)
    hard.grid(row = 6, column = 2)
    root.mainloop()
```

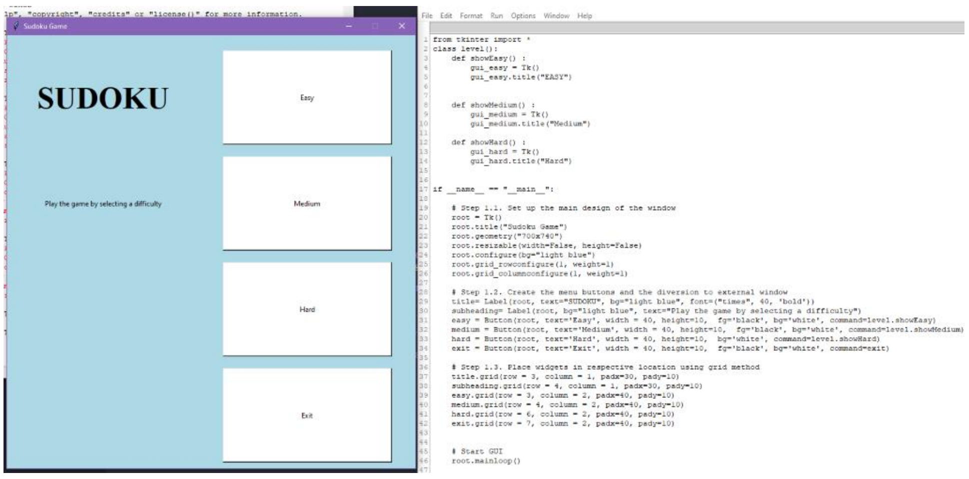


The implementation is testing the functionality of the buttons. This being if it creates a GUI window for the implementation of the sudoku grid. This showed to be successful as seen by the GUI window named ‘Hard’ when the ‘Hard’ button is pressed.


File Edit Format Run Options Window Help
SUDOKU
Play the game by selecting a difficulty
Easy
Medium
Hard
Exit
Hard

```python
from tkinter import *

def showEasy() :
    gui_easy = Tk()
    gui_easy.title("EASY")


def showMedium() :
    gui_medium = Tk()
    gui_medium.title("Medium")

def showHard() :
    gui_hard = Tk()
    gui_hard.title("Hard")


if __name__ == "__main__":

    # Step 1.1. Set up the main design of the window
    root = Tk()
    root.title("Sudoku Game")
    root.configure(bg="light blue")

    # Step 1.2. Create the menu buttons and the diversion to external window
    title= Label(root, text="SUDOKU", bg="dark grey", font=("times", 28, 'bold'))
    subheading= Label(root, text="Play the game by selecting a difficulty")
    easy = Button(root, text='Easy', width = 20, height=5, fg='black', bg='white', command=showEasy)
    medium = Button(root, text='Medium', width = 20, height=5,  fg='black', bg='white', command=showMedium)
    hard = Button(root, text='Hard',width = 20, height=5, fg='black', bg='white', command=showHard)
    exit = Button(root, text='Exit',width = 20, height=5, fg='black', bg='white', command=exit)

    # Step 1.3. Place widgets in respective location using grid method
    title.grid(row = 1, column = 1)
    subheading.grid(row = 2, column = 1)
    easy.grid(row = 3, column = 2)
    medium.grid(row = 4, column = 2)
    hard.grid(row = 6, column = 2)
    exit.grid(row = 7, column = 2)

    # Start GUI
    root.mainloop()
```

The next implementation is arranging the board to suite the design.

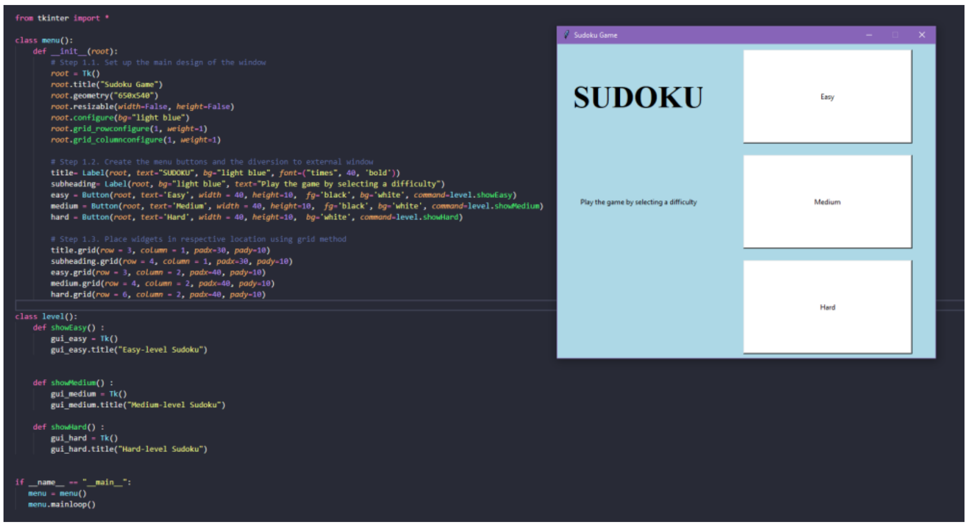

lp", "copyright", "credits" or "license()" for more information.
Sudoku Game
SUDOKU
Play the game by selecting a difficulty
Easy
Medium
Hard
Exit
File Edit Format Run Options Window Help

```python
from tkinter import *
class level():
    def showEasy() :
        gui_easy = Tk()
        gui_easy.title("EASY")


    def showMedium() :
        gui_medium = Tk()
        gui_medium.title("Medium")

    def showHard() :
        gui_hard = Tk()
        gui_hard.title("Hard")


if __name__ == "__main__":

    # Step 1.1. Set up the main design of the window
    root = Tk()
    root.title("Sudoku Game")
    root.geometry("700x740")
    root.resizable(width=False, height=False)
    root.configure(bg="light blue")
    root.grid_rowconfigure(1, weight=1)
    root.grid_columnconfigure(1, weight=1)

    # Step 1.2. Create the menu buttons and the diversion to external window
    title= Label(root, text="SUDOKU", bg="light blue", font=("times", 40, 'bold'))
    subheading= Label(root, bg="light blue", text="Play the game by selecting a difficulty")
    easy = Button(root, text='Easy', width = 40, height=10,  fg='black', bg='white', command=level.showEasy)
    medium = Button(root, text='Medium', width = 40, height=10,  fg='black', bg='white', command=level.showMedium)
    hard = Button(root, text='Hard', width = 40, height=10,  bg='white', command=level.showHard)
    exit = Button(root, text='Exit', width = 40, height=10,  fg='black', bg='white', command=exit)

    # Step 1.3. Place widgets in respective location using grid method
    title.grid(row = 3, column = 1, padx=30, pady=10)
    subheading.grid(row = 4, column = 1, padx=30, pady=10)
    easy.grid(row = 3, column = 2, padx=40, pady=10)
    medium.grid(row = 4, column = 2, padx=40, pady=10)
    hard.grid(row = 6, column = 2, padx=40, pady=10)
    exit.grid(row = 7, column = 2, padx=40, pady=10)


    # Start GUI
    root.mainloop()
```

18.6.23

Aim: The main aim will be to segregate the main code into modules through classes and create the board. The board itself will required to be randomly generated and allow users to input only numbers

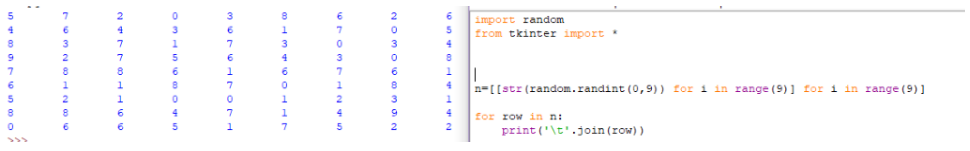
As seen by the image on the left, the previous design of the 'menu' has been diverted to a class named 'menu'. This helps to free the main looping function.



