Minesweeper Game

* tkinter – Python interface to Tcl/ Tk
* tkinter package is the standard python interface to the Tcl/ Tk GUI toolkit.
* In this game we have several cells , that we click and open.
* The main goal is not to click on a cell that has mine behind it.
* If we get a number, there is no mine present.
* If we click on a mine we will lose the game, the main aim is to click o locations which do not have a mine and win the game.

# To create a board using tkinter library:

1. to create a board using tkinter library
2. Instantiate a window instance, can be considered as basic element “root”
3. To change the size of the window we will be using geometry method which will accept string “root.geometry(WIDTHxHEIGHT)
4. We can change the title using title method
5. Not to resize the window pass “False” & “False” for width and height parameters.
6. To close the window use root.mainloop() method, all the code will be written in the “root ---- root.mainloop()”
7. Create a basic element in a window a fame, which is just like container which divides the window into some sections to locate where to put the buttons and helps in visualizing the elements.
8. Create “Frame” class , when we instantiate from the “Frame” class it will be responsible for instantiating the frame, this frame must receive an element to locate the frame in (window, background color, width, height)
9. To start this frame use “place” method which will receive a value in pixels, for that need to specify x-axis and y-axis value as (0,0)
10. Create another frame for left sidebar where we will show the score, so we will create another left frame class and instantiate it.
11. In the left frame we take y = 180 as we want to start the frame just after the previous frame, we won’t be using too much hard coded numbers for pixels, for that we create settings.py and utils.py
12. We use formatted string for width and height using settings.py file
13. Not to use hard code values for any parameter we use settings.py file.
14. Create functions that will help us create how much is 25% of the height, we a create a new py file (utils.py)
15. We will use the same procedure for width parameter.
16. Create another frame dedicated for the centre frame.

**Minesweeper.py**

from tkinter import \*

import settings

import utils

root = Tk()

#override the settings of the window

root.geometry(f"{settings.WIDTH}x{settings.HEIGHT}")

root.configure(bg="black")

root.title("Minesweeper Game")

root.resizable(False, False)

top\_frame = Frame(

    root,

    bg = "black", #change later to black

    width = settings.WIDTH,

    height = utils.height\_prct(25) #25% height of the window

)

top\_frame.place(x= 0, y= 0)

left\_frame = Frame(

    root,

    bg = "black", #"blue"- change later to black

    width = utils.width\_prct(25),

    height = utils.height\_prct(75) #here we are using 75% of the entire height

)

left\_frame.place(x= 0, y= utils.height\_prct(25))

center\_frame = Frame(

    root,

    bg="black", #green - change later to black

    width = utils.width\_prct(75),

    height = utils.height\_prct(75)

)

center\_frame.place(x = utils.width\_prct(25), y = utils.height\_prct(25))

 #Run the widow

root.mainloop()

**settings.py**

WIDTH = 1440

HEIGHT = 720

**Utils.py**

import settings

def height\_prct(percentage):

    return ( settings.HEIGHT / 100 ) \* percentage

#to get the % height from the original window height from settings.py

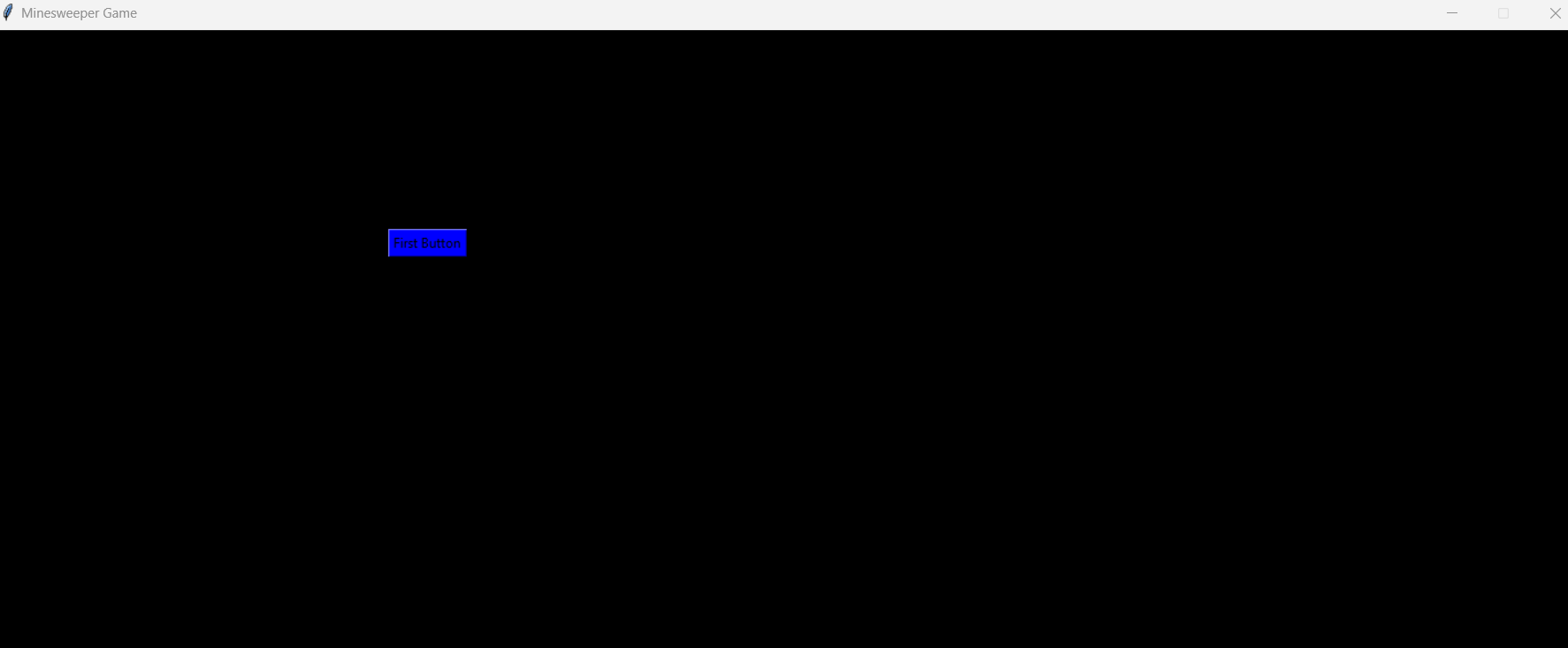
# we will use this in the original file

# print(height\_prct(25))

#similar function will be defined for width

def width\_prct(percentage):

