Master Mind Project:

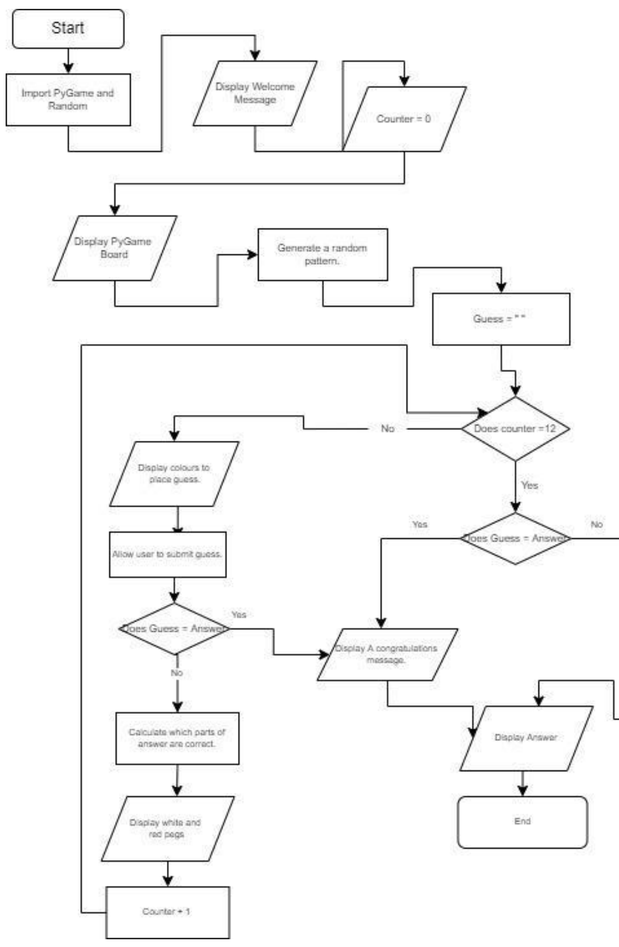
Analysis:

Breaking down the Mastermind Game into components:

1. Displaying a Welcoming message.
2. Displaying the Board and Hiding the solution.
3. Randomly generating the 4 coloured Pegs.
4. Allowing the User to place a guess.
5. Cross referencing against the result.
6. Placing down white pegs for correct pegs but wrong place in random holes.
7. Placing down red pegs for correct pegs in the right place in the remaining random holes.
8. Removing 1 of the counter of guesses the user is allowed.
9. Allowing the user to guess again.
10. Keep repeating the process of 4-9 until the user guesses it correctly or until they have used up all their guesses.
11. Display correct solution.
12. Present either a congratulation message or try again message.

Functions that need to be imported:
- Random Function – To randomly generate colours.
- PyGame – To create a display board.

Flow chart to represent the computation of the code:
Link to image if it appears too distorted:
https://app.diagrams.net/?src=about#LMasterMind%20Flow%20Diagram



Problem Statement for Game:
The aim of the game is to be able to predict a randomly generated sequence of 4 pegs which could possibly have different colours. The 4 pegs which are assigned a colour and position at the beginning of the game are hidden from the user and the user has 12 guesses to try and figure out the correct pegs, colour and order of the pegs. After each guess the computer will compare it with the answer and will generate white and red pegs to help the user with their next guess. White pegs indicate that the user has a correct colour but in the wrong position, a red peg indicates that one of the pegs is the right colour and in the right position. However, the catch to these pegs is that they are not in the correct order so you have to deduct in further guesses which colours were correct and which ones were in the correct positions. The challenge of the game is to complete the challenge in the fewest moves possible. If you succeed in guessing the pattern correctly before you have had all 12 guesses a congratulatory message will be displayed and the number of guesses it took will also be displayed, if you are unfortunate enough not to be able to guess the pattern within 12 guesses a good attempt message will be displayed on the screen and the correct pattern that was previously hidden will be revealed so the user can figure out how close they were to the real solution.

My goals for this project:
- To create a running version of mastermind.
- To begin to understand the capabilities of PyGame and to explore the graphics and potential it holds.
- To extend myself to create a graphical version of mastermind.
- To reduce the amount of possible errors to as small as possible.
- To create an effective and enjoyable version of mastermind.
- To extend myself to create a leader board of scores using data handling techniques to extend myself further.
- To create a reflective and honest evaluation of my project and suggest different techniques and goals I would consider for future projects.