```python
from tkinter import *

class menu():
    def __init__(root):
        # Step 1.1. Set up the main design of the window
        root = Tk()
        root.title("Sudoku Game")
        root.geometry("650x540")
        root.resizable(width=False, height=False)
        root.configure(bg="light blue")
        root.grid_rowconfigure(1, weight=1)
        root.grid_columnconfigure(1, weight=1)

        # Step 1.2. Create the menu buttons and the diversion to external window
        title= Label(root, text="SUDOKU", bg="light blue", font=("times", 40, 'bold'))
        subheading= Label(root, bg="light blue", text="Play the game by selecting a difficulty")
        easy = Button(root, text='Easy', width = 40, height=10,  fg='black', bg='white', command=level.showEasy)
        medium = Button(root, text='Medium', width = 40, height=10,  fg='black', bg='white', command=level.showMedium)
        hard = Button(root, text='Hard', width = 40, height=10,  bg='white', command=level.showHard)

        # Step 1.3. Place widgets in respective location using grid method
        title.grid(row = 3, column = 1, padx=30, pady=10)
        subheading.grid(row = 4, column = 1, padx=30, pady=10)
        easy.grid(row = 3, column = 2, padx=40, pady=10)
        medium.grid(row = 4, column = 2, padx=40, pady=10)
        hard.grid(row = 6, column = 2, padx=40, pady=10)

class level():
    def showEasy() :
        gui_easy = Tk()
        gui_easy.title("Easy-level Sudoku")


    def showMedium() :
        gui_medium = Tk()
        gui_medium.title("Medium-level Sudoku")

    def showHard() :
        gui_hard = Tk()
        gui_hard.title("Hard-level Sudoku")


if __name__ == "__main__":
   menu = menu()
   menu.mainloop()
```

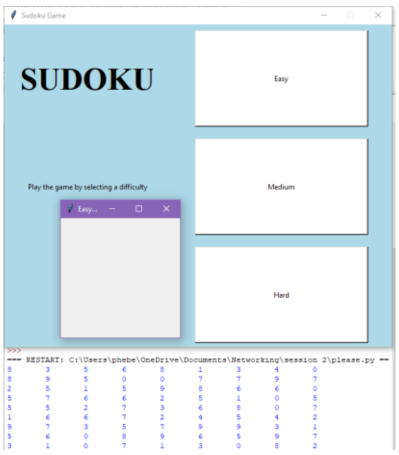
Implementation 1:

To create the board that was randomly generates number, modular testing will be conducted. This was created through a variable named 'n' (for number). With the intention of optimising the lines for this modular testing. The print function below, '/t' was used to separate the lines after n was performed (creates first row), yet '.join' combines it into one string. Whilst this successfully created 81 random number in a 9x9 format, it did not adhere to the goal of allowing users to input numbers. As a result, further investigation must be conducted. The first image below shows the initial test in a segregated workspace. The second being the combining this code into the main code.



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

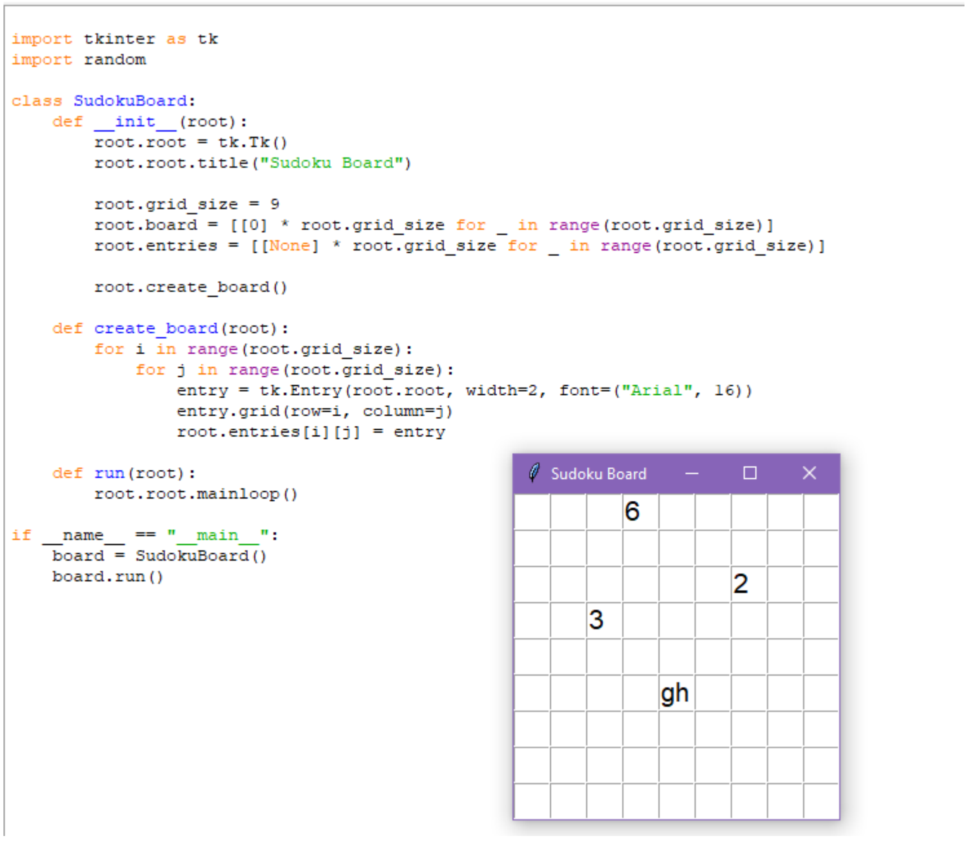
```python
import random
from tkinter import *


n=[[str(random.randint(0,9)) for i in range(9)] for i in range(9)]

for row in n:
    print('\t'.join(row))
```



```
>>>
=== RESTART: C:\Users\phebe\OneDrive\Documents\Networking\session 2\please.py ==
8	3	5	6	8	1	3	4	0
8	9	5	0	0	7	7	9	7
2	5	1	5	9	8	6	6	0
5	7	6	6	2	5	1	0	5
5	5	2	7	3	6	8	0	7
1	6	6	7	2	4	5	4	2
9	7	3	5	7	9	9	3	1
5	6	0	8	9	6	5	9	7
3	1	0	7	1	3	0	8	2
```

Implementation 2:

To create the board, a separate file will be created for modular testing of the board. Based off examples which have been found on stack overflow and other external websites, the main recommendation is to use the ‘tk’ library for the creation of a board that can supply inputted numbers. This is due to tk being recognised as the official tkinter module whilst * is mainly used when referencing one or two elements. As a result, this code below was created.

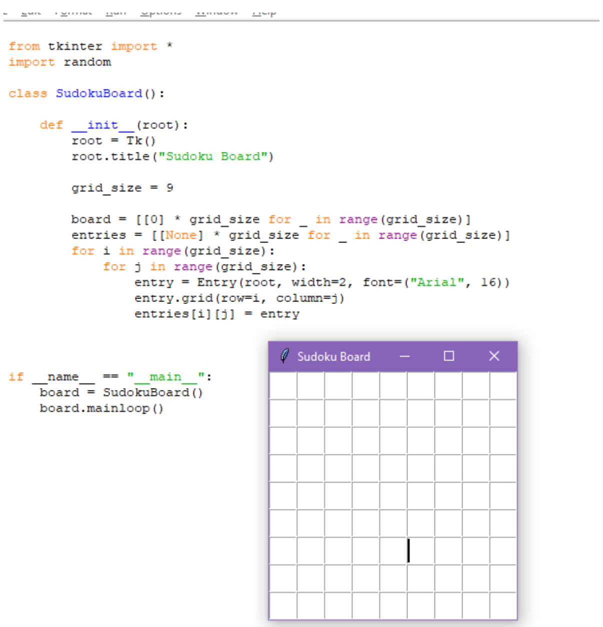


```
import tkinter as tk
import random

class SudokuBoard:
    def __init__(root):
        root.root = tk.Tk()
        root.root.title("Sudoku Board")

        root.grid_size = 9
        root.board = [[0] * root.grid_size for _ in range(root.grid_size)]
        root.entries = [[None] * root.grid_size for _ in range(root.grid_size)]

        root.create_board()

    def create_board(root):
        for i in range(root.grid_size):
            for j in range(root.grid_size):
                entry = tk.Entry(root.root, width=2, font=("Arial", 16))
                entry.grid(row=i, column=j)
                root.entries[i][j] = entry

    def run(root):
        root.root.mainloop()

if __name__ == "__main__":
    board = SudokuBoard()
    board.run()
```

As seen by the results in the board, this code was successful in creating the board, yet it was not limited to one number or only numbers. Alongside this, the modular testing used the tk library is not used in the main code. As a result, the next steps are to create limitations and convert the code to the * module.

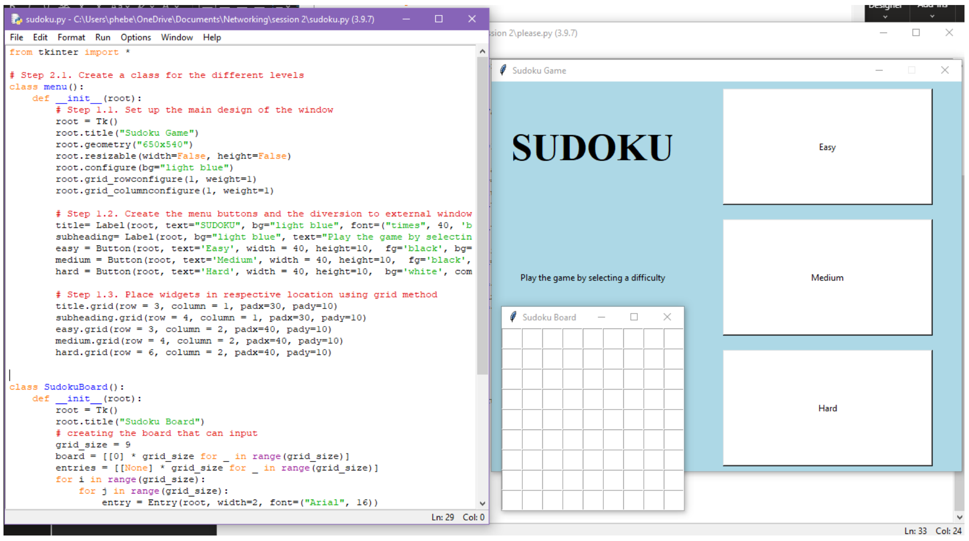
Implementation 3:

The next implementation was the conversion of the tk module to the * module. This showed to be successful.