    return (settings.WIDTH / 100) \* percentage



# Creating cells and mines

1. To create a button using tkinter library we will create a class (cell)
2. Create a class “Button”, and we want to abstract it with custom class , to give attribute.
3. This Button class will be used inside the “cell” class.
4. (Constructor in a method which will be called immediately once the class is instantiated.)
5. Create cell.py file and write constructor of the class.
6. Import the “Cell” class from cell.py file in main.py( minesweeper.py) file
7. In the cell file we are going to create a button instance that is going to belong the each cell object, i.e., in the cell.py file we will create (Self.cell\_btn\_object = ) that is how relation between cell and button will be formed.
8. Self.cell\_btn\_object = None at first, we will create an instance method that will create this button for us and assign it to Self.cell\_btn\_object.
9. From tkinter import the Button class
10. Create a button object and receive self only, instantiating an instance of that button class.
11. To use at least one parameter, we use location (to pass an information)
12. Whenever we call this method, we will refer to actual center\_frame which will be from minesweeper.py (main.py) file.
13. Give a random text as a beginner
14. Self.cell\_btn\_object = btn (object which we just have created )
15. That will help us customize this button once I assign a attribute in this button object.

from tkinter import Button

class Cell:

    def \_\_init\_\_(self, is\_mine = False):

      self.is\_mine = is\_mine

  self.cell\_btn\_object = None #None at first

        #we will create a instance  method that will create this button for us and assign it to self.cell\_btn\_object

def create\_btn\_object(self, location):

btn = Button(

location,

text = "Text"

  ) #we are just instantiating an instance of Button class

#locate the element using location

#if we receive the parameter location, then we should pass it ,

whenever we call it, we refer to actual center\_frame

#which will be from minesweeper.py

self.cell\_btn\_object = btn

1. C1.create\_btn\_object (pass a location here in the centre frame as a parameter)
2. C1.cell\_btn\_object.place( x= 0, y= 0)
3. If we create similar button object for c2 we might face an issue as the buttons might overlap if we do not provide the location properly.
4. When we want to create multiple elements dynamically then “place” method might not be the best option. So, we need to change the place method when it comes to dynamically creating the elements.
5. We sift from “place” to “grid” ,
6. Grid takes the parent element and turns it into columns and rows which counts from 0.
7. This will help in creating multiple buttons.

A computer screen with a black background

Description automatically generated

A screenshot of a computer

Description automatically generated

(the above buttons are created using grid(column, row)

To create the buttons as we want in the minesweeper game:

1. To create the buttons for game, we will use nested for loop where we will try to create 25 (5\*5) buttons for x, y respectively and instantiate a object (Cell()), give grid dimension as shown below:

for x in range(5):

    for y in range(5):

      #here we are creating 25 buttons

        c = Cell() #instantiating an object

        c.create\_btn\_object(center\_frame)

        c.cell\_btn\_object.grid(

            column = y, row = x

        )

1. Instead of hard coding the range(5) , we can write in settings.py, and define “GRID\_SIZE”
2. To assign events to our buttons, event = a list of action once you click on a button
3. To assign events to the created buttons we will write in cell.py file we work with “bind()” method
4. When we want to assign an event to a button, we work with bind() method, with bind() method we want to print something when we left click on the button.
5. We want to print something to left click on a button, for that we pass two arguments key (clicked on the button) , and what is the function that you want to execute once you click on the button.

A screenshot of a computer

Description automatically generated

**Cells.py**

from tkinter import Button

class Cell:

    def \_\_init\_\_(self, is\_mine = False):

        self.is\_mine = is\_mine

        self.cell\_btn\_object = None #None at first

        #we will create a instance  method that will create this button for us and assign it to self.cell\_btn\_object

    def create\_btn\_object(self, location):

        btn = Button(

            location,

            text = "Text"

        ) #we are just instantiating an instance of Button class

        #locate the element using location

        #if we receive the parameter location, thn we should pass it , whenever we call it, we refer to actual center\_frame

        #which will be from minesweeper.py

        btn.bind('<Button-1>', self.left\_click\_actions)

        btn.bind('<Button-3>', self.right\_click\_actions)

        self.cell\_btn\_object = btn

        #Button-1 = left click and Button - 3 = right click

        #here we are not calling the method, just passig the reference of that method.

    def left\_click\_actions(self, event ):

        print(event)

        #we need more parameter to assign an event

        print("I am left clicked")

    def right\_click\_actions(self, event ):

        print(event) #we need more parameter to assign an event

        print("I am right clicked")

**Minesweeper.py**

from tkinter import \*

from cell import Cell

import settings

import utils

root = Tk()

#override the settings of the window

root.geometry(f"{settings.WIDTH}x{settings.HEIGHT}")

root.configure(bg="black")

root.title("Minesweeper Game")

root.resizable(False, False)

top\_frame = Frame(

    root,

    bg = "black", #change later to black

    width = settings.WIDTH,

    height = utils.height\_prct(25) #25% height of the window

)

top\_frame.place(x= 0, y= 0)

left\_frame = Frame(

    root,

    bg = "black", #"blue"- change later to black

    width = utils.width\_prct(25),

    height = utils.height\_prct(75) #here we are using 75% of the entire height

)

left\_frame.place(x= 0, y= utils.height\_prct(25))

center\_frame = Frame(

    root,

    bg="black", #green - change later to black

    width = utils.width\_prct(75),

    height = utils.height\_prct(75)

)

center\_frame.place(x = utils.width\_prct(25), y = utils.height\_prct(25))

# #to create a first button

# btn1 = Button(

#     center\_frame,

#     bg = "blue",

#     text = "First Button"

# )

# btn1.place(x = 0, y = 0)

 #Run the widow

# c1 = Cell()

# c1.create\_btn\_object(center\_frame)  #given center\_frame as a parameter

# c1.cell\_btn\_object.grid(

#     column = 0, row = 0

# )

# c2 = Cell()

# c2.create\_btn\_object(center\_frame)

# c2.cell\_btn\_object.grid(

#     column = 0, row = 1

#  )

for x in range(settings.GRID\_SIZE):

    for y in range(settings.GRID\_SIZE):

        c = Cell() #instantiating an object

        c.create\_btn\_object(center\_frame)

        c.cell\_btn\_object.grid(

            column = x, row = y

        )

root.mainloop()

**settings.py**

WIDTH = 1440

HEIGHT = 720

GRID\_SIZE = 6

# Minesweeper Algorithm part 1

1. We will increase the size of our buttons for readability, which can be achieved by going to the instantiation of Button class, we will pass width and height in the above class after location.

def create\_btn\_object(self, location):

        btn = Button(

            location,

            width = 12,

            height = 4,

            text = "Text"

        ) #we are just instantiating an instance of Button class

        #locate the element using location

        #if we receive the parameter location, thn we should pass it , whenever we call it, we refer to actual center\_frame

        #which will be from minesweeper.py

        btn.bind('<Button-1>', self.left\_click\_actions)

        btn.bind('<Button-3>', self.right\_click\_actions)

        self.cell\_btn\_object = btn

A screenshot of a computer

Description automatically generated

1. We don’t have indicators about the buttons, or some attributes.

We add X, y as attributes in the Cell class.