Techniques that will be beneficial to use and coding style:
For an extensive project such as this it is important that I maintain a good coding style and ensure that my code is as efficient, readable, bug free and concise as possible to aid both my understanding of the code and anyone else who may be looking at the code. Techniques that will help me to achieve this will be an intensive use of sub-programs e.g. functions and procedures to reduce the quantity of code required and to add a level of organisation to the code. Due to PyGame requiring some objects there will be occasional lines of object orientated programming within the code and this technique will be necessary for the display and projecting anything to the display is most cases. The use of comments and indentations within the code will also add to the organisation of the code and will ensure that I know the function of each section of my code. Naming variables and sub routines with sensible and easily recognisable names will reduce the time it takes to code my project as well as helping to ensure that the code is maintainable. As I intend to code this in python installed on my laptop installing PyGame as an extension of python will be necessary and creating an organised coding folding within my own files will minimise the risk of errors occurring as a result of files being stored in external locations to the file where the main program is being stored.

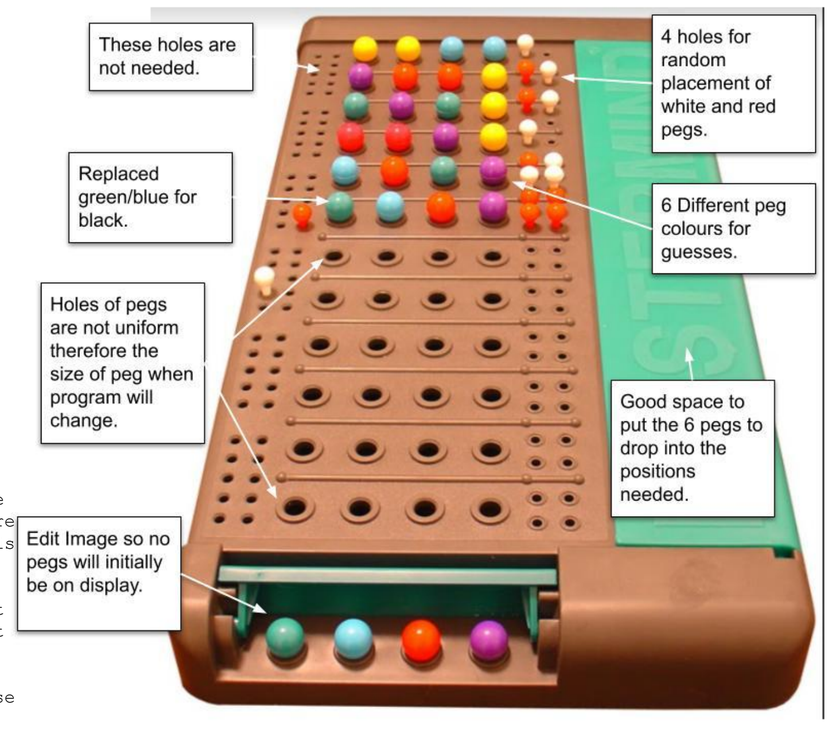
Design:

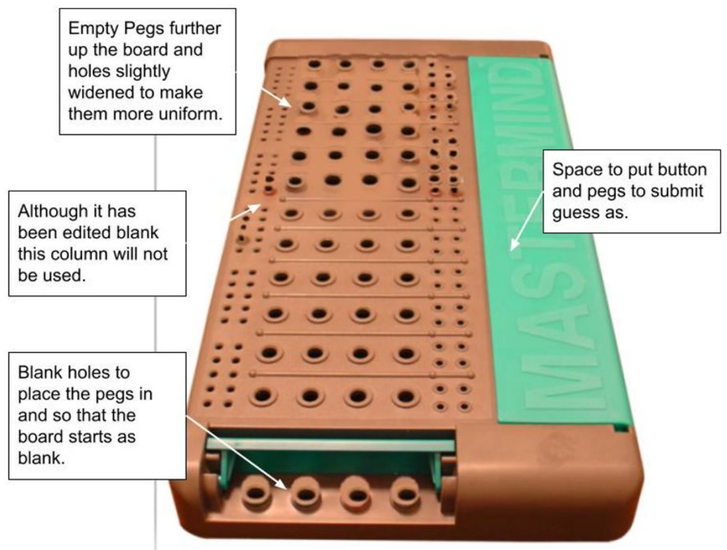
Within this problem I intend to create a version of mastermind that is graphically supported with a mastermind board, coloured pegs which the user can use, buttons and messages that display on the screen. It would be possible to create a version of mastermind within an array and a basic 2D design but for my project I desire to create the most realistic version within my ability. Within the design brief we were provided with an image of a mastermind board that already had some coloured pegs displayed upon it and therefore my first stage within the project was to create a board that was blank so that I could manipulate it fully when coding. However, this will not be the only graphical element of the project, coloured pegs will also be a key feature of my project.