```python
from tkinter import *
import random

class SudokuBoard():

    def __init__(root):
        root = Tk()
        root.title("Sudoku Board")

        grid_size = 9

        board = [[0] * grid_size for _ in range(grid_size)]
        entries = [[None] * grid_size for _ in range(grid_size)]
        for i in range(grid_size):
            for j in range(grid_size):
                entry = Entry(root, width=2, font=("Arial", 16))
                entry.grid(row=i, column=j)
                entries[i][j] = entry


if __name__ == "__main__":
    board = SudokuBoard()
    board.mainloop()
```

This code was then added the main code, which showed to be successful.



sudoku.py - C:\Users\phebe\OneDrive\Documents\Networking\session 2\sudoku.py (3.9.7)
File Edit Format Run Options Window Help
Sudoku Game
SUDOKU
Play the game by selecting a difficulty
Easy
Medium
Hard
Sudoku Board
Ln: 29 Col: 0
Ln: 33 Col: 24

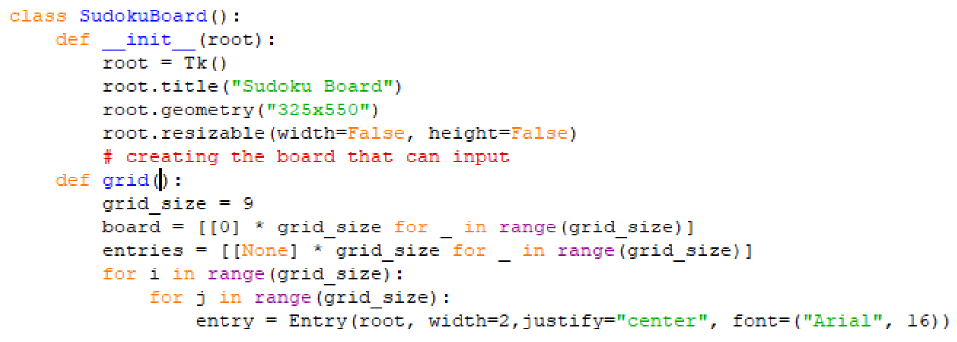
```python
from tkinter import *

# Step 2.1. Create a class for the different levels
class menu():
    def __init__(root):
        # Step 1.1. Set up the main design of the window
        root = Tk()
        root.title("Sudoku Game")
        root.geometry("650x540")
        root.resizable(width=False, height=False)
        root.configure(bg="light blue")
        root.grid_rowconfigure(1, weight=1)
        root.grid_columnconfigure(1, weight=1)

        # Step 1.2. Create the menu buttons and the diversion to external window
        title= Label(root, text="SUDOKU", bg="light blue", font=("times", 40, 'b
        subheading= Label(root, bg="light blue", text="Play the game by selectin
        easy = Button(root, text='Easy', width = 40, height=10,  fg='black', bg=
        medium = Button(root, text='Medium', width = 40, height=10,  fg='black',
        hard = Button(root, text='Hard', width = 40, height=10,  bg='white', com

        # Step 1.3. Place widgets in respective location using grid method
        title.grid(row = 3, column = 1, padx=30, pady=10)
        subheading.grid(row = 4, column = 1, padx=30, pady=10)
        easy.grid(row = 3, column = 2, padx=40, pady=10)
        medium.grid(row = 4, column = 2, padx=40, pady=10)
        hard.grid(row = 6, column = 2, padx=40, pady=10)


class SudokuBoard():
    def __init__(root):
        root = Tk()
        root.title("Sudoku Board")
        # creating the board that can input
        grid_size = 9
        board = [[0] * grid_size for _ in range(grid_size)]
        entries = [[None] * grid_size for _ in range(grid_size)]
        for i in range(grid_size):
            for j in range(grid_size):
                entry = Entry(root, width=2, font=("Arial", 16))
```

19/6/23-20/3/23

Aim: To create the limitations of the Sudoku board (less than 2 numbers and only numbers)

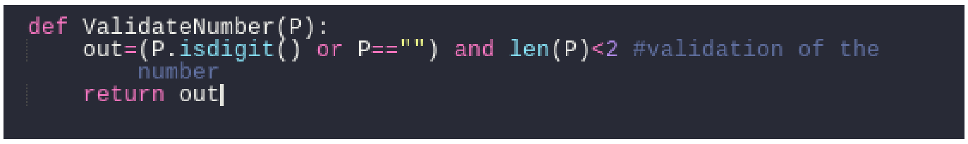
Implementation:

This was initially started by rearranging the design to create a method for the Sudoku board.



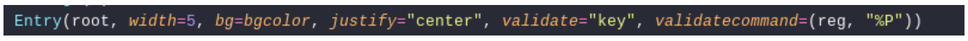
```
class SudokuBoard():
    def __init__(root):
        root = Tk()
        root.title("Sudoku Board")
        root.geometry("325x550")
        root.resizable(width=False, height=False)
        # creating the board that can input
    def grid():
        grid_size = 9
        board = [[0] * grid_size for _ in range(grid_size)]
        entries = [[None] * grid_size for _ in range(grid_size)]
        for i in range(grid_size):
            for j in range(grid_size):
                entry = Entry(root, width=2,justify="center", font=("Arial", 16))
```

This was successful and allowed the same events to occur. Now the main aim is to create the limitations of the sudoku board. This will be created through a function named validateNumber() where a parameter named P will be used check if the variable is a digit, letter or more than one character. This validation will be checked for each square on the grid.



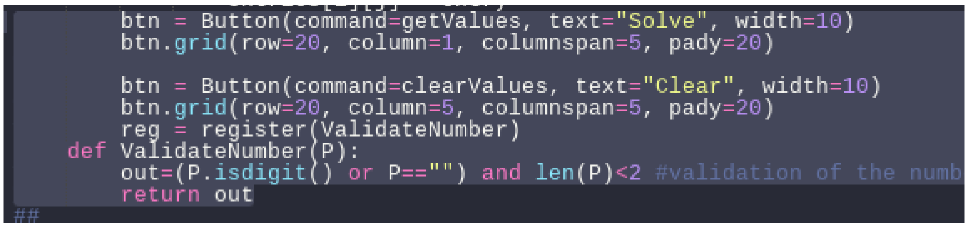
```
def ValidateNumber(P):
    out=(P.isdigit() or P=="") and len(P)<2 #validation of the
        number
    return out
```

This was used within the entry function.

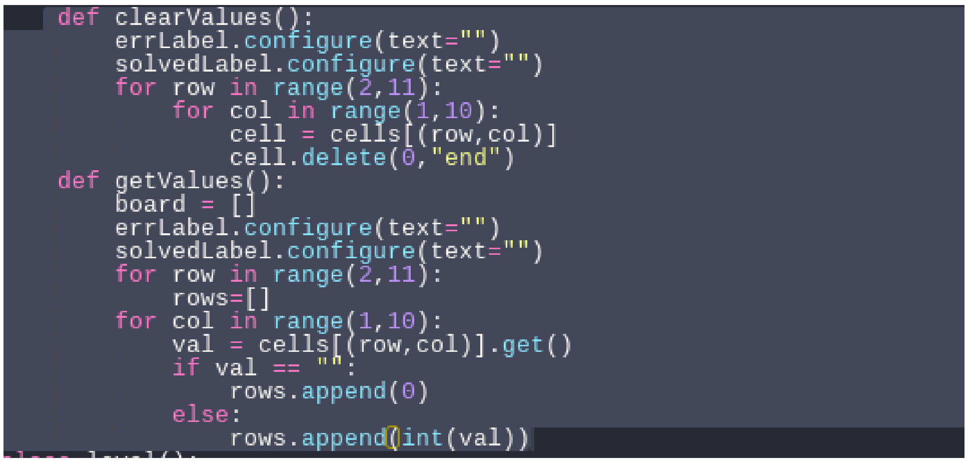


```
Entry(root, width=5, bg=bgcolor, justify="center", validate="key", validatecommand=(reg, "%P"))
```

This failed to work. As result, this issue was deprioritized and the issue of creating a clear button, solve button and the shuffle button was focused upon. Through doing this, I found issues with the placement of the board where the board could not be moved, centred in its whole. This meant that when a button was placed in a row below the board in a column that aligned with the sudoku board, the sudoku board split in half to accommodate the board. This issue meant that the board was not viewed as a complete entity but as a several rows and column that fit each other.