1. We add x,y as input in the Cell class and also the f string for text.

for x in range(settings.GRID\_SIZE):

    for y in range(settings.GRID\_SIZE):

        c = Cell(x,y) #instantiating an object #add x, y

        c.create\_btn\_object(center\_frame)

        c.cell\_btn\_object.grid(

            column = x, row = y

        )

root.mainloop()

from tkinter import Button

class Cell:

    def \_\_init\_\_(self, x, y, is\_mine = False):

        self.is\_mine = is\_mine

        self.cell\_btn\_object = None #None at first

        self.x = x

        self.y = y

        #we will create a instance  method that will create this button for us and assign it to self.cell\_btn\_object

    def create\_btn\_object(self, location):

        btn = Button(

            location,

            width = 12,

            height = 4,

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

1. When we start the game , the game picks up random cell, it converts those cells which are forbidden to click which means “mines”

(we need to write an algorithm that will be responsible to take some cells and turn them into mines)

1. We have is\_mine attribute which is “False”, we will write a method that will pick some objects and which will convert those attributes from is\_mine = False to is\_mine = True so that we will have a great beginning of minesweeper game going on.
2. We define a static method which will be called from main.py file right after we instantiate the objects which does not belong to each instance, it belongs globally to a class. (Randomize mines) we have to write a logic which will convert some cells into mines.
3. How to store instances inside a list?
4. We instantiate 36 instances, but we do not have a control , to take further action from Cell class, we define class attribute “all” which will be an empty list and we will append all the objects of that cell class dynamically inside the init method.
5. We get 36 elements.
6. Using “Cell.all” we get 36 elements here.
7. We create some magic methods based on which the objects will be represented

def \_\_repr\_\_(self):

      return f"Cell({self.x},{self.y})"

1. Each time we want to check , that logic written inside the randomize\_mines() is correct
2. We call randomize\_mines static method in main.py file, every time we write code in it , execute the entire game we get the output.

**Cell.randomize\_mines() #staticmethod**

1. We will import random library which helps in taking random numbers
2. We define a list with names, and randomly selects two names from the list, and print those names while executing the program.
3. Same thing we want to implement with mines, we want to randomly select some cell objects, and change the attribute of “is\_mine” to true.
4. We create picked\_cells = random.sample(Cell.all,9) and print them, where Cell.all will have all the 36 cells out of which 9 will be considered as mines.

@staticmethod

def randomize\_mines():  #it takes couple of cells and turns them into mine

   picked\_cells = random.sample(Cell.all,9) #it includes all the instances

    print(picked\_cells)

A black screen with white text

Description automatically generated

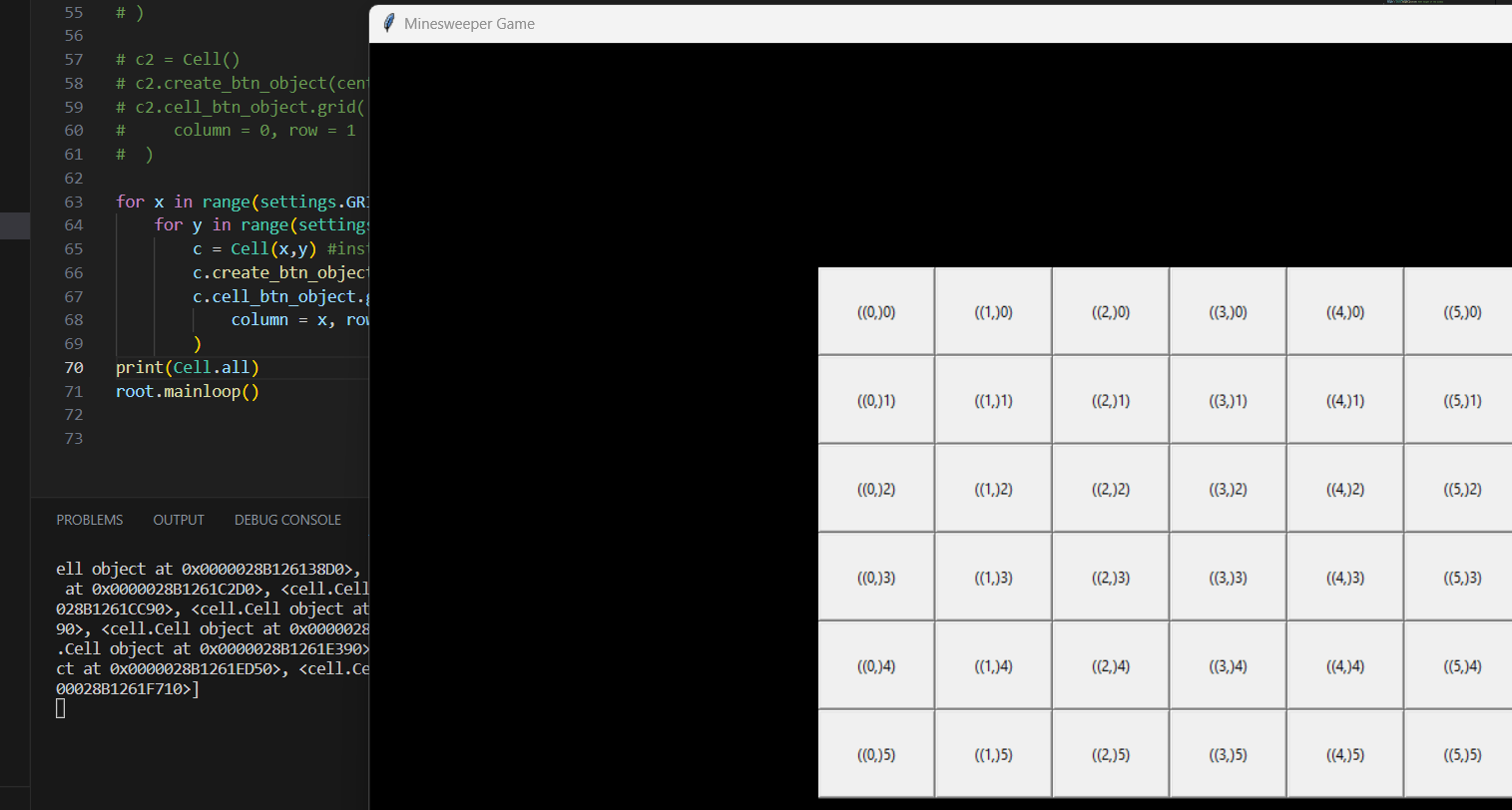
1. We can iterate over the picked\_cells list based on the is\_mine = True or False
2. We test the same in main.py file for is\_mine

for picked\_cell in picked\_cells:

picked\_cell.is\_mine = True

for c in Cell.all:

  print(c.is\_mine)   #to get the values of is\_mine for each cell after picking the random cells as mines



(note above work is correct due to some other changes output is now as below, both are correct,  
there was some syntax issue which was showing different text.)

