College of Computer and
Cyber Sciences

**University of Prince Mugrin**

PROJECT REPORT

# Project Name: Matched the Colored Blocks

Group Members

4110718

**Mohammed Umer Raheemuddin**

3910249

**Alobaidi, A. Abdullah**

Date of submission: 10/12/2020

# Abstract

In this report we will show our process in creating our game: Matched the Colored Blocks.

Our motivation for the project was to create an application that gives a user small mental exercise by memorization. The technical reason we wanted the project was to code an application that the user interacts. Our methodology is simple, to make the program as dynamic as possible for the user and the programmer. For example, the application uses randomization to keep each experience unique for the user. Furthermore, the user can decide how many blocks do they want by changing the number of colors. Also, we made sure to write our code in an organized way. We used functions to separate each part of our game, so it is easier to edit. Besides, our program does not use hard-coded ways to perform. As a result, we have a game that keeps the user experience different and makes the program pleasant to edit.

# Contents

# 1. Introduction

The program divided into two different sections: The GUI and matching the blocks. The GUI needs to be done first and then write the code to match the blocks. We mainly used what we learned from the class, but for the GUI, we had to use online sources like Stack overflow and Real python. We had to use online sources because GUI was first. The purpose of this report is to provide a comprehensive look at the program's functionality.

# 2. Background

We had no previous experiences with GUI, and some of us in python. As such, we wanted to improve our python programming skills by creating a dynamic application that uses not only basic python fundamentals but also includes GUI. For GUI, we used the built-in Tkinter library because of its simplicity and popularity, because it was easier to solve many problems. We also used the built-in library Random library to create a unique experience for the user in each game.

## 2.1 Project brief

The main purpose of this project was to create a game in which the user interacts to match the same-colored blocks. Create a game that gives the user a memorization exercise.

# 3. Project Management

We had about one and a half months to complete our project. However, we had some changes in the group, and we also had midterm exams, so this delayed our project a little bit. Umar was responsible for the GUI and the report as he was more familiar with GUI. Abdullah was responsible for coding the recognition of the colors and the presentation as he was more familiar with the python fundamentals. We both also assisted each other in our tasks.

We divided the program into two major parts: the GUI and matching the same-colored blocks. Firstly, we had to do the GUI first because the GUI was the foundation of our program, where we could see our solutions to match the same-colored blocks were working or not. Secondly, we created randomly generated positions because we can base our solutions for the program on randomly generated positions. Thirdly, we then revised our code to increase the efficiency of our program by removing many unnecessary code lines. Finally, we then did the main calculation for our programs.

Then in the final week, we did the report and the presentation. Umar did the report, and Abdullah did the presentation.

# 4. Application Details

## 4.1 Overview

We divided the application into two main sections: the GUI and matching the same-colored blocks. But first, we need to import the required libraries and declare the variables.

Importing the required libraries for our program.

```
from tkinter import *
from tkinter import messagebox
from random import choice
```

We then declared the required variables for our game.

```
colors_list = ["red", "blue", "green", "cyan", "magenta", "pink", "orange",
"yellow"]
button_number = 0
total_colors = []
stored_colors = []
row = 0
column = 0
canvas_list = []
matched_canvas = []
```

- Color_list is the list of all possible unique colors.
- Button_number is the total blocks present in the game.
- Total_colors is all the colors of the game in order of the button_number.
- Stored_colors store the colors from the blocks which the user has picked.
- Matched_canvas stores the canvas in which the user has matched the same colors.

## 4.2 GUI

The GUI contains three parts: buttons, canvases, and a For loop. We used buttons to create an unknown block which the user clicks to find the color. We then used canvases to display the colors on top of the buttons.

Firstly, we need to create a class that makes the program organized. The __init__ is method creates the main window of the game.

```
class main_window:
    def __init__(self):
        self.root = Tk()
        self.root.title("Main Window")
```

### 4.2.1 Buttons

We used a function because we needed multiple buttons. A canvas is created on the location of the button when clicked. We also used a grid to have more control over the placement of both canvases and buttons. We used Lambda to initiate the canvas function. Lambda creates an undefined function within the button canvas. We used Lambda because we wanted the program to place all the buttons first then all the canvases.

```
def button(self, column, row, button_number, color, store):

    button = Button(text=("Button", button_number), bg="black",
                    width=19, height=9,
                    command=lambda: [
                        self.canvas(column, row, color, canvas_list, store,
matched_canvas)
```

```
                    ], activebackground=color)
    button.grid(column=column, row=row)
```

### 4.2.2 Canvases

We created a function for a canvas because the position of the button is the same as the canvas.

```
  def canvas(self, column, row, color, canvas_list, stored_colors,
matched_canvas):
    canvas = Canvas(self.root, bg=color, width=139, height=142)
    canvas.grid(column=column, row=row)
```

### 4.2.3 For Loop

We then need to create a For loop that creates multiple buttons and canvases. But first, we need to create the main window by using the main_window class.

```
window = main_window()
```

After that, we create a For loop that creates multiple buttons and canvases. But first, the For loop needs to assign the position of the buttons and canvases randomly. We can do this by multiplying the colors_list by two to create duplicate colors in the list. Then use the choice module to pick a color from the color_list and assign it to the color variable.