```
        btn = Button(command=getValues, text="Solve", width=10)
        btn.grid(row=20, column=1, columnspan=5, pady=20)

        btn = Button(command=clearValues, text="Clear", width=10)
        btn.grid(row=20, column=5, columnspan=5, pady=20)
        reg = register(ValidateNumber)
    def ValidateNumber(P):
        out=(P.isdigit() or P=="") and len(P)<2 #validation of the numb
        return out
##
```



```
def clearValues():
    errLabel.configure(text="")
    solvedLabel.configure(text="")
    for row in range(2,11):
        for col in range(1,10):
            cell = cells[(row,col)]
            cell.delete(0,"end")
def getValues():
    board = []
    errLabel.configure(text="")
    solvedLabel.configure(text="")
    for row in range(2,11):
        rows=[]
    for col in range(1,10):
        val = cells[(row,col)].get()
        if val == "":
            rows.append(0)
        else:
            rows.append(int(val))
```

Due to these errors, I opted to change the entire design to the tk module. This decision was rash and may lead to much more errors. But this decision was also based on optimising efficiency. This is due to my current inefficient method of writing in a separate text editor ('Visual Studio Code') and copying it into the tkinter code compiler as a result of Visual Studio Code not supporting the * module.

this issue was ignored for the time being to include buttons and edit the window. When adding different testing was conducted and a found errors with the sudoku board in terms of placement

I also opted to change the code to the tk module. This is because I find this module difficult to edit in the text editor 'Visual Studio Code'.

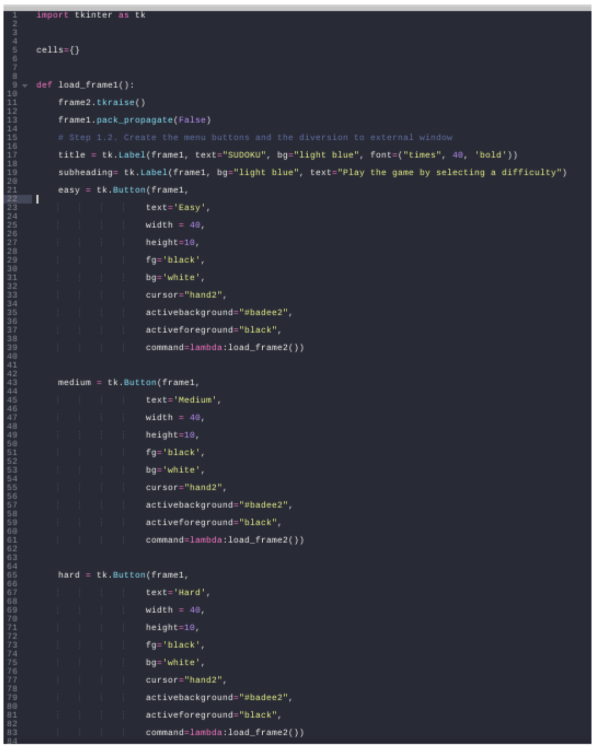
Note: Due to weak time management, I have focused on creating the code and have not documented a significant portion of my code. As a result, this documentation the conversion and changing elements is limited. Deu to this, the documentation will now be based on written steps, challenges or major aspects which were achived.This technique of documentation will be used for the remaining of this assignment.

1. Changing to the tk module
    a. This was not difficult but some errors were made. These were errors like incorrect referencing of the 'tk' in tk.[variable]. These were slightly time-consuming to fix (20 minutes)
    b. After doing so, the realsiaiton that you could put the import tkinter and the from tkinter import .. in the same code.
2. Fixing the sudoku board
    a. The previous code had many bugs ranging from the inputs (due to lack of limitations), the development of the board (were aligned but not one entity) and the effect the board had on other design elements. As a result, I changed the board into two sections. One being a 3x3 board that printed the rows and columns and another being the collation of these 3x3 board to make a 9x9 board
3. Changing and adding features to the main board.
    a. I wanted to change the tactility of the button to include features that were often found in HTML like when a button is hovered over the mouse cursor

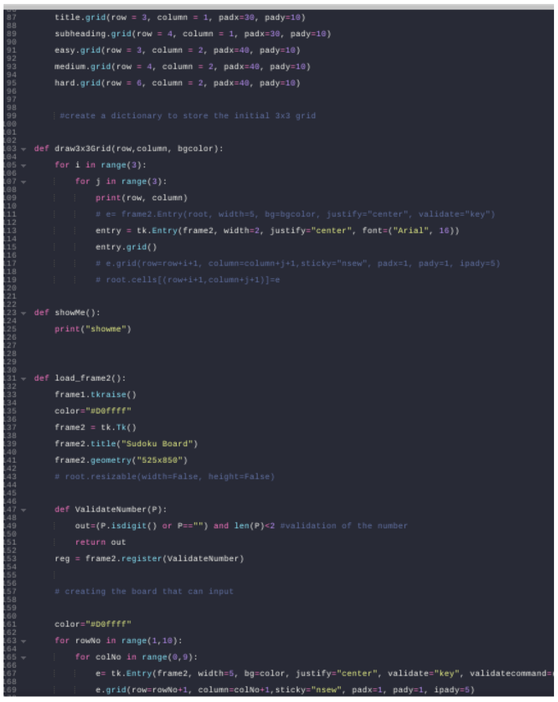
changes, or having and active foreground and background. As a result, I added this into the buttons

4. Changing the name of the methods and removing he class temporarily
    a. The method names were changed and the methods were minimised so that there were only a few main ones (drawing the 3x3 grid, frame1(the menu), frame2(external window for creating the board)

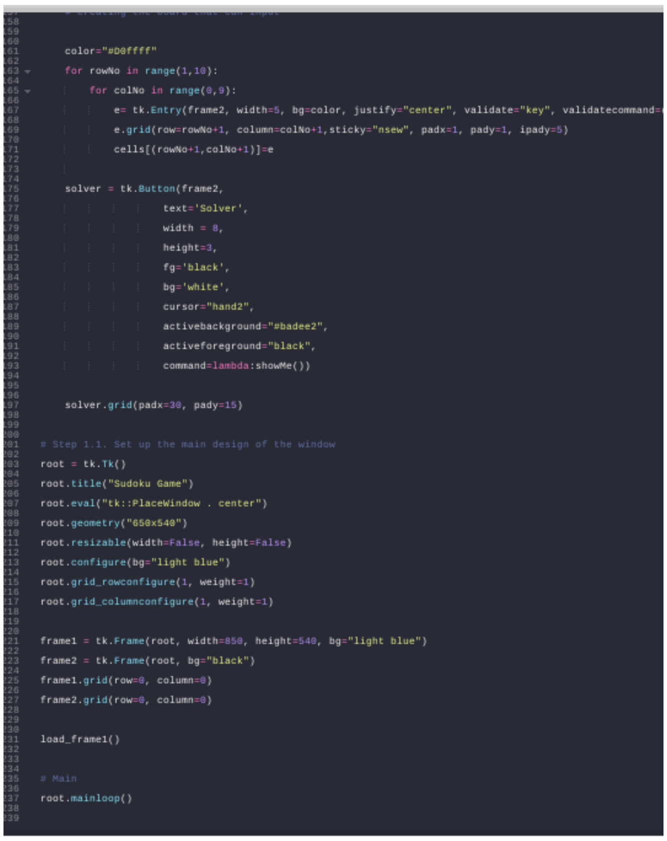
Here is the full code so far:



```python
import tkinter as tk


cells={}


def load_frame1():

    frame2.tkraise()

    frame1.pack_propagate(False)

    # Step 1.2. Create the menu buttons and the diversion to external window

    title = tk.Label(frame1, text="SUDOKU", bg="light blue", font=("times", 40, 'bold'))

    subheading= tk.Label(frame1, bg="light blue", text="Play the game by selecting a difficulty")

    easy = tk.Button(frame1,

                        text='Easy',

                        width = 40,

                        height=10,

                        fg='black',

                        bg='white',

                        cursor="hand2",

                        activebackground="#badee2",

                        activeforeground="black",

                        command=lambda:load_frame2())


    medium = tk.Button(frame1,

                        text='Medium',

                        width = 40,

                        height=10,

                        fg='black',

                        bg='white',

                        cursor="hand2",

                        activebackground="#badee2",

                        activeforeground="black",

                        command=lambda:load_frame2())


    hard = tk.Button(frame1,

                        text='Hard',

                        width = 40,

                        height=10,

                        fg='black',

                        bg='white',

                        cursor="hand2",

                        activebackground="#badee2",

                        activeforeground="black",

                        command=lambda:load_frame2())
```



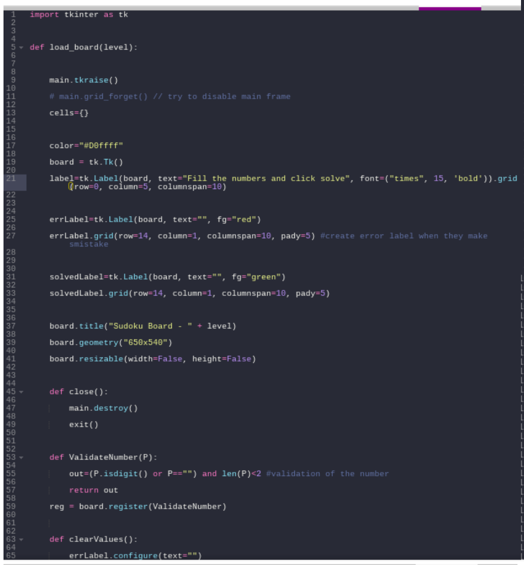
```python
    title.grid(row = 3, column = 1, padx=30, pady=10)

    subheading.grid(row = 4, column = 1, padx=30, pady=10)

    easy.grid(row = 3, column = 2, padx=40, pady=10)

    medium.grid(row = 4, column = 2, padx=40, pady=10)

    hard.grid(row = 6, column = 2, padx=40, pady=10)


