We used the “tkinter” library for our GUI platform in python. This framework provides Python users with a simple way to create GUI elements using the widgets found in the Tk toolkit. At first place, we need to declare our main windows in tkinter by using this command:

```
top = tk.Tk()
```

It helps to display the root window and manages all the other components of the tkinter application.

After declaring our main windows, we need to demonstrate our main problem; We need to create 4*4 matrix for our memory game which has 16 numbers that are 2 by 2 pairs and in these 16 numbers, we have the numbers 1 to 8.

For generating these 16 numbers, we created the Table_get function. These functions get width and height (as x, y) in input and it returns a 2-D array as our matrix. At first, we needed to create a 1-D array which has 1 to 8 on it and for this, we used np.arange() for it; This function returns evenly spaced values within a given interval. We used it in our code like below:

```
arr_first = np.arange(1,int((x*y)/2)+1)
arr_sec = np.arange(1,int((x*y)/2)+1)
```

By calling this function with those arguments, we created a 2 1-D function which equals to [1,2,3,4,5,6,7,8]. In addition, we will merge them by using the np.concatenate() function.

```
arr_puzzle = np.concatenate((arr_first, arr_sec))
```

Our arr_puzzle variable will be equal to [1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8]; Therefore, we need to shuffle it to get the appropriate result for this function. We do this so that the argument we use in the game is random every time.

```
random.shuffle(arr_puzzle)
```

And at the end of this part, we need to reshape our matrix to 2-D array with shape of 4*4 by using below command:

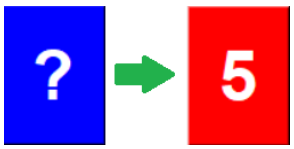
```
arr_puzzle = arr_puzzle.reshape(x,y)
```

After creating this function, we need to create our button; these buttons will be used as our cart which will have our 16 numbers on them. We created a faction for this action and we named it declare(). Every time this function is called, we will have 4*4 table which in any position of it, we have 1 button.

```
button_dict[0]=tk.Button(top, command=lambda: changeColor(button_dict[0]),text="?", font=helv36)
button_dict[0].configure(bg='#00f', fg='#fff')
button_dict[0].grid(row=0,column=0)
```

Every time we create a button, we add it to our button dictionary. Tk.Button is the basic syntax for the declaration. As first argument we give our windows name to it and as second argument we give the function which will be call, every time we click these buttons. We filled it with "?” and we used the helv36 fonts which we declared in our code. By using configure, we can edit out button information. For instance, we change this button color by changing its background color to blue (00f) and its foreground color to white (fff). At the furthest, we used .grid to demonstrate its position in our matrix.

The important part in this project is for the changeColor function which changes the tie button status every time we click on it. Let’s start this function process by assuming that it is our first time we click on our whole buttons. We just need to change this color button to red and replace the text with “?” to its real number which we created in our table_gen function.



We will find to number of each element by using this below code:

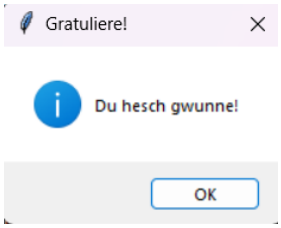
```
value_indx2 = list(button_dict.keys())[list(button_dict.values()).index(btn_counter[len(btn_counter)-1])]
value_btn2 = table[int(value_indx2/4),int(value_indx2%4)]
```

The first line will be the index of our button dictionary in which we store all of the buttons on it and after that by this index we can calculate the position of our element in our 2-D matrix.

In the next step we need to consider this state where the button which we click on isn't our first button. We need to compare this button and previous button for checking if they are the same or not; If they are the same, we will change their color to yellow and lock them.

```
 if(value_btn1 == value_btn2):
    btn.config(text=value_btn1)
    btn_counter[len(btn_counter)-1].config(text=value_btn1)
    btn.config(state = tk.DISABLED)
    btn_counter[len(btn_counter)-1].config(state = tk.DISABLED)
    btn.config(bg='#ff0')
    btn_counter[len(btn_counter)-1].config(bg='#ff0')
```

Every time we find a set of buttons which are the same, we need to increase the “done” variable by one which we declare to find when our game is finished. When this variable is equal to 8, we will find out that we found 8 pairs and after that we will show the below message and restart the game.




```
if done == 8: ##reset part
    done = 0
    tkMessageBox.showinfo(title="Gratuliere!", message="Du hesch gwunne!")
    table = Table_gen(4,4)
    declare()
```

We just need to pay attention that we compared buttons in the upper part by their value, so if we click on a button, twice, in the upper part we consider this button as a pair! So, we need to check that those buttons which have the same values are different objects. We can easily can handle this state like this:

```
if(btn_counter[0] == btn):
    btn.configure(bg='#00f', fg='#fff')
    btn_counter[len(btn_counter)-1].configure(bg='#00f', fg='#fff')
    btn.config(text="?")
    btn_counter[0].config(text="?")
```

Those lines will be considering this state and will change this button which we click on to the default status (color = blue and text =?).

In the last state we need to consider the situation in which those 2 buttons aren't the same; we will reset their state to our default again.

If this button were not the same, we won’t change those colors until we click on another button.

```
if (len(btn_counter) == 2):
 btn_counter[len(btn_counter)-2].config(text="?")
 btn_counter[len(btn_counter)-1].config(text="?")
 btn_counter[len(btn_counter)-2].config(bg='#00f', fg='#fff')
 btn_counter[len(btn_counter)-1].config(bg='#00f', fg='#fff')
```

These are our functions which we created for our project with them. We just need to call bellow values to start our game:

```
table = Table_gen(4,4)
declare()
top.mainloop()
```