The multiplied colors_list should look like this.

```
          ["red",  "blue",  "green",  "cyan",  "magenta",  "pink",  "orange",
  "yellow",  "red",  "blue",  "green",  "cyan",  "magenta",  "pink",  "orange",
  "yellow"]
```

```
for color in colors_list * 2:

    # Then the for loop randomly picks a color from the color_list.

    color = choice(colors_list)

    # Then the function no_more_than_two_colors is called to check and
randomly replace the same color.

    color = window.no_more_than_two_colors(color)
```

The method in the window class checks if the color is no more than two colors in the list and then replaces any color that picked more than twice.

```
def no_more_than_two_colors(self, color):
    while total_colors.count(color) > 1:
        color = choice(colors_list)
    return color
```

Then we create an If statement, which places the buttons and canvases in the first four columns of the window and then places the buttons and canvases in the first four columns of the next row. The process repeats until each color from the duplicate color_list is assigned to a button and a canvas.

```
if column % 4 == 0:
    row += 1
    column = 0
```

After column, row, and color are ready to be sent to the button function. Then, assign the value of the next column to the column variable and then store colors in the total_list. The total_list is in order according to the button number. We will talk about stored_colors later.

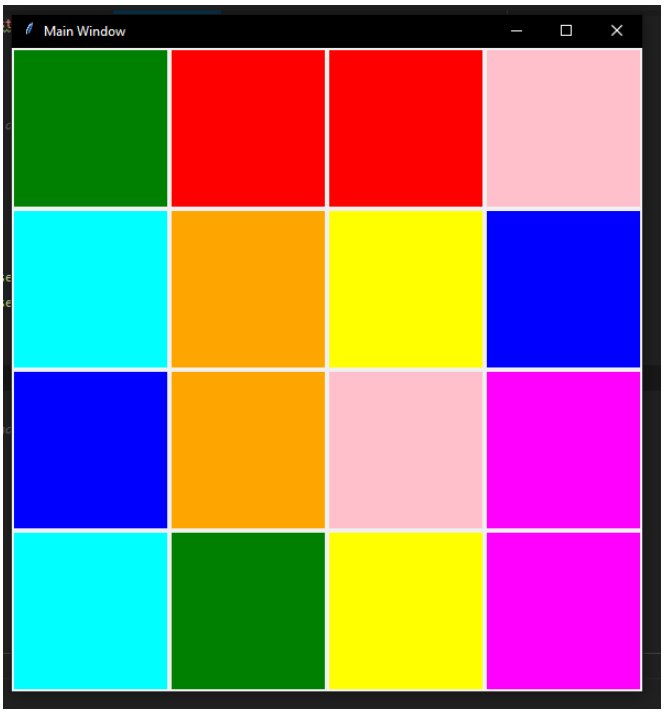
```
window.button(row, column, button_number, color, stored_colors)

# Calculate the next column

column += 1

# The total_colors stores the colors in the order of the button_numbers

total_colors += [color]

# Button_number is the total blocks present in the game.

button_number += 1
```

Finally, print the total_colors and display the window. The total_colors is not necessary for the game; it is just a cheat sheet for the game.

```
  print(total_colors)
window.root.mainloop()
```

The program should look like this:


When we click all the buttons, we should see something like this.




But the program cannot recognize the same-colored blocks. In the next section, we will add the code to recognize the same-colored blocks.

## 4.3 Matching the Same Colored Blocks

We divided the matching of the same colored blocks into two parts: recognizing the same colored blocks and resetting the incorrect block.

### 4.3.1 Recognizing the Same Colored Blocks

When we were assigning the columns, rows, and colors to each button and canvas. We also sent the stored_colors, canvas_list, and matched_canvas to the canvas function. In the canvas function, we have created nested if statements to recognize the same colors.

Before that, we need to create two lists that store the colors and canvases the user has clicked.

```
canvas_list.append(canvas)
stored_colors.append(color)
```

After the user has clicked two buttons, we then created an if statement that checks if the first color in the stored_colors is the same as the second color in the stored_colors.

If the first color in the stored_colors is the same as the second color in the stored_colors, then display a message to the user that the colors match and store the canvases of those colors in the matched_colors.

```
if len(canvas_list) == 2:

    if stored_colors[0] == stored_colors[1]:

        messagebox.showinfo("Colors Match", "Colors Matched Successfully")

        matched_canvas += canvas_list
```

### 4.3.2 Deleting the Incorrect Blocks

If the blocks do not match. Display a message to the user that the colors do not match. After that, create a For loop, which removes the canvases from matched_colors in the canvas_list. We do this because we do not want to delete the canvas which the user has matched correctly.

```
else:
    messagebox.showinfo("Color Match", "Colors do not match")

    for one__canvas in matched_canvas:

        if one__canvas in canvas_list:

            canvas_list.remove(one__canvas)
```