A screenshot of a computer

Description automatically generated

1. How to be dynamic for the picked\_mine cell amount
2. Use settings.py to calculate how many mines we want in the game, mines\_count

**Cell.py**

  @staticmethod

    def randomize\_mines():  #it takes couple of cells and turns them into mine

        picked\_cells = random.sample(Cell.all,settings.MINES\_COUNT) #it includes all the instances

        print(picked\_cells)

        for picked\_cell in picked\_cells:

            picked\_cell.is\_mine = True

    def \_\_repr\_\_(self):

        return f"Cell({self.x},{self.y})"

**Minesweeper.py**

        )

# print(len(Cell.all))

Cell.randomize\_mines() #staticmethod

root.mainloop()

**settings.py**

WIDTH = 1000

HEIGHT = 720

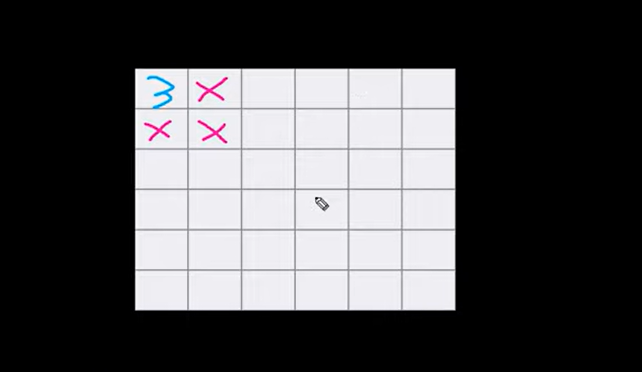
GRID\_SIZE = 6

MINES\_COUNT = (GRID\_SIZE\*\* 2)//4

# Minesweeper Algorithm part 2

What we should calculate when we click on a cell in terms of how many mines are located, it will require more methods, functions, and data to put in.

1. remove the text # text = f"({self.x}{self.y})" and re-run
2. if we click on cell, it should give a number of how many mines are surrounded by that cell



1. if clicked cell is a mine then we have to take some actions, for that we define a show\_mine() method in which we write actions which will be responsible for showing that cell as an mine.

def show\_mine(self):

#a logic to interrupt the game and display a message that player lost!

#converting a background of that cell into a red background should be enough to highlight the mine

self.cell\_btn\_object.configure(bg ="red")

based on this whenever we click on a mine that cell will be highlighted in red.

A white and red squares

Description automatically generated

1. we must design if cell is not a mine , we must show the number of mines surrounded by that cell.
2. We add the text line again to show the cell number by using that we will show the number of mines surrounded by that cell.
3. If we select one cell, then we should start iterating over other cell than that cell to check the mines, collecting the number of mines and that needs to be displayed here.
4. We will define a method that will give x, y as a parameter and give us back the object of cell.

In this method we will return the cell object if value is equal to given input, if it’s a mine we won’t get the cell number.

Note: error : we get None value not cell(0,0) when clicked on a non-mine cell. (Check again)

So if clicked cell is not a mine we should see the object

**Solution of the error:**

class Cell:

    all = []

    def \_\_init\_\_(self, x, y, is\_mine = False):

        self.is\_mine = is\_mine

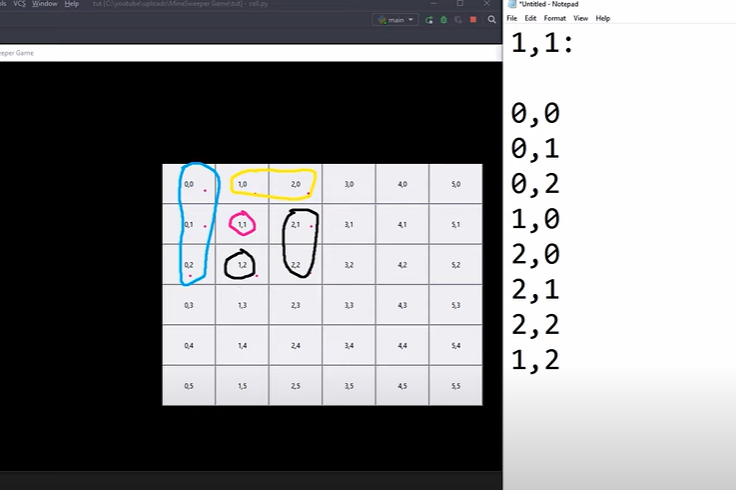
        self.cell\_btn\_object = None #None at first

        self.x = x

        self.y = y

#Here removed the comma written after (,) which was giving an issue while returning the cell object with x and y value in get\_cell\_by\_axis method defined for show\_cell

1. Imagine click on 1,1 , we have to collect all those surrounded cells of click cell(1,1) as below:



Make a list “surrounding\_list” which will have those 8 objects by get\_cell\_by\_axis method.

e.g. for (1,1) we should decrease x by 1, y by 1 = (0,0) similarly for all other write formulas.

So to test we click on (1,1) and check the answer:

A screenshot of a computer

Description automatically generated

We get all the 8 objects, but now problem is if we click on (0,0) it has three surrounding cells.

This might lead to not working, we can test that.

