3.1 Functions & modules

3.2 Flow chart

1. Display of text:

We have created three frames:

- Top frame

  It includes the title (game name).

- Left frame

  It have a count of remaining cells.

- Centre frame

  The main purpose & what the game is.....

Use the mouse's left and right buttons: The mouse is the only tool that you'll need to play Minesweeper. The left mouse button is used to click squares that contain mines (colour =RED),
while the right mouse button doesn't have a mine.
(colour =. ORANGE)

- On higher difficulties, you'll need to mark squares that you suspect contain mines until you can verify that they do contain mines.

Don't worry about your first click. The first square that you click will never have a mine beneath it; clicking a square will clear off some of the board while numbering other squares.

Know what the numbers mean. A number on a square refers to the number of mines that are currently touching that square. For example, if there are two squares touching each other and one of the squares has "1" on it, you know that the square next to it has a mine beneath it.

- Frame=A frame is a widget that displays as a simple rectangle. Typically, you use a frame to organize other widgets both visually and at the coding level. To create a frame, Syntax: frame_name = Frame(parent, **options)

- Message box = MessageBox Widget is used to display the message boxes in the python applications. This module is used to display a message using provides a number of functions.

  Syntax : tkMessageBox.FunctionName(title, message [, options])
- Button=The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.
- Functions = To perform certain task & to complete the purpose.
- Text =It allows user to edit multiple text & format the way it has to be displayed.

OOP's concepts ~

- **Classes & objects**- A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

Functionality used

- **On Left click**:

```
def left_click_actions(self,event):
        if self.is_mine:
            self.show_mine()
        else:
            if self.surrounded_cells_mines_lenght == 0:
                for cell_obj in self.surrounded_cells:
                    cell_obj.show_cell()
            self.show_cell()
            # if mines count is equal to the cells count, player won
            if Cell.cell_count == Mines_count:
                messagebox.showinfo('Game Over','Congratulations! You
won the game!')

        #cancel left and right clicks events if cell is already opened:
        self.cell_btn_object.unbind('<Button-1>')
        self.cell_btn_object.unbind('<Button-3>')
```

- **Right click** :

```
def right_click_actions(self,event):
        if not self.is_mine_candidate:
            self.cell_btn_object.configure(bg='orange')
            self.is_mine_candidate = True
        else:
            self.cell_btn_object.configure( bg='SystemButtonFace')
            self.is_mine_candidate = False
```

- **To show mine** :

```
def show_mine(self):
        self.cell_btn_object.configure(bg='red')
        messagebox.showinfo('Game Over','you clicked on a mine')
        self.window.destroy()
        sys.exit(0)
```

- **Show cell:**
  In this function we have a label is to be designed to count cells which show the remaining cells & thus do the functionality as per the code.
- Also we make the class for finding surrounding cells
- **Static method**:
  Methods that are bound to a class rather than its object. They do not require a class instance creation. So, they are not dependent on the state of the object. The @staticmethod is a built-in decorator that defines a static method in the class in Python. A static method doesn't

receive any reference argument whether it is called by an instance of a class or by the class itself.

- **Property method**:
  The @property is a built-in decorator for the property() function in Python. It is used to give "special" functionality to certain methods to make them act as getters, setters, or deleters when we define properties in a class.

- **Random method**:

Python Random module is an in-built module of Python which is used to generate random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc

3.3 Project code

We have created two files:

One was main file name as minesweeper.py

Another one was cell.py file which consist the main part just like classes and all other functionality.