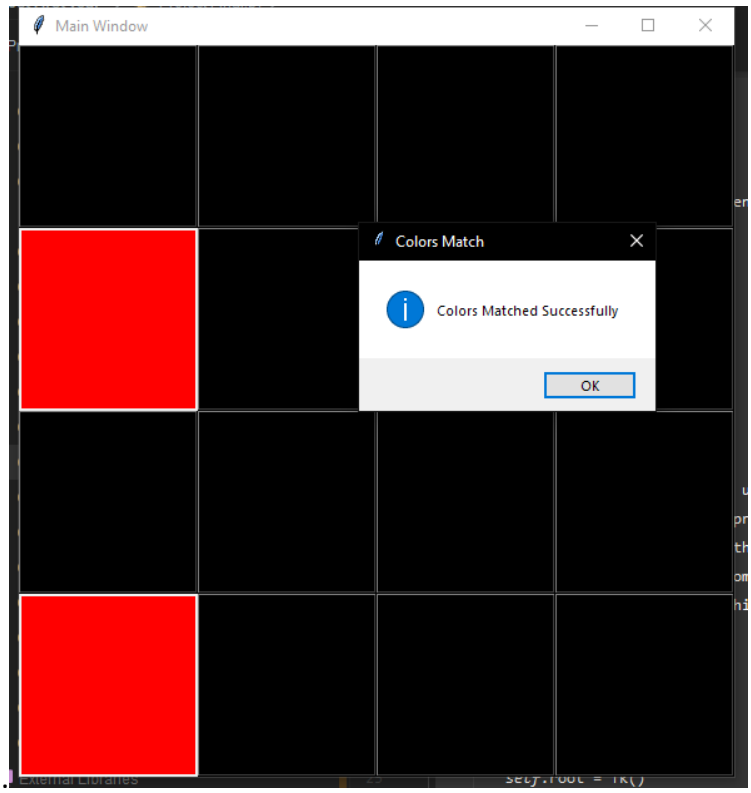
Finally, we have the canvases that the user did not match correctly. We need to reset the canvases that the user has picked incorrectly by resetting the canvas_list and stored_colors and returning the two lists.

```
        for each_canvas in canvas_list:
            each_canvas.destroy()

    canvas_list.clear()
    stored_colors.clear()
return canvas_list, matched_canvas
```

# 5. Results

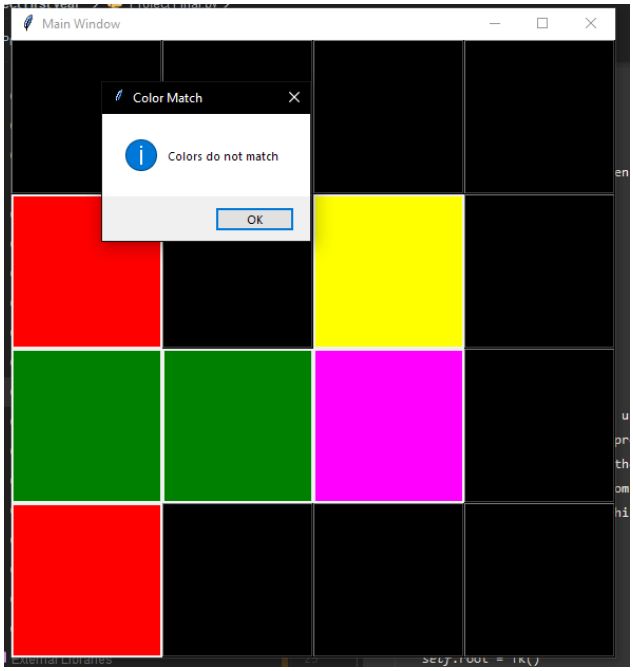
## 5.1 Same-Colored Blocks

Our application is now able to recognize the same-colored blocks and display a message to the user that the colors match




## 5.2 Blocks Do Not Match

When the blocks do not match, the application should display a message to the user that the blocks do not match.




The game then removes the incorrect blocks, and then the user has to match with different blocks to proceed.

# 6. Challenges Faced

We have faced many challenges in the programming part and collaborating parts of the project. The issue we have faced in the collaborating part was the inability to meet in person because Umar lives in Riyadh and Abdullah lives in Madinah. To overcome this issue, we have met with each other in Microsoft Teams and arranged weekly meetings about the project. However, this made it difficult for us to communicate with each other, thus causing a delay by one week of our expected completion date. As for the programming part, we had difficulty in coding the removal of the canvas when the blocks do not match. We overcame this problem by creating a list of canvases. We also had to do the GUI first, so we had learned from online sources to complete the project in time.

# 7. Future Work

## 7.1 Future Ideas

We wanted to include randomly generated colors in the game. However, the colors generated were similar to each other. That made it difficult for a human to differentiate the colors. The challenge was taking a considerable amount of time, so we have dropped the idea.

## 7.2 Bugs

The only issue we have face was that if the user clicked a button and moved the cursor. The program would display the color and does not register the clicked button as an input from the user.

# 8. Conclusion

The project was challenging for both of us. From the code to the report and presentation, we had to plan thoroughly. Although we were unable to complete the randomly generated colored blocks, we are quite satisfied with the results. We have also learned the knowledge about GUI and python fundamentals that we can apply in our future projects.