We get “None” for few of the values as below when clicking on (0,0)

A screen shot of a computer

Description automatically generated

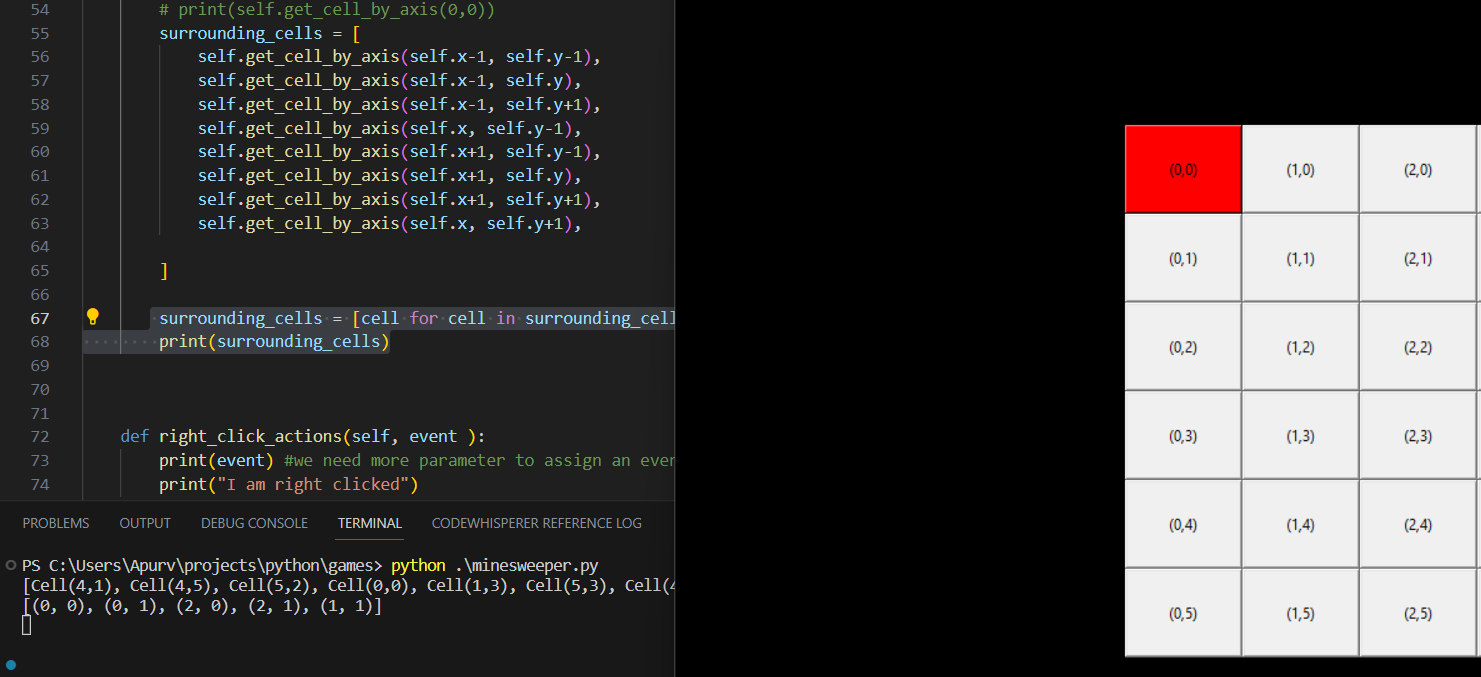
(we need to remove those None values later values, which can be done by list comprehension)

1. We are going to overwrite the surrounding\_cells list using list comprehension, we want the exact same list except the None values.
2. With the help of list comprehension, the results are same as above, but we can use loop that will take care of None value.
3. So, if click on a cell we get all the requires values except None using below code line:

surrounding\_cell = [cell for cell in surrounding\_cells if cell is not None]

print(surrounding\_cells)

e.g., if we click on (1,0) we get following output without “None” value.



1. We create separated method, so that we could have a property that will represent surrounding cell object. A property is an attribute that is read-only , which we can mark read-only using a decorator.
2. So we can access the surrounding cells by accessing the property.
3. We have created a method that brings in a surrounding cell object , then iterate over that and identify which one is mine and which one is not, we define one more method for that def (surrounding\_cells\_mines\_length) which will count the number of mines present in surrounding cells whenever a cell is clicked.

@property

def surrounded\_cells\_mine\_length(self):

#it will count the mines that are in the surrounding cells, whenever a cell is clicked.

#this will also be a read-only attribute

     counter = 0

     for cell in self.surrounded\_cells:

        if cell.is\_mine:

            counter += 1

     return counter

error : AttributeError: 'tuple' object has no attribute 'is\_mine' (need to check again) for above code.

1. We can now change the behaviour of show\_cell method. We call the cell\_btn\_object and configure the text based on surrounded\_cells\_mine\_length and remove the text from create\_btn\_object

A screenshot of a game

Description automatically generated

1. So we have now displayed the amount of mines for each cell.

**Cell.py**

from tkinter import Button

import random

import settings

class Cell:

    all = []

    def \_\_init\_\_(self,x, y, is\_mine=False):

        self.is\_mine = is\_mine

        self.cell\_btn\_object = None

        self.x = x

        self.y = y

        # Append the object to the Cell.all list

        Cell.all.append(self)

    def create\_btn\_object(self, location):

        btn = Button(

            location,

            width=12,

            height=4,

        )

        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

    def left\_click\_actions(self, event):

        if self.is\_mine:

            self.show\_mine()

        else:

            self.show\_cell()

    def get\_cell\_by\_axis(self, x,y):

        # Return a cell object based on the 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\_length(self):

        counter = 0

        for cell in self.surrounded\_cells:

            if cell.is\_mine:

                counter += 1

        return counter

    def show\_cell(self):

        self.cell\_btn\_object.configure(text=self.surrounded\_cells\_mines\_length)

    def show\_mine(self):

        # A logic to interrupt the game and display a message that player lost!

        self.cell\_btn\_object.configure(bg='red')

    def right\_click\_actions(self, event):

        print(event)

        print("I am right clicked!")

    @staticmethod

    def randomize\_mines():

        picked\_cells = random.sample(

            Cell.all, settings.MINES\_COUNT

        )

        for picked\_cell in picked\_cells:

            picked\_cell.is\_mine = True

    def \_\_repr\_\_(self):

        return f"Cell({self.x}, {self.y})"

# Display Game Data

We will design to show more information to the game. There are total 36 cells, if we click on one cell we have 35 cells to click on. We need to display this kind of information; we will develop a functionality that will open the surrounding cells if the click cell is zero. If there are zero mines surrounding the cell then there is no point to not automatically click, so if there are zero mines then clicking on them is safe. This will be done automatically for us, which will improve the speed of the game.

1. How to automatically open the surrounding cells if we see the number of 0 if we click on any cell (i.e., there are no mines present)
2. Check if display number is 0 or not , for that we use property of surrounding\_cells\_mines\_length , based on which we define a for loop which will be responsible to show the surrounding mines length for cells. We have a property that shows a list with all the surrounding\_cells\_object.
3. Now if we click on 0, that automatically opens all other cell which are safe to click (other than mine as below)

def left\_click\_actions(self, event):

     if self.is\_mine:

          self.show\_mine()

      else:

          if self.surrounded\_cells\_mines\_length == 0:

               for cell\_obj in self.surrounded\_cells:

                    cell\_obj.show\_cell()

            self.show\_cell()

A screenshot of a game

Description automatically generated

1. Information like how many cells are left in every situation, every time we click on a cell, the amount of cell decreases, which we must show. Approach will be like create button.