Minesweeper.py file:---

```
from tkinter import*

from cell import Cell

WIDTH=1448
```

```
HEIGHT=720

grid_size=6

CELL_COUNT = grid_size ** 2

Mines_count=(CELL_COUNT)//4

def height_prct(percentage):

    return ( HEIGHT / 100) * percentage

print(height_prct(25))



def width_prct(percentage):

    return ( WIDTH / 100) * percentage

print(width_prct(25))


root=Tk()

root.configure(bg="black")

root.geometry("1448x720")

root.title(f'{WIDTH}x{HEIGHT}')

root.resizable(False,False)



top_frame=Frame(root,bg="black",height=height_prct(25),width=WIDTH)

top_frame.place(x=0,y=0)

game_title =Label(top_frame,bg='black',fg='white',font=('',48),text='Minesweeper Game')

game_title.place(x=width_prct(25),y=0)

left_frame=Frame(root,bg="black",width=width_prct(25),height=HEIGHT)

left_frame.place(x=0,y=height_prct(25))


center_frame=Frame(root,bg="white",height=height_prct(75),width=width_prct(75))
```

```
center_frame.place(x=width_prct(25),y=height_prct(25))


for x in range(grid_size):
    for y in range(grid_size):
        c1=Cell(x,y,root)
        c1.create_btn_object(center_frame)
        c1.cell_btn_object.grid(column=x,row=y)


#call the label from the cell class
Cell.create_cell_count_Label(left_frame)
Cell.cell_count_Label_object.place(x=0,y=0)


Cell.randomize_mines()
"now place method is not good to identify the position of each and every cell"
#c2=Cell()
#c2.create_btn_object(center_frame)
#c2.cell_btn_object.grid(column=1,row=0)



root.mainloop()
```

Cell.py file:---

```
from tkinter import Button,Label
import random   #for random performance
from tkinter import messagebox
import sys
WIDTH=1448
HEIGHT=720
grid_size=6
```

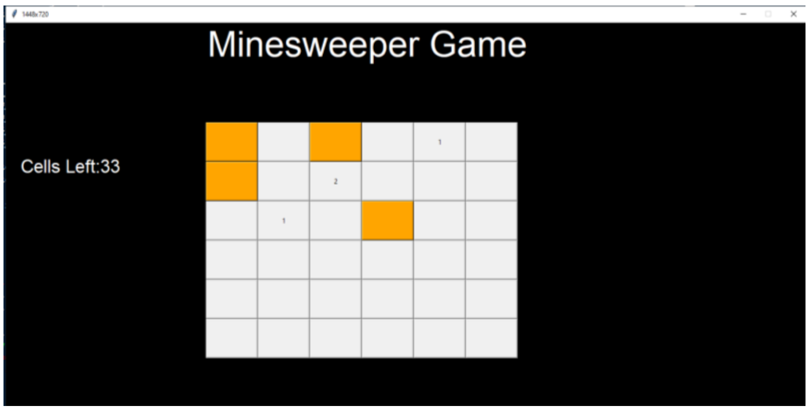
```
CELL_COUNT = grid_size ** 2

Mines_count=(CELL_COUNT)//4

class Cell:

    all=[]

    cell_count = CELL_COUNT

    cell_count_Label_object = None

    def __init__(self,x,y,window,is_mine=False):

        self.is_mine=is_mine

        self.is_opened = False

        self.is_mine_candidate = False

        self.cell_btn_object=None

        self.x=x

        self.y=y

        self.window=window

        Cell.all.append(self)

#append the object to the Cell.all list

#constructor=it is a metal that is going to be called immediately once a class is being
instantiated

    def create_btn_object(self,location):

        btn=Button(location,width=12,height=4)

                    #text=f"{self.x},{self.y}"

        btn.bind("<Button-1>" ,self.left_click_actions)#left click

        btn.bind("<Button-3>" ,self.right_click_actions)#right click

        self.cell_btn_object=btn


    @staticmethod

    def create_cell_count_Label(location):

        lbl=Label(location,bg='black',fg='white',width=12,height=4,font=("",25),text=f"Cells
Left:{Cell.cell_count}")
```

```
    Cell.cell_count_Label_object = lbl

def left_click_actions(self,event):
    if self.is_mine:
        self.show_mine()
    else:
        if self.surrounded_cells_mines_lenght == 0:
            for cell_obj in self.surrounded_cells:
                cell_obj.show_cell()
        self.show_cell()
        # if mines count is equal to the cells count, player won
        if Cell.cell_count == Mines_count:
            messagebox.showinfo('Game Over','Congratulations! You won the game!')

    #cancel left and right clicks events if cell is already opened:
    self.cell_btn_object.unbind('<Button-1>')
    self.cell_btn_object.unbind('<Button-3>')

def get_cell_by_axis(self,x,y):
    #return a cell object based on value of x,y
    for cell in Cell.all:
        if cell.x == x  and cell.y == y:
            return cell

@property
def surrounded_cells(self):
    cells=[
        self.get_cell_by_axis(self.x-1,self.y-1),
        self.get_cell_by_axis(self.x-1,self.y),
        self.get_cell_by_axis(self.x-1,self.y+1),
        self.get_cell_by_axis(self.x,self.y-1),
        self.get_cell_by_axis(self.x+1,self.y-1),
        self.get_cell_by_axis(self.x+1,self.y),
        self.get_cell_by_axis(self.x+1,self.y+1),
        self.get_cell_by_axis(self.x,self.y+1),
    ]
    cells=[cell for cell in cells if cell is not None]
    return cells

@property
def surrounded_cells_mines_lenght(self):
    counter=0
    for cell in self.surrounded_cells:
        if cell.is_mine:
            counter += 1
    return counter

def show_cell(self):
    if not self.is_opened:
        Cell.cell_count -= 1
        self.cell_btn_object.configure(text=self.surrounded_cells_mines_lenght)
        # Replace the text of cell count label with the newer count
        if Cell.cell_count_Label_object:
            Cell.cell_count_Label_object.configure(text=f"Cells Left:{Cell.cell_count}")
        #if this was a mine candidate ,then for safety ,we should
        #configure the background color to SystemButtonFace
```

```
            self.cell_btn_object.configure(bg='SystemButtonFace')


        # Mark the cell as opened (Use is as the last line of this method)
        self.is_opened= True


    def show_mine(self):
        self.cell_btn_object.configure(bg='red')
        messagebox.showinfo('Game Over','you clicked on a mine')
        self.window.destroy()
        sys.exit(0)


    def right_click_actions(self,event):
         if not self.is_mine_candidate:
            self.cell_btn_object.configure(bg='orange')
            self.is_mine_candidate = True
        else:
            self.cell_btn_object.configure( bg='SystemButtonFace')
            self.is_mine_candidate = False


    @staticmethod
    def randomize_mines():
        picked_cells=random.sample(Cell.all,Mines_count)
        for picked_cell in picked_cells:
            picked_cell.is_mine=True


    def __repr__(self):
        return f"Cell({self.x},{self.y})"  #to display the cell values
```