Key Graphical elements:
- Blank Mastermind board.
- Coloured Pegs – 8 colours .
- Messages on screen.
- Possible Leader Board

Key colours within the graphics:
- White – RGB value: 255,255,255
- Black – RGB value: 000,000,000
- Blue – RGB value: 112, 206, 226
- Orange – RGB value: 242, 101, 36
- Pink – RGB value: 242, 110, 113
- Purple – RGB value: 114, 61, 151
- Red – RGB value: 205,092,092
- Yellow – RGB value: 255,255,000

Problems that could occur with the graphics:
- Editing the image provided will require a period of time and risks reducing the quality of the image present.
- The peg holes decrease in size the further up the board you are so the code will have to allow for a change in size of the peg which will require manipulation in order to achieve this and will make the code less concise to achieve this.
- Creating a button on the screen that is functioning and making it so that the user can create their input by placing there pegs in certain locations will be a challenge because it is unlike anything I have ever done before and this will make it more challenging.





Edited Graphics:
I took the original mastermind project graphics provided to us in the project brief and I was able to edit out the original pegs as well as being able to make it so that some of the holes were marginally more uniform and there would be a less drastic change in size for the pegs. I also managed to ensure that there was still space available on the other side to ensure that you could store the pegs which would become the input system for the pegs as well as for the button.

Algorithms within the design:
Algorithms are used consistently when coding however due to the scale of this project and the number of different tasks to complete within it algorithms will be used to ensure that code is concise and functions. There are many different instances when algorithms can be used in this project e.g. creating a random number, comparing whether or not the guess is the same as the answer, Algorithm to convert the colour of the best to a numerical value in order to be able to compare them, and the use of algorithms when repeating through up to 12 counts of guesses for the user.

What is an algorithm?
An algorithm can simply be defined as the process of following rules or instructions during a period of problem solving. We use algorithms because it aids us when doing arithmetically and helps to prevent any member creating incorrect code logically and the use of algorithm means the code is more usable as it can be transferred to other code and with the lack of specific data within the equation it means that the code can be used more than once in more than one instance which becomes a time saver especially when within loops.

Technical Implementation:

Using the same game break down used when analysing the code, screen shots of the code are provided in that order with a brief description of the code and key elements of the code. Due to poor quality when attempting to attach screenshots the corresponding images can be found on the page attached.