We will create a method that will create a text element in the window and then we will call this method from main.py file.

(This is going to be the class we will initialize )

1. For label object we are going to do the same method, like button object, create an attribute that belong to class method, and not instance method.
2. We can get the label by cell\_count\_label\_object and give place

@staticmethod

  def create\_cell\_count\_label(location):

        lbl = Label(

            location,

            text = f"Cells Left: {settings.CELL\_COUNT}"

      )

        Cell.cell\_count\_label\_object = lbl

#This label does not belong to each of our rows.

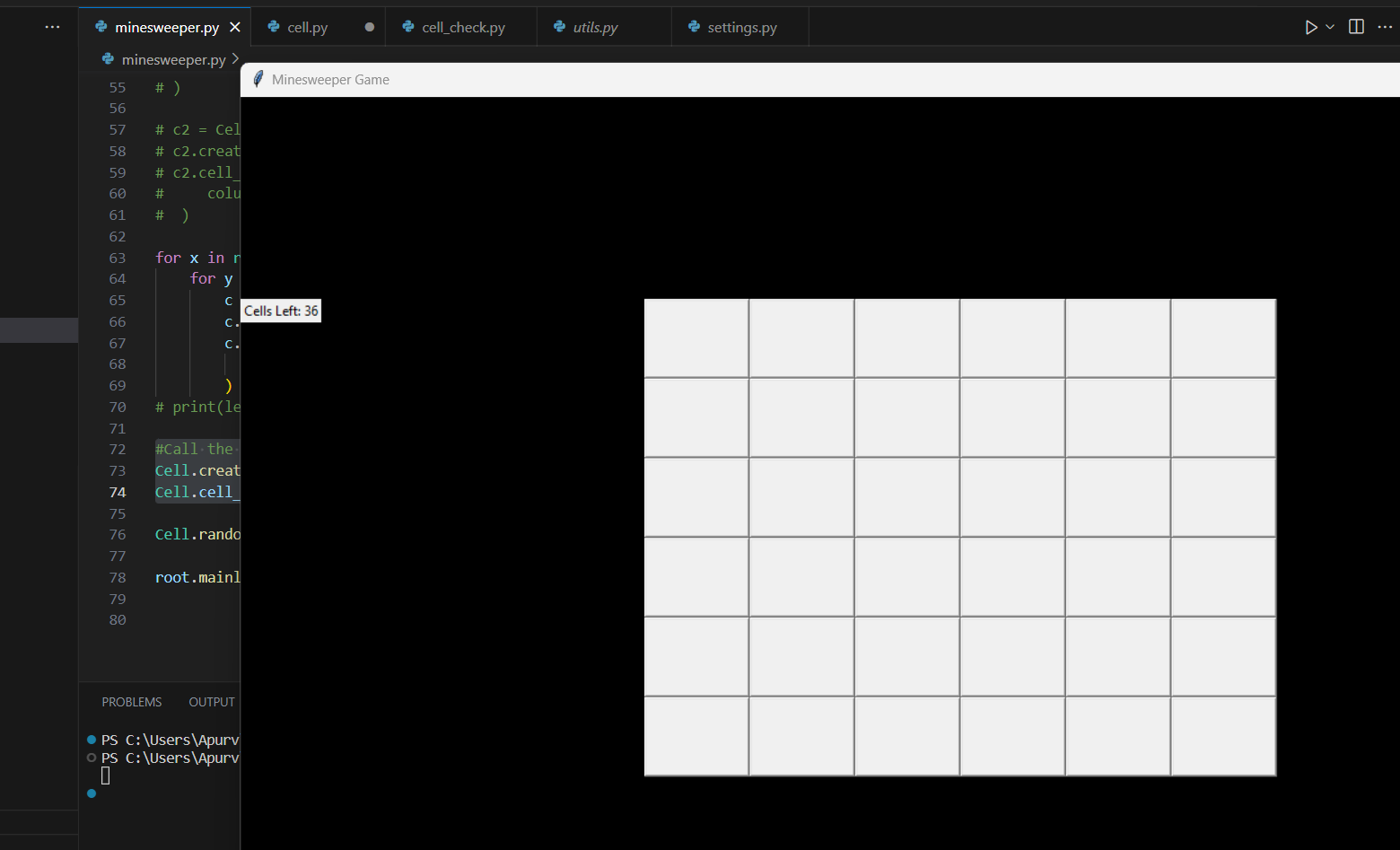
#This cannot be an instance method this is a one time, we do not want to call this method for each cell.

#thats why it needs to be staticmethod, which is for use case of the class, and not for use case of the instance.

#Call the label from the Cell class

Cell.create\_cell\_count\_label(left\_frame)

Cell.cell\_count\_label\_object.place(x=0, y=0)



We increase the width and height of the label object by adding width & height.

As well as add font, front and back colour. Which gives result like below:

@staticmethod

def create\_cell\_count\_label(location):

lbl = Label(

          location,

            text = f"Cells Left: {settings.CELL\_COUNT}",

            bg = "black",

            fg = "white",

            width = 12,

            height = 4,

            font = ("", 30)

        )

       Cell.cell\_count\_label\_object = lbl

A screenshot of a computer

Description automatically generated

(font by default accepts a tuple)

1. How interactively change the text every time that we click on a cell, and click cell which is actually cell not a mine when finishes the game, this means that we need to change the number of cells left in the text that we have located in left frame, for that we need to make changes in show\_cell()
2. #Replace the text of cell count label with newer count

For that we need to check cell\_count\_label\_object is None or not.

And then configure its text to newer text.

def show\_cell(self):

     self.cell\_btn\_object.configure(text=self.surrounded\_cells\_mines\_length)

     #Repalce the text of cell count label with new count

     if Cell.cell\_count\_label\_object:

        Cell.cell\_count\_label\_object.configure(text= 'Changed!')

A screenshot of a computer

Description automatically generated

Cell count changes every time the show\_cell() method is called. We decrease the amount of cell every time show\_cell() is called.

cell\_count = settings.CELL\_COUNT

we decrease the value every time the show\_cell() method is called.