Code screenshot

- Main display window:

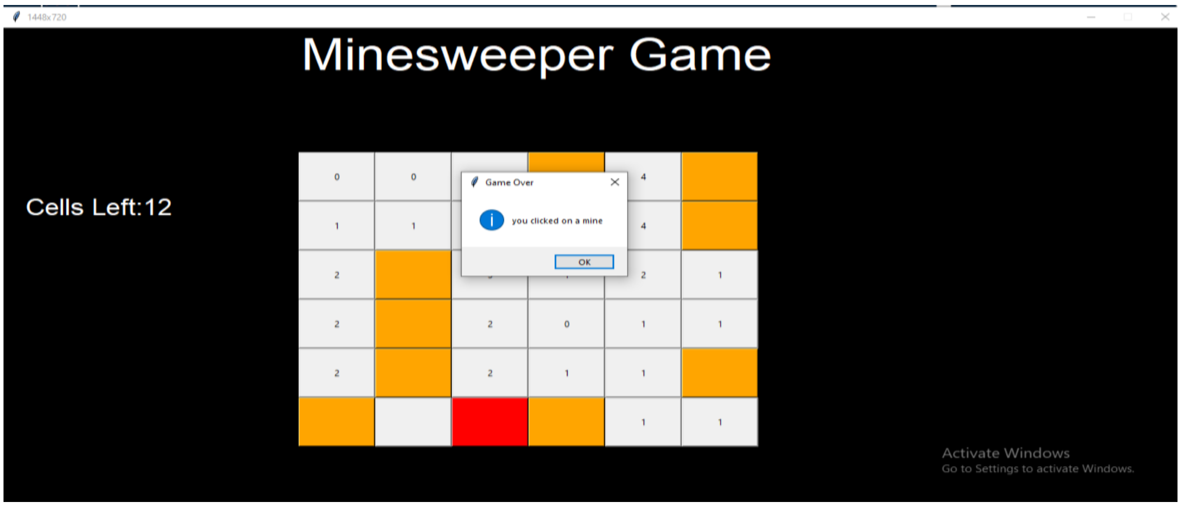


1448x720

Minesweeper Game

Cells Left:33

- Game was ready to play ..Yes it is hard to win!!!!



1448x720

Minesweeper Game

Cells Left:12

Game Over

you clicked on a mine

OK