    #create a dictionary to store the initial 3x3 grid


def draw3x3Grid(row,column, bgcolor):

    for i in range(3):

        for j in range(3):

            print(row, column)

            # e= frame2.Entry(root, width=5, bg=bgcolor, justify="center", validate="key")

            entry = tk.Entry(frame2, width=2, justify="center", font=("Arial", 16))

            entry.grid()

            # e.grid(row=row+i+1, column=column+j+1,sticky="nsew", padx=1, pady=1, ipady=5)

            # root.cells[(row+i+1,column+j+1)]=e


def showMe():

    print("showme")



def load_frame2():

    frame1.tkraise()

    color="#D0ffff"

    frame2 = tk.Tk()

    frame2.title("Sudoku Board")

    frame2.geometry("525x850")

    # root.resizable(width=False, height=False)


    def ValidateNumber(P):

        out=(P.isdigit() or P=="") and len(P)<2 #validation of the number

        return out

    reg = frame2.register(ValidateNumber)


    # creating the board that can input


    color="#D0ffff"

    for rowNo in range(1,10):

        for colNo in range(0,9):

            e= tk.Entry(frame2, width=5, bg=color, justify="center", validate="key", validatecommand=

            e.grid(row=rowNo+1, column=colNo+1,sticky="nsew", padx=1, pady=1, ipady=5)
```



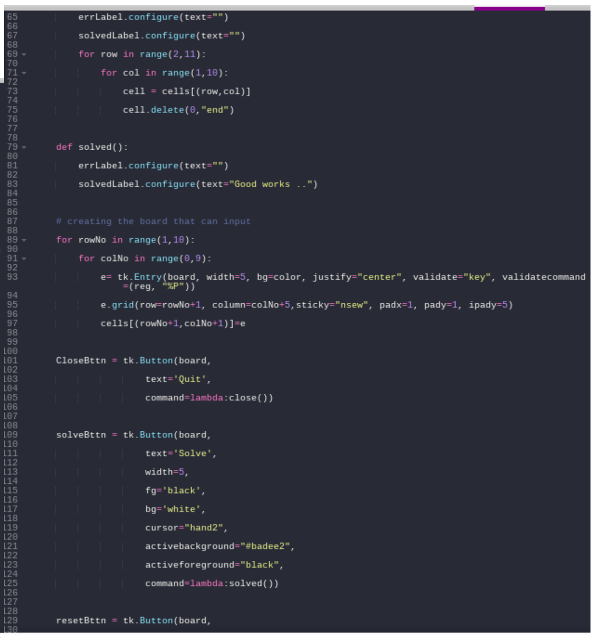
```python
    # Creating the board that can input


    color="#D0ffff"
    for rowNo in range(1,10):
        for colNo in range(0,9):
            e= tk.Entry(frame2, width=5, bg=color, justify="center", validate="key", validatecommand=
            e.grid(row=rowNo+1, column=colNo+1,sticky="nsew", padx=1, pady=1, ipady=5)
            cells[(rowNo+1,colNo+1)]=e


    solver = tk.Button(frame2,
                    text='Solver',
                    width = 8,
                    height=3,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:showMe())


    solver.grid(padx=30, pady=15)


# Step 1.1. Set up the main design of the window
root = tk.Tk()
root.title("Sudoku Game")
root.eval("tk::PlaceWindow . center")
root.geometry("650x540")
root.resizable(width=False, height=False)
root.configure(bg="light blue")
root.grid_rowconfigure(1, weight=1)
root.grid_columnconfigure(1, weight=1)


frame1 = tk.Frame(root, width=850, height=540, bg="light blue")
frame2 = tk.Frame(root, bg="black")
frame1.grid(row=0, column=0)
frame2.grid(row=0, column=0)


load_frame1()


# Main
root.mainloop()
```

21/3/23

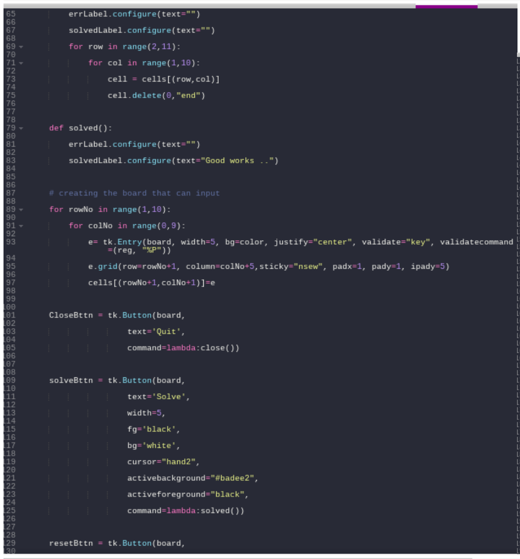
1. Edited the main design of the menu to disable the menu when one option is opened
    a. This was not essential but I disliked how you could open several windows from different options. As a result, I disabled the function once a button was clicked.
2. Create label for sudoku board, error labels and success label for error checking for future implementation of the algorithm.
    a. These labels were used for clarity and to confirm or deny if the user had correctly inputted the data. This will also lead into the testing of the correctness of the board (validation of board)
3. Added another button to the board
    a. The button that was added was a button to close the board and the menu. Whilst this can be easily done through closing the window manually, I did this to optimise my time used to close the window and test code.
4. Reorganised the methods and the structure of the main code
    a. The main methods, and structure of the code is vastly different because I wished to make all the functions, classes, variable more similar and based around the concept instead of being random.



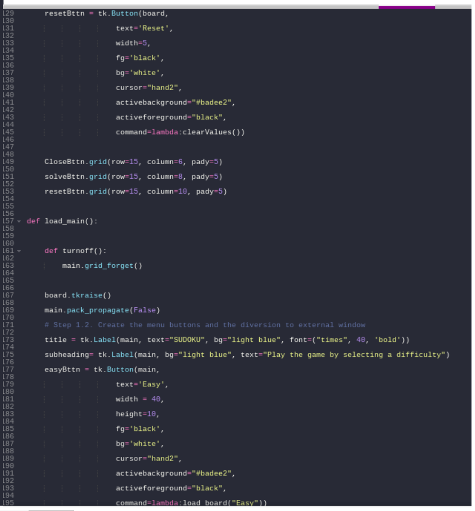
```python
import tkinter as tk


def load_board(level):


    main.tkraise()

    # main.grid_forget() // try to disable main frame

    cells={}


    color="#D0ffff"

    board = tk.Tk()

    label=tk.Label(board, text="Fill the numbers and click solve", font=("times", 15, 'bold')).grid
        (row=0, column=5, columnspan=10)


    errLabel=tk.Label(board, text="", fg="red")

    errLabel.grid(row=14, column=1, columnspan=10, pady=5) #create error label when they make
        smistake


    solvedLabel=tk.Label(board, text="", fg="green")

    solvedLabel.grid(row=14, column=1, columnspan=10, pady=5)


    board.title("Sudoku Board - " + level)

    board.geometry("650x540")

    board.resizable(width=False, height=False)


    def close():

        main.destroy()

        exit()


    def ValidateNumber(P):

        out=(P.isdigit() or P=="") and len(P)<2 #validation of the number

        return out

    reg = board.register(ValidateNumber)


    def clearValues():

        errLabel.configure(text="")
```



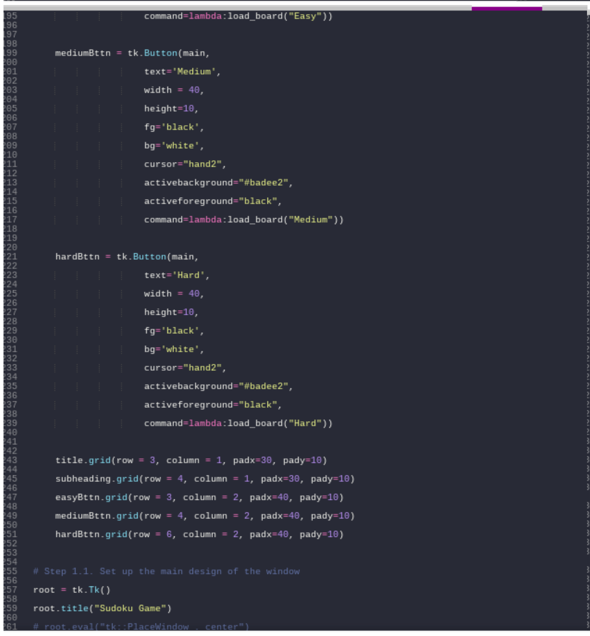
```python
        errLabel.configure(text="")

        solvedLabel.configure(text="")

        for row in range(2,11):

            for col in range(1,10):

                cell = cells[(row,col)]

                cell.delete(0,"end")


    def solved():

        errLabel.configure(text="")

        solvedLabel.configure(text="Good works ..")