1. This first screen shot is evidence of creating a display screen as well as creating a welcome message that displays for four seconds before the screen resets ready to import the Mastermind board. The key techniques needed for this was to import the time module to have access to the times commands and using some very basic object orientated programming to create a display screen which is set to the size of the board. Key features needed to display to the screen are RGB colour values, X-start co-ordinates, Y-start co-ordinates, size and font. It was at this point that I also captioned the entire game to be Mastermind so that is a user lost the tab they could identify what it was from the caption.
2. The second screen shot is the code to display the graphic I edited to present a blank Mastermind board, to do this I had to initially load the image into python and ensure that the file for the image was stored in the same folder as the main program code for the program. And then it followed a similar process to displaying the welcome message however this time there was no need to remove the message from the screen.
3. The next stage in forming the program was to create the answer as well as assigning all the colour values which I would use in the future. Storing the pattern in Array allows me to display the values when needed and reduces the lines of code required to be written.
4. The next stage of the coding process proved to be far more challenging to complete which was allowing a user to place an input on their prediction for the placement of pieces. I decided that the best way to do this was to create buttons out of the holes in the board as buttons that when clicked on changed colour based on a function that looped the possible values around. I then created an additional black button in order to submit the users guess at the placement of colours. To do this I needed to find the properties of the buttons as well as the area they covered in order to determine whether the mouse had clicked a button and if so which button they had hit. Furthermore, I had to ensure that you could not submit another guess after the original guess had been entered and ensure that the loop was broken. To store my prediction I created an array that stored the RGB colour values for each peg ready to compare in the next programming step.
5. To compare the Answers I used string and list handling techniques to manipulate the two array so that I could compare each element within them and establish whether or not they were all identical in the right place or if they weren't identical and the user hadn't established the correct pattern. After comparing each element and establishing if they were identical or not I began to include the code for steps 6 and 7 to figure out the number of pegs and type of pegs needed to give the user a clue on the layout of the pattern as using the function to do this was more efficient and would save time long term.
6. Although it is listed as step six I chose to complete seven before six and I could therefore use the data from seven to help me calculate the number of white pegs that would be needed within the program and using the data from seven I could also plot their location as most of it was already provided in seven I simply had to rearrange the data around.
7. The next step was to calculate the number of red pegs that needed to be displayed and where the correct location to display them was to do this I used trial and error to find appropriate locations. You cannot physically have more than three red pegs as otherwise the solution would be solved and therefore I only needed 3 elements to my if statement. I decided to code it within a function so that I can reuse the code later on within the program and to be more efficient as a programmer.
8. To ensure that the user doesn't get more than 12 attempts at guessing the solution I created a counter to restrict the number of guesses and every time you guess the counter increases and a while loop will prevent it from going over 12 guesses.
9. By placing the entire code for guessing one solution within a while loop that operates as long as the counter is under 12 or the answer hasn't been found continues repeating the entire solution and comparisons of each bit of data until either the 12 guess limit has been reached or the correct solution has been found. Using an array of different x and y co-ordinates stored within my program I can precisely plot each point within the diagram and the code will work as a repeat without me having to make any additional edits.
10. By ensuring that at the end of every guess I check that the number of guesses is still below the maximum number of guess it ensures that no additional errors will present within the code and that the game is fair for all who play it.
11. I used another function to displaying the correct solution to the problem that will appear on the screen for 10 seconds and allows the user to confirm that they are right if they got it within the allotted number of guesses or if they did not reach the correct solution it provides visualisation for where it went wrong and why it is wrong.
12. Displaying a message with a very similar format to the welcome message it makes it obvious to the user whether or not they were correct and provides confirmation. This confirmation begins the conclusion of the program and refreshing the screen to display a message effectively resets the board and makes it obvious that they have completed a round of Mastermind and can determine there success of it.
13. An additional element that I included of my own was to include an external leader board of the scores with the smallest number of guesses being the desired outcome. I allowed the user to enter their name and then saved it to an external list of scores. I then imported this list and arranged them from lowest to highest scores and project the top 5 scores onto the screen so that the user can see the main competition.

All of the code is available to view on the page attached with some labelled with specific sections to highlight elements of the code.

In total 300 lines of code were written for this project and 3 libraries were imported into the python program. You require both the shell and the display screen in order to be able to fully use the game and in future it may have been beneficial to create an input in the display of the actual program however, the addition of a leader board makes the game more unique and allows users to be able to interact with each other and motivate them to continue playing the game.

Testing and Evaluation:

Error Testing For:	Data Used	Type of Data	Expected Outcome	Outcome	Actions needed:
Data Type	Str(input(123)	Integer	"123!	"123"	N\a - This will be used for the leader board.
Type Error	Display(heigh, width, size) *testing when two required	Parameters	Type Error – too many parameters	Type error	No action needed as long I test all functions have correct number of parameters.
Logic Error	Area_rect = o.5 * base * height	Acceptable data	No error produced Area of square calculated	No error produced Area of triangle given instead.	Logic Errors are harder to identify but I need to be aware when I receive unexpected outcomes to check the logic.
Syntax Error	If counter = number:	Appropriate data	Analyses whether counter is the same value as the variable number.	Systematic error – assignment operator instead of comparison.	Ensure that == is for comparing values and = for assigning values, essentially spell checking code.
Name Error	B = 7 print(a)	Erroneous Data	Name Error	Name Error	Ensure variables are appropriately named.