We change the reference of cell count in create\_cell\_count\_label

@staticmethod

    def create\_cell\_count\_label(location):

        lbl = Label(

            location,

            text = f"Cells Left: {Cell.cell\_count}",

this is the variable which we are decreasing. We configure the same text in show\_cell()

def show\_cell(self):

        Cell.cell\_count -=1

        self.cell\_btn\_object.configure(text=self.surrounded\_cells\_mines\_length)

        #Repalce the text of cell count label with new count

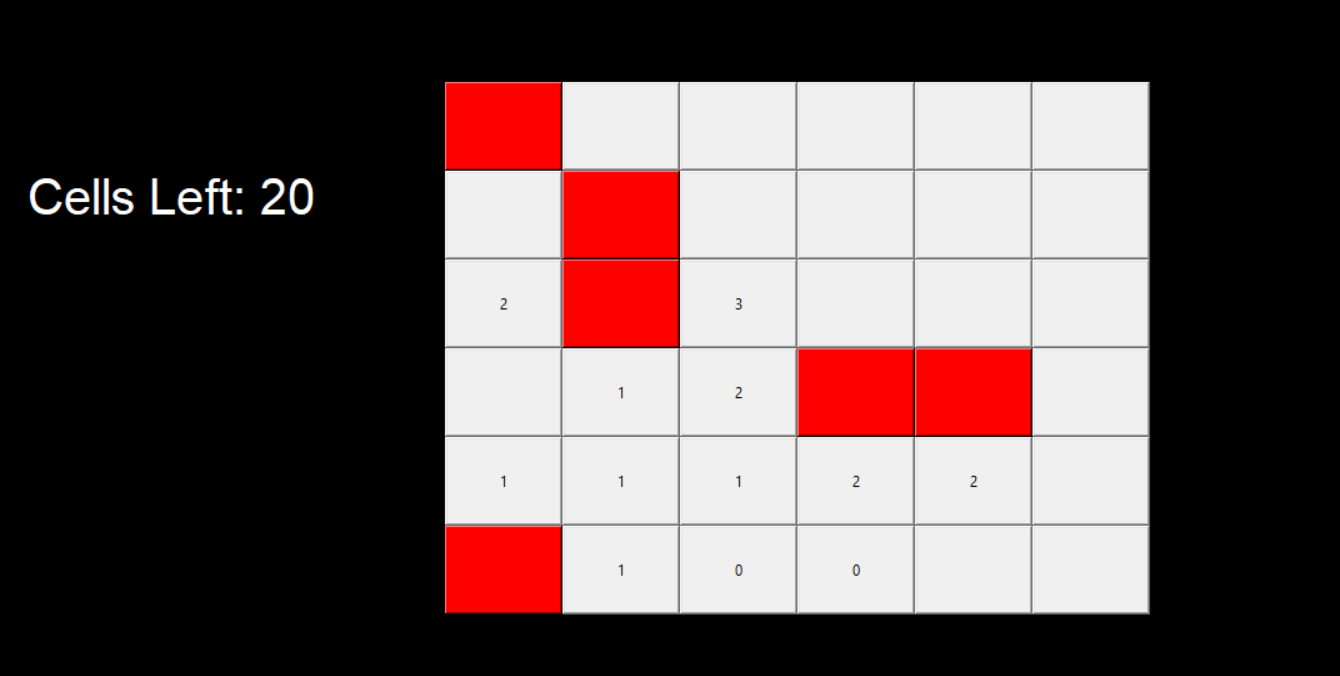
        if Cell.cell\_count\_label\_object:

            Cell.cell\_count\_label\_object.configure(

                text = f"Cells Left: {Cell.cell\_count}"

            )

We also check whether the count is getting reduced properly or not, as the calculation must be updated, due to different cell count when we click on 0 cell



To solve this problem, we set up an attribute which will belong to each cell (is\_open) Boolean type,

We convert this Boolean variable into True, that will mark the cell as open. Then we can condition our program to open , decrease the cell count if the cell is not open yet.

 def show\_cell(self):

        if not self.is\_opened:

            Cell.cell\_count -=1

            self.cell\_btn\_object.configure(text=self.surrounded\_cells\_mines\_length)

            #Repalce the text of cell count label with new count

            if Cell.cell\_count\_label\_object:

                Cell.cell\_count\_label\_object.configure(

                    text = f"Cells Left: {Cell.cell\_count}"

                )

We execute the above code only if the cell is not opened yet.

We get perfect result now as below:

A screenshot of a computer

Description automatically generated

Cell.py

from tkinter import Button, Label

import random

import settings

class Cell:

    all = []

    cell\_count = settings.CELL\_COUNT

    cell\_count\_label\_object = None

    def \_\_init\_\_(self,x, y, is\_mine=False):

        self.is\_mine = is\_mine

        self.is\_opened = False

        self.cell\_btn\_object = None

        self.x = x

        self.y = y

        # Append the object to the Cell.all list

        Cell.all.append(self)

    def create\_btn\_object(self, location):

        btn = Button(

            location,

            width=12,

            height=4,

        )

        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,

            text = f"Cells Left: {Cell.cell\_count}",

            bg = "black",

            fg = "white",

            width = 12,

            height = 4,

            font = ("", 30)

        )

        Cell.cell\_count\_label\_object = lbl

    #this label does not belong to each of our rows.

    #this cannot be an instance method this is a one time , we do not want to call this method for each cell.

    #thats why it needs to be staticmethod, which is for use case of the class, and not for use case of the instance.

    def left\_click\_actions(self, event):

        if self.is\_mine:

            self.show\_mine()

        else:

            if self.surrounded\_cells\_mines\_length == 0:

                for cell\_obj in self.surrounded\_cells:

                    cell\_obj.show\_cell()

            self.show\_cell()

    def get\_cell\_by\_axis(self, x,y):

        # Return a cell object based on the 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\_length(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\_length)

            #Repalce the text of cell count label with new count

            if Cell.cell\_count\_label\_object:

                Cell.cell\_count\_label\_object.configure(

                    text = f"Cells Left: {Cell.cell\_count}"

                )

        #mark the cell as open (Use this as last line  of this method )

        self.is\_opened = True

    def show\_mine(self):

        # A logic to interrupt the game and display a message that player lost!

        self.cell\_btn\_object.configure(bg='red')

    def right\_click\_actions(self, event):

        print(event)

        print("I am right clicked!")