    # creating the board that can input

    for rowNo in range(1,10):

        for colNo in range(0,9):

            e= tk.Entry(board, width=5, bg=color, justify="center", validate="key", validatecommand
                =(reg, "%P"))

            e.grid(row=rowNo+1, column=colNo+5,sticky="nsew", padx=1, pady=1, ipady=5)

            cells[(rowNo+1,colNo+1)]=e


    CloseBttn = tk.Button(board,

                    text='Quit',

                    command=lambda:close())


    solveBttn = tk.Button(board,

                    text='Solve',

                    width=5,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:solved())


    resetBttn = tk.Button(board,
```



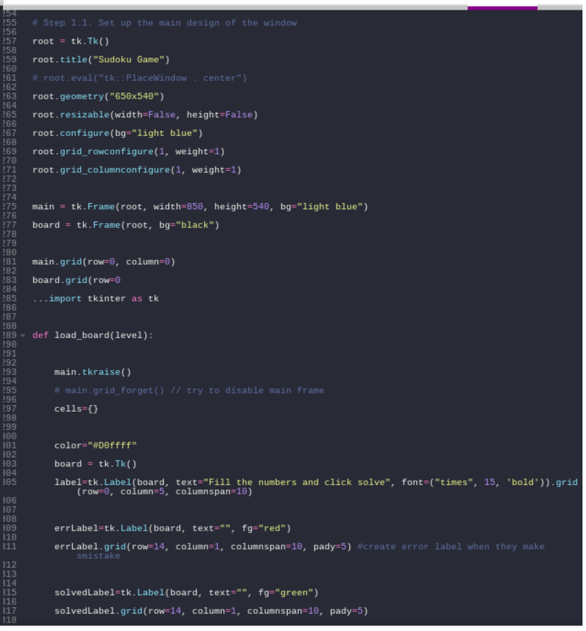
```python
        errLabel.configure(text="")

        solvedLabel.configure(text="")

        for row in range(2,11):

            for col in range(1,10):

                cell = cells[(row,col)]

                cell.delete(0,"end")


    def solved():

        errLabel.configure(text="")

        solvedLabel.configure(text="Good works ..")


    # creating the board that can input

    for rowNo in range(1,10):

        for colNo in range(0,9):

            e= tk.Entry(board, width=5, bg=color, justify="center", validate="key", validatecommand
                =(reg, "%P"))

            e.grid(row=rowNo+1, column=colNo+5,sticky="nsew", padx=1, pady=1, ipady=5)

            cells[(rowNo+1,colNo+1)]=e


    CloseBttn = tk.Button(board,

                    text='Quit',

                    command=lambda:close())


    solveBttn = tk.Button(board,

                    text='Solve',

                    width=5,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:solved())


    resetBttn = tk.Button(board,
```



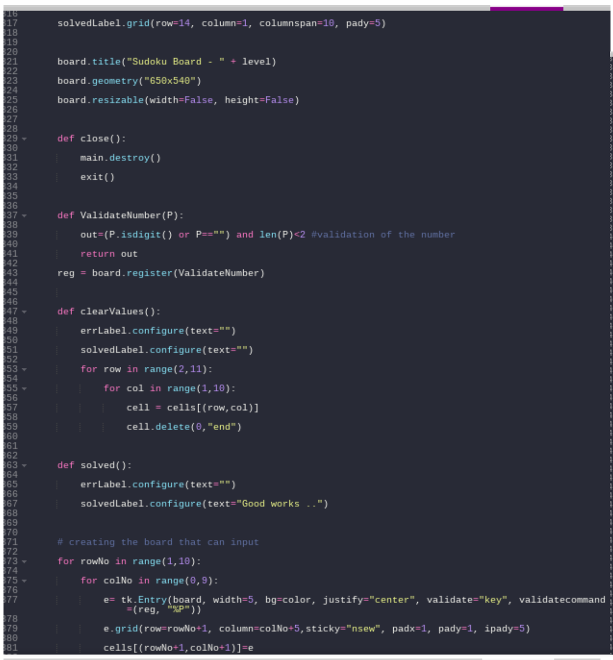
```python
    resetBttn = tk.Button(board,

                    text='Reset',

                    width=5,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:clearValues())


    CloseBttn.grid(row=15, column=6, pady=5)

    solveBttn.grid(row=15, column=8, pady=5)

    resetBttn.grid(row=15, column=10, pady=5)


def load_main():


    def turnoff():

        main.grid_forget()


    board.tkraise()

    main.pack_propagate(False)

    # Step 1.2. Create the menu buttons and the diversion to external window

    title = tk.Label(main, text="SUDOKU", bg="light blue", font=("times", 40, 'bold'))

    subheading= tk.Label(main, bg="light blue", text="Play the game by selecting a difficulty")

    easyBttn = tk.Button(main,

                    text='Easy',

                    width = 40,

                    height=10,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:load_board("Easy"))
```



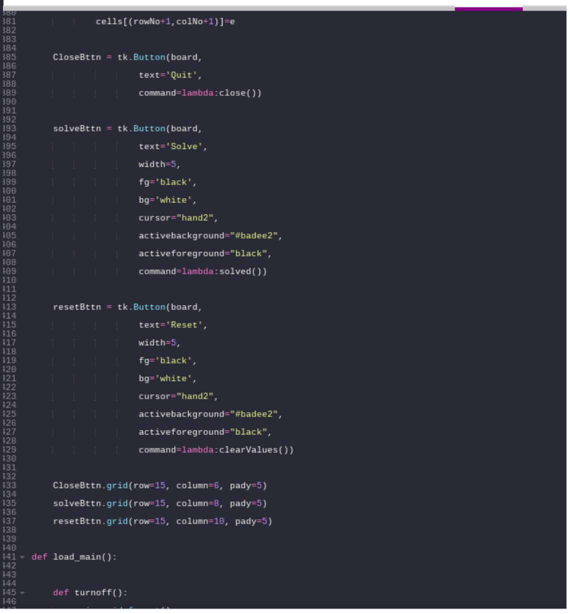
```python
                    command=lambda:load_board("Easy"))


    mediumBttn = tk.Button(main,

                    text='Medium',

                    width = 40,

                    height=10,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:load_board("Medium"))


    hardBttn = tk.Button(main,

                    text='Hard',

                    width = 40,

                    height=10,

                    fg='black',

                    bg='white',

                    cursor="hand2",

                    activebackground="#badee2",

                    activeforeground="black",

                    command=lambda:load_board("Hard"))


    title.grid(row = 3, column = 1, padx=30, pady=10)

    subheading.grid(row = 4, column = 1, padx=30, pady=10)

    easyBttn.grid(row = 3, column = 2, padx=40, pady=10)

    mediumBttn.grid(row = 4, column = 2, padx=40, pady=10)

    hardBttn.grid(row = 6, column = 2, padx=40, pady=10)


# Step 1.1. Set up the main design of the window

root = tk.Tk()

root.title("Sudoku Game")

# root.eval("tk::PlaceWindow . center")
```



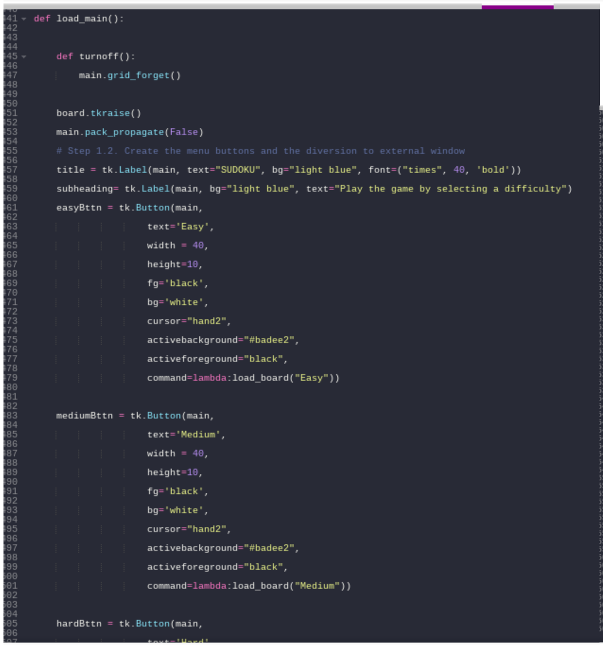
```python
# Step 1.1. Set up the main design of the window

root = tk.Tk()

root.title("Sudoku Game")

# root.eval("tk::PlaceWindow . center")

root.geometry("650x540")

root.resizable(width=False, height=False)

root.configure(bg="light blue")

root.grid_rowconfigure(1, weight=1)

root.grid_columnconfigure(1, weight=1)


main = tk.Frame(root, width=850, height=540, bg="light blue")

board = tk.Frame(root, bg="black")


main.grid(row=0, column=0)

board.grid(row=0

...import tkinter as tk


def load_board(level):


    main.tkraise()