Testing and error preventing:
To prevent errors in my code and to make my code more resistant to erroneous data I used certain techniques to eliminate the possibilities of errors. For example when letting the user input there button choices I did not set any of the values to white I set it to a colour that one of the pegs were instead a colour that was not available as it reduces an invalid guess being made and reduces the number of errors I have to proof my code against. Another way I prevented errors in my code when taking inputs was to ensure that when entering a name for the leader board that it was entered as a string even though they could technically enter a number meaning that if they set there username to be a number that it would not affect the format of the leader board but would just be outputted as a numerical name. When coding this problem testing as I went was the most crucial thing to the success of the project as by commenting out sections of code to find bugs and errors in the code and being able to improve the logic of the code without having to waste time on additional procedures that I have already tested or do not currently need to be tested. However, a main element to making my code error proof was limiting the user inputs to an extent that the game was still enjoyable and the user felt involved, but also didn't create a large amount of possible errors that I would have to try and prevent.

Evaluation of project:
Summarising the effectiveness of the project as well as my own individual handling of the project I would say was done well. I am proud to have been able to extend myself to create a leader board that was not included as part of the project brief and was a unique element to the code that allows the user to interact more with it an create a competition as the game itself is not a multiplayer game and by allowing a leader board it introduces a greater element of a competition within the game which will encourage the user to play the game more than once. However, some elements of the code could also be stronger as using x and y co-ordinates for the location of the coloured pegs is effective however, I did not use this idea for the white and red pegs and the further you move down the board they are tending to become more and more off-centre from the holes of pegs and where they should actually be placed upon the board, I have made an attempt to remedy this by slightly adjusting the x-co-ordinate each time you analyse the closeness of the guess made by the user to the actual answer of the program however, in future finding a more symmetrical board would be more beneficial and would greatly improve the visuals of the code and how users can interact with it. Whilst conducting the programming project I have found that creating a flow chart and decomposing the problem as far as possible has been highly beneficial to my approach to the program and has allowed me to be more organised about how I am coding my solutions and the order in which I do so that means I have to make fewer corrections and the process was in general more efficient overall. Nevertheless, I believe that I could have analysed problems that I could encounter better such as the x and y co-ordinates of the board as well as the way in which I would code for the number of white and red pegs proved to be surprisingly tricky so if I had analysed the possible problems I would encounter more I may have been able to approach these problems better and done more research of coding techniques that would solve this problem. Overall the code I produced fit the project brief for Mastermind and was effective with some added elements to make the game more complex and interesting to interact with. If I was to redo this entire project I would look into dragging and dropping pins into the correct solution to make it more realistic and to add a greater presence of a user input within the code. I additionally would look into making a game replay button so that the user did not have to reload the code every time that they wanted to play again and look at creating a unique leader board so that they can see their progress within a game. To further extend the project I would consider creating a login system and sign up system so that the user can have a personal account and so that they will have a permanent record of their personal best, number of games played, success and losses etc so that it was more interactive and to prevent unauthorised access to the system and to the game. I would still include an option for a user to play a guest as well though. Overall I am pleased with the outcome of the project and have a greater understanding for how I would approach the next problem.

By Hannah Johnson
Due: Wednesday the 19th of January 2022
Dr Grant Computer Science A-level

```python
#Importing the necessary libraries into python.
import pygame
import random
import time

#Initialising the game screen and captioning it as Master Mind
pygame.init()
gameDisplay = pygame.display.set_mode((600, 637))
pygame.display.set_caption('Mastermind.')

#Setting the screen to a colour to display a welcome message and updating the screen.
colour = (145, 184, 255)
gameDisplay.fill(colour)
pygame.display.flip()

#Creating A function to display a welcome message.
def WelcomeMessage():
#Setting the font, size and text as well as the positioning of the text.
    Welcome_font = pygame.font.Font('freesansbold.ttf',36)
    Welcome_text = Welcome_font.render("Welcome to MasterMind, ",True,(0,0,0))
    Welcome_text1 = Welcome_font.render("I hope you enjoy the game!",True,(0,0,0))
    gameDisplay.blit(Welcome_text,((70), (317)))
    gameDisplay.blit(Welcome_text1,((50), (357)))
    pygame.display.update()
#Using the time module setting how long the message displays for and then resetting the screen.
    time.sleep(4)
    gameDisplay.fill(colour)
    pygame.display.flip()

WelcomeMessage()
```

1. This is the code for setting up a screen and displaying a welcome Message.

```python
#Importing the edited graphic of the Mastermind board and updating the screen.
image = pygame.image.load('MasterMindBoard.jpg')
gameDisplay.blit(image, (0, 0))
pygame.display.update()
```