    @staticmethod

    def randomize\_mines():

        picked\_cells = random.sample(

            Cell.all, settings.MINES\_COUNT

        )

        for picked\_cell in picked\_cells:

            picked\_cell.is\_mine = True

    def \_\_repr\_\_(self):

        return f"Cell({self.x}, {self.y})"

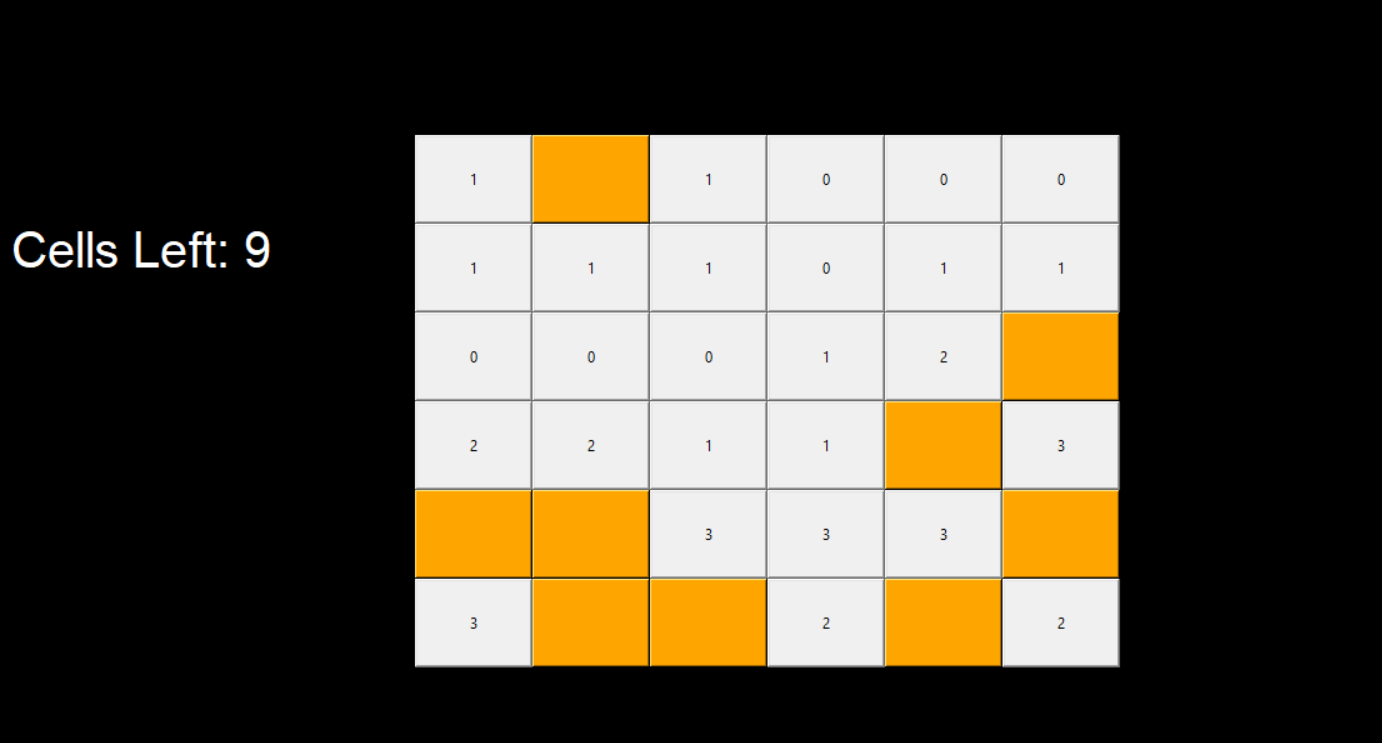
# Last Finishes

* How to mark cells as mine candidates, some additions.

1. To figure out what could be the mines (the cells which we should not click on) , we can develop a feature that will colour the cell in a different colour , so that we can differentiate between the cells and mark as mine candidates.
2. We use right click of the arrow button, so if right click colour will be different, mark as mine candidate and unmark the cell as mine candidate.
3. Create a new attribute for each our cell object

self.is\_mine\_candidate = False

1. We can change the attribute from False to True in right click actions as defined before
2. Check if cell is not a mine candidate, change the background colour of the cell
3. Here if we click on mine cell then the cell count is not changed, which is an important behaviour.
4. We also define the opposite action, after that we can undo the action



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

We should not click on mine with left click.

1. If we use <Button-3> , it might give some problem in that case use different number for Button-3
2. When we click on a mine, we do not finish a game. If we click on a mine, we lose the game.
3. Import “ctypes” library to show the message. If we click on a mine, we throw a message. In show\_mine() we show the message.

def show\_mine(self):

# A logic to interrupt the game and display a message that player lost!

self.cell\_btn\_object.configure(bg='red')

this is where we write the message.

def show\_mine(self):

# A logic to interrupt the game and display a message that player lost!

self.cell\_btn\_object.configure(bg='red')

ctypes.windll.user32.MessageBoxW(0 , "You clicked on a mine ", 'Game Over!', 0)

#generic message box to throw.

* The first string is responsible for body of the message box
* Second text is responsible for title.
* 0 in the last will give only one option to click which is “ok”

A screenshot of a computer

Description automatically generated

* Once we click on mine, we terminate the game, and exit by using sys.exit()

import sys

def show\_mine(self):

# A logic to interrupt the game and display a message that player lost!

self.cell\_btn\_object.configure(bg='red')

ctypes.windll.user32.MessageBoxW(0 , "You clicked on a mine ",'Game Over!', 0)  #generic message box to throw.

sys.exit()

1. To avoid the bug like “if we click on an open cell and it is a mine” it should have its original colour.

Go to show\_cell() and change the background colour to system button face.

def show\_cell(self):

        if not self.is\_opened:

            Cell.cell\_count -=1

            self.cell\_btn\_object.configure(text=self.surrounded\_cells\_mines\_length)

            #Repalce the text of cell count label with new 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"

            )

1. Cancelling the events for the buttons which have already been opened. Use the opposite action of assigning the event.

We assign events for left-click and right-click.

def left\_click\_actions(self, event):

        if self.is\_mine:

            self.show\_mine()

        else:

            if self.surrounded\_cells\_mines\_length == 0:

                for cell\_obj in self.surrounded\_cells:

                    cell\_obj.show\_cell()

            self.show\_cell()

        #cancel left and right click events if cell is already opened:

        self.cell\_btn\_object.unbind('<Button-1>')

        self.cell\_btn\_object.unbind('<Button-3>')

1. We will win the game if amount of cell is equal to amount of mine.

#If Mines count is equal to cells count then player won!

        if Cell.cell\_count == settings.MINES\_COUNT:

            ctypes.windll.user32.MessageBoxW(0 , "Congratulations! You won the game! ", 'Game Over!', 0)

1. Also add a title to the game:

In main.py

Game\_title

game\_title = Label(

    top\_frame,

    bg = "black",

    fg = "white",

    text = "Minesweeper Game",

    font = ("", 48)

)

game\_title.place(

    x = utils.width\_prct(25), y = 0

)

**Game Development Done!**