    # main.grid_forget() // try to disable main frame

    cells={}


    color="#D0ffff"

    board = tk.Tk()

    label=tk.Label(board, text="Fill the numbers and click solve", font=("times", 15, 'bold')).grid
        (row=0, column=5, columnspan=10)


    errLabel=tk.Label(board, text="", fg="red")

    errLabel.grid(row=14, column=1, columnspan=10, pady=5) #create error label when they make
        smistake


    solvedLabel=tk.Label(board, text="", fg="green")

    solvedLabel.grid(row=14, column=1, columnspan=10, pady=5)
```



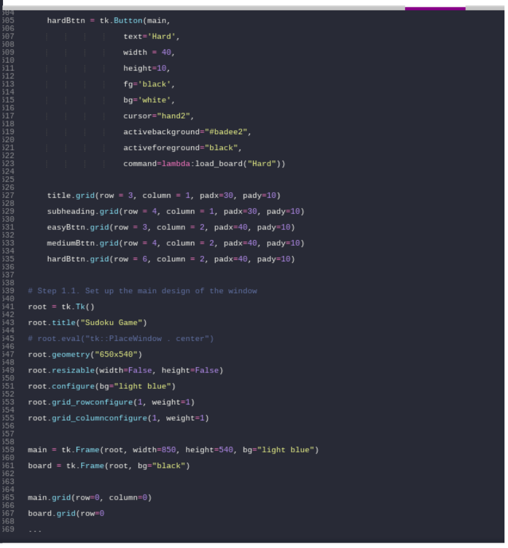
```python
    solvedLabel.grid(row=14, column=1, columnspan=10, pady=5)


    board.title("Sudoku Board - " + level)
    board.geometry("650x540")
    board.resizable(width=False, height=False)


    def close():
        main.destroy()
        exit()


    def ValidateNumber(P):
        out=(P.isdigit() or P=="") and len(P)<2 #validation of the number
        return out
    reg = board.register(ValidateNumber)


    def clearValues():
        errLabel.configure(text="")
        solvedLabel.configure(text="")
        for row in range(2,11):
            for col in range(1,10):
                cell = cells[(row,col)]
                cell.delete(0,"end")


    def solved():
        errLabel.configure(text="")
        solvedLabel.configure(text="Good works ..")


    # creating the board that can input
    for rowNo in range(1,10):
        for colNo in range(0,9):
            e= tk.Entry(board, width=5, bg=color, justify="center", validate="key", validatecommand
                =(reg, "%P"))
            e.grid(row=rowNo+1, column=colNo+5,sticky="nsew", padx=1, pady=1, ipady=5)
            cells[(rowNo+1,colNo+1)]=e
```



```python
            cells[(rowNo+1,colNo+1)]=e


    CloseBttn = tk.Button(board,
                    text='Quit',
                    command=lambda:close())


    solveBttn = tk.Button(board,
                    text='Solve',
                    width=5,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:solved())


    resetBttn = tk.Button(board,
                    text='Reset',
                    width=5,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:clearValues())


    CloseBttn.grid(row=15, column=6, pady=5)
    solveBttn.grid(row=15, column=8, pady=5)
    resetBttn.grid(row=15, column=10, pady=5)


def load_main():


    def turnoff():
```



```python
def load_main():


    def turnoff():
        main.grid_forget()


    board.tkraise()
    main.pack_propagate(False)
    # Step 1.2. Create the menu buttons and the diversion to external window
    title = tk.Label(main, text="SUDOKU", bg="light blue", font=("times", 40, 'bold'))
    subheading= tk.Label(main, bg="light blue", text="Play the game by selecting a difficulty")
    easyBttn = tk.Button(main,
                    text='Easy',
                    width = 40,
                    height=10,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:load_board("Easy"))


    mediumBttn = tk.Button(main,
                    text='Medium',
                    width = 40,
                    height=10,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:load_board("Medium"))


    hardBttn = tk.Button(main,
```



```python
    hardBttn = tk.Button(main,
                    text='Hard',
                    width = 40,
                    height=10,
                    fg='black',
                    bg='white',
                    cursor="hand2",
                    activebackground="#badee2",
                    activeforeground="black",
                    command=lambda:load_board("Hard"))


    title.grid(row = 3, column = 1, padx=30, pady=10)
    subheading.grid(row = 4, column = 1, padx=30, pady=10)
    easyBttn.grid(row = 3, column = 2, padx=40, pady=10)
    mediumBttn.grid(row = 4, column = 2, padx=40, pady=10)
    hardBttn.grid(row = 6, column = 2, padx=40, pady=10)


# Step 1.1. Set up the main design of the window
root = tk.Tk()
root.title("Sudoku Game")
# root.eval("tk::PlaceWindow . center")
root.geometry("650x540")
root.resizable(width=False, height=False)
root.configure(bg="light blue")
root.grid_rowconfigure(1, weight=1)
root.grid_columnconfigure(1, weight=1)


main = tk.Frame(root, width=850, height=540, bg="light blue")
board = tk.Frame(root, bg="black")


main.grid(row=0, column=0)
board.grid(row=0
...
```

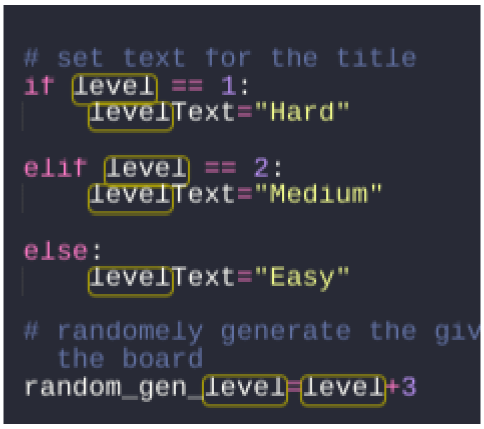
This is unoptimized code with testing and running, this is not efficient and will be significantly reduce for the final code

21/6/23-22/6/23

Aim: I will complete the random generation, validation method using brute force. Major aspects such as sort functions, reading and updating data from other files will be ignored due to time constraints

Random generation

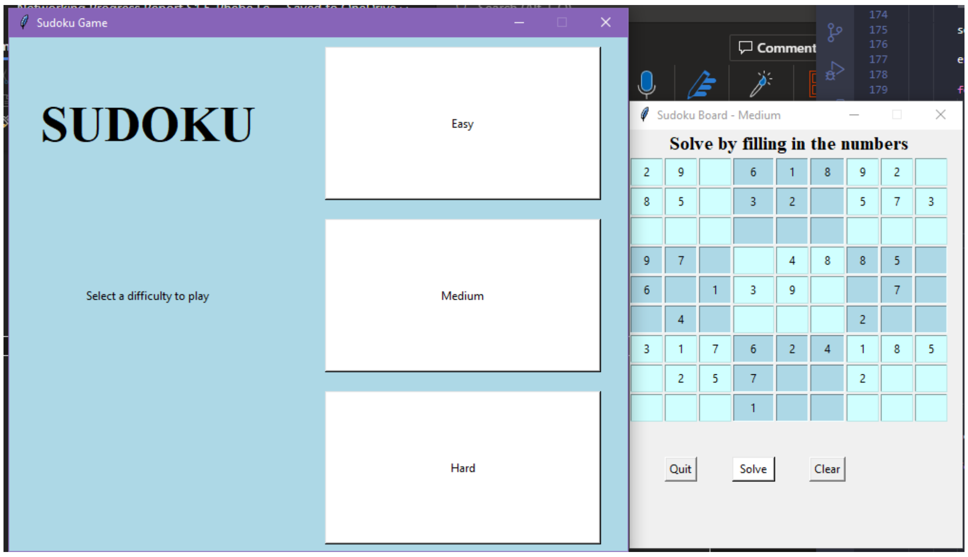
- Through the process of creating the random generation I found problems such as the inability to randomly generate for the whole board. The random generation of this sudoku is conducted through the 3x3 squares, this means whilst the squares are all unique and different, the placement and the numbers of each row may not range exclusively from 1-9. Yet through this process, I created a validation method to check of the numbers were ranging from 1-9. This worked through the command-line prompt yet did not work on the main GUI window.
- Changing the 3x3 board will be too late, the random generation will only occur within the 3x3 boards now
- This process according to my previous code should have been repeated to the other levels but I created a level-based system to create the random generations based on what number the level was (see below). This section impacts how many hints the user will receive when they first open the puzzle. Easy being 6, medium being 5 and hard being 4 hints. When the user presses the solve button, the function evaluates the answer
- Testing the solve button
    - When a board with procreated random ints, is evaluated by the solve button, the message displays invalid answer. This also leads the terminal to display a board that removes the errors within the code from repeated numbers



```
# set text for the title
if level == 1:
    levelText="Hard"

elif level == 2:
    levelText="Medium"

else:
    levelText="Easy"