2. This is the code for setting up a screen and displaying the Mastermind board onto the screen.

```python
#Generating the 4 ball sequence.
#Setting Colours for the future
blue = (112, 206, 226)
orange = (242, 101, 36)
pink = (242, 110, 113)
purple = (114, 61, 151)
red =(205,92,92)
white = (255,255,255)
black = (0,0,0)
yellow = (255,255,0)

#Setting up an array to store the answer
Answer = []

#Creating four values for the four pegs
for j in range(4):
    num = random.randint(1, 6)
    if num == 1:
        num = blue
    elif num ==2:
        num = orange
    elif num ==3:
        num = pink
    elif num ==4:
        num = yellow
    elif num ==5:
        num = black
    else:
        num = purple
    Answer.append(num)
```

3. This is the code for determining the patter of the four pegs which are the solution and storing them as an array.

4. The code for allowing the user to enter one input and allowing them to submit this input through the use of buttons and a loop of colours. Also ensures that the guess is recorded so that it can be compared to the answer in the next step.

```python
#Drawing circles onto the game board.
#X and Y co-ordinates of the peg holes
Xco_ords1 = [188,228,268,308,186,226,266,306,186,226,266,306,179,219,259,299,170,214,258,302,170,214,258,302,165,211,257,303,155,205,255,305,147,200,
Yco_ords1 = [44,69,94,119,148,177,215,251,291,337,385,436]

#Setting the counter to 0
counter = 0

#Function needed to print the circles to the screen in the right position. Yellow as the default colour.
def printRow(counter):
    circles = counter * 4
    circle_rect1 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[counter]),10, 0)
    x1 = Xco_ords1[circles]
    circles += 1
    circle_rect2 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[counter]),10, 0)
    x2 = Xco_ords1[circles]
    circles += 1
    circle_rect3 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[counter]),10, 0)
    x3 = Xco_ords1[circles]
    circles += 1
    circle_rect4 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[counter]),10, 0)
    x4 = Xco_ords1[circles]
    circles += 1
    y1 = Yco_ords1[counter]
    pygame.display.update()
    return circle_rect1,x1,circle_rect2,x2,circle_rect3,x3,circle_rect4,x4,y1

#Creating a loop of colours for the button
def getcolour(click):
    colour = [yellow, blue, orange, pink, purple,black]
    length = len(colour)
    if click>(length-1):
        click = 0
    col = colour[click]
    return click, col

#Drawing the submit button onto the screen.
circle_rect5 = pygame.draw.circle(gameDisplay, black, (450,500),20, 0)
pygame.display.update()
```

5. The code needed to check if the guess is correct against the answer and beginning to calculate the number of pegs required. This code has been edited since to update calculating the number of pegs.

```python
#Function to establish if the answer is correct.
def checkAnswer(Answer, Guess, counter):
#Setting the necessary variables to 0.
    num_white_pegs = 0
    num_red_regs = 0
    ArrayCounter = 0
#Checking if they are identical.
    if Answer[0] == Guess[0] and Answer[1]==Guess[1] and Answer[2]==Guess[2] and Answer[3] == Guess[3]:
        correct = True
        return correct
#Beginning to calculate the pegs if they are not identical.
    else:
        counter+=1
        for k in range(4):
            if Answer[ArrayCounter] ==Guess[ArrayCounter]:
                num_red_pegs+= 1
        ArrayCounter = 0
        for l in range(3):
            if Answer[ArrayCounter] == Guess[(ArrayCounter+1)]:
                num_white_pegs+=1
        ArrayCounter = 1
        for n in range(2):
            if Answer[ArrayCounter] == Guess[(ArrayCounter+1)]:
                num_white_pegs+=1
        ArrayCounter = 2
        if Answer[ArrayCounter] == Guess[(ArrayCounter+1)]:
            num_white_pegs+=1
```