# randomely generate the giv
  the board
random_gen_level=level+3
```

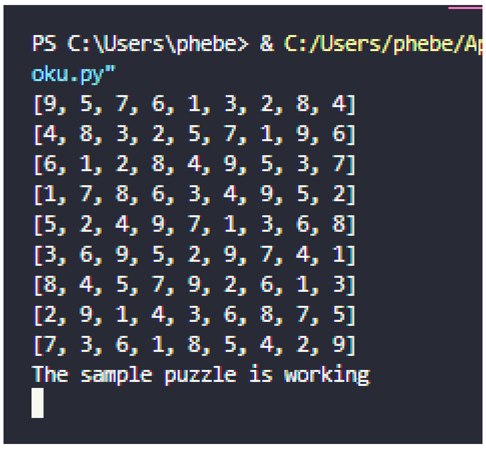
Brute force algorithm

- The brute force algorithm is used when the users evaluate the sudoku board through the solve button. Whilst this doesn’t solve the puzzle, it does allow the user to check if the answer is correct.
- The brute force method also allows the user to check if the board is solvable but only through the terminal.
- Testing the brute force method:
    - Use the clear button and input a valid sudoku board. This results in a confirmation message that the sudoku board is correct
    -



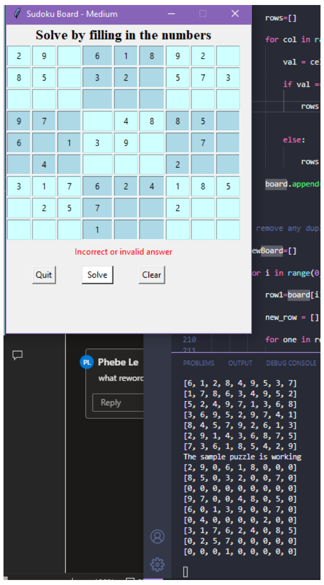
2	9		6	1	8	9	2	
8	5		3	2		5	7	3
9	7			4	8	8	5	
6		1	3	9			7	
	4					2		
3	1	7	6	2	4	1	8	5
	2	5	7			2		
			1					

This is when the medium button is pressed



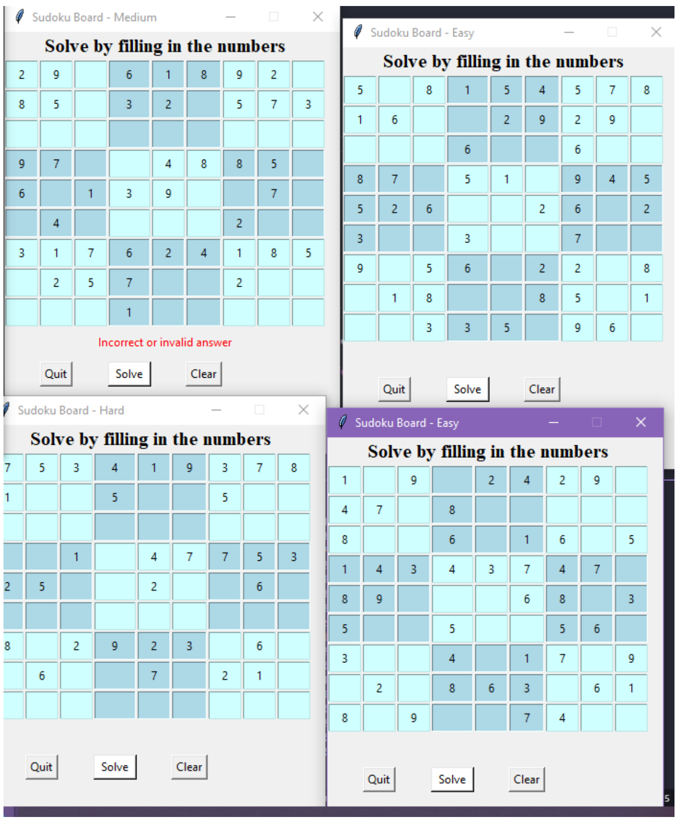
```
PS C:\Users\phebe> & C:/Users/phebe/Ap
oku.py"
[9, 5, 7, 6, 1, 3, 2, 8, 4]
[4, 8, 3, 2, 5, 7, 1, 9, 6]
[6, 1, 2, 8, 4, 9, 5, 3, 7]
[1, 7, 8, 6, 3, 4, 9, 5, 2]
[5, 2, 4, 9, 7, 1, 3, 6, 8]
[3, 6, 9, 5, 2, 9, 7, 4, 1]
[8, 4, 5, 7, 9, 2, 6, 1, 3]
[2, 9, 1, 4, 3, 6, 8, 7, 5]
[7, 3, 6, 1, 8, 5, 4, 2, 9]
The sample puzzle is working
```

In the terminal, this sample puzzle is inputted out for users if they want to test the clear and validity



```
[6, 1, 2, 8, 4, 9, 5, 3, 7]
[1, 7, 8, 6, 3, 4, 9, 5, 2]
[5, 2, 4, 9, 7, 1, 3, 6, 8]
[3, 6, 9, 5, 2, 9, 7, 4, 1]
[8, 4, 5, 7, 9, 2, 6, 1, 3]
[2, 9, 1, 4, 3, 6, 8, 7, 5]
[7, 3, 6, 1, 8, 5, 4, 2, 9]
The sample puzzle is working
[2, 9, 0, 6, 1, 8, 0, 0, 0]
[8, 5, 0, 3, 2, 0, 0, 7, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[9, 7, 0, 0, 4, 8, 0, 5, 0]
[6, 0, 1, 3, 9, 0, 0, 7, 0]
[0, 4, 0, 0, 0, 0, 2, 0, 0]
[3, 1, 7, 6, 2, 4, 0, 8, 5]
[0, 2, 5, 7, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0]
```

This code shows the same puzzle when the solve button is pressed. These inputs an incorrect label and displays the edited restriction to a board within the terminal



Solve by filling in the numbers

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

Incorrect or invalid answer

Solve by filling in the numbers

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

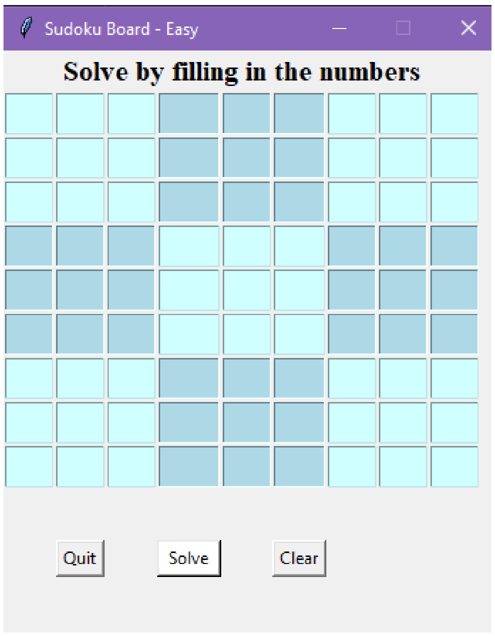
Solve by filling in the numbers

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

Solve by filling in the numbers

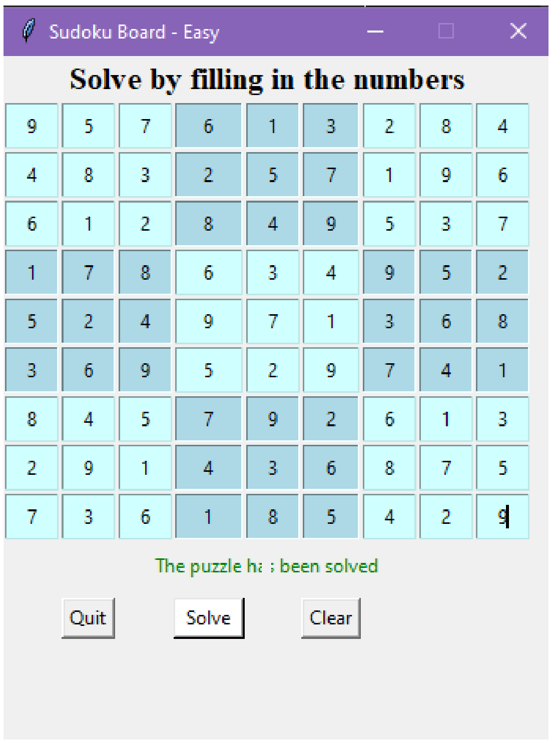
1		9		2	4	2	9	
4	7		8					
8			6		1	6		5
1	4	3	4	3	7	4	7	
8	9				6	8		3
5			5			5	6	
3			4		1	7		9
	2		8	6	3		6	1
8		9			7	4		

The code above is the generation of other boards within the random function for other boards and the number of hints



Solve by filling in the numbers

This is the result of pressing the clear button



Solve by filling in the numbers

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

The puzzle ha : been solved

The cleared board was then inputted with numbers from a solved board. This resulted in a confirmation method.

This is a list of errors and strange exceptions with the code.

- Random generation
  - Works for all puzzle modes. Yet the accuracy of 1-9 random generation is limited to a 3x3 square. This means 1-9 are often repeated in the GUI window.
  - Yet in the serial monitor the validation method (which was supposed to be initially used to test and further implement into the GUI window) works simultaneously to the GUI window. It shows the errors and immediately turns them into zeros, or other corresponding numbers which effect that invalid number.

Error

- Random generate do the puzzle and check due to backtracking
- Backtrack has error
- Logic will work for some case (algorithm may work)
- Serial monitor
- To check validation to check the logic and the validation method
- To check the algorithm, you can singlehandedly type it or use the sample to test it out. The serial monitor will show the correct/incorrectness