7. Placing down the correct red pegs in the needed spaces and calculating how many red pegs are needed. Last Photo is more efficient.

```python
#Displaying Red Pegs:
def display_red_pegs(checkanswer,x4, y1):
    num_red_pegs = checkanswer
    print(num_red_pegs)
    if num_red_pegs ==1:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        pygame.display.update()
    elif num_red_pegs ==2:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, red, ((x4+(27+9)), (y1-3)),6,0)
        pygame.display.update()
    else:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, red, ((x4+(27+9)), (y1-3)),6,0)
        x4 = x4 -10
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1+10)),6,0)
        pygame.display.update()
```

```python
#Checking if they are identical.
    if Answer[0] == Guess[0] and Answer[1]==Guess[1] and Answer
        correct = True
        return correct
#Beginning to calculate the pegs if they are not identical.
    else:
        if Answer[0] ==Guess[0]:
            num_red_pegs +=1

        if Answer[1] ==Guess[1]:
            num_red_pegs +=1
        if Answer[2] ==Guess[2]:
            num_red_pegs +=1
        if Answer[3] ==Guess[3]:
            num_red_pegs +=1
        return num_red_pegs
```

```python
#Beginning to calculate the pegs if they are not identical.
    else:
        '''if Answer[0] ==Guess[0]:
            num_red_pegs +=1

        if Answer[1] ==Guess[1]:
            num_red_pegs +=1
        if Answer[2] ==Guess[2]:
            num_red_pegs +=1
        if Answer[3] ==Guess[3]:
            num_red_pegs +=1
        return num_red_pegs'''
        for w in range(4):
            if Answer[w] == Guess[w]:
                num_red_pegs += 1
                used[w] = True
        return num_red_pegs
```

```python
#Displaying White Pegs:
def display_white_pegs(num_white_pegs,x4, y1):
    num_red_pegs = checkanswer
    print(num_red_pegs)
    if num_white_pegs ==1:
        x4+=10
        pygame.draw.circle(gameDisplay, white, ((x4+27+9), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==2:
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1+10)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==3:
        pygame.draw.circle(gameDisplay, white, ((x4+(9+37)), (y1-3)),6,0)
        pygame.draw.circle(gameDisplay, white, ((x4+(27)), (y1+10)),6,0)
        x4 = x4 +10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==4:
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1-3)),6,0)
        x4 = x4 -10
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1+10)),6,0)
        x4 = x4 +10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()
```

6. Placing down the correct white pegs in the needed spaces and calculating how many red pegs are needed.

```python
#Calculating the number of white pegs needed.
def displaying_white_pegs(used, Answer, Guess):
    num_white_pegs = 0
    for z in range(4):
        for e in range(4):
            if not used[e] and Guess[e] == Answer[z]:
                num_white_pegs += 1
                used[e] = True
    return num_white_pegs
```

```python
        pygame.display.update()
elif circle_rect5.collidepoint(x,y):
    Guess = [colour1, colour2, colour3, colour4]
    print(Guess)
    counter+= 1
    break
```

8 + 9 When coding for the submit input button I created an addition to the counter so that it wouldn't go above 12 and the standard guess would be given.

10. Allowing the user to guess more than once and to begin to deduce how to solve the pattern

```python
    circle_rect1,x1, circle_rect2,x2, circle_rect3,x3, circle_rect4,x4,y1 = printRow(counter)
    while counter ==c:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                # Set the x, y postions of the mouse click
                x, y = event.pos
                if circle_rect1.collidepoint(x, y):
                    click1+=1
                    click1, colour1 = getcolour(click1)
                    pygame.draw.circle(gameDisplay, colour1, (x1, y1),10, 0)
                    pygame.display.update()

                elif circle_rect2.collidepoint(x, y):
                    click2+=1
                    click2, colour2 = getcolour(click2)
                    pygame.draw.circle(gameDisplay, colour2, (x2, y1),10, 0)
                    pygame.display.update()
                elif circle_rect3.collidepoint(x, y):
                    click3+=1
                    click3, colour3 = getcolour(click3)
                    pygame.draw.circle(gameDisplay, colour3, (x3, y1),10, 0)
                    pygame.display.update()
                elif circle_rect4.collidepoint(x, y):
                    click4+=1
                    click4, colour4 = getcolour(click4)
                    pygame.draw.circle(gameDisplay, colour4, (x4, y1),10, 0)
                    pygame.display.update()
                elif circle_rect5.collidepoint(x,y):
                    Guess = [colour1, colour2, colour3, colour4]
                    print(Guess)
                    counter +=1
                    next
```

11. Displaying the solution.

```python
try:
    checkanswer,used = checkAnswer(Answer, Guess, counter)
except:
    for h in range(4):
        num =Answer[(h-1)]
        if num == yellow:
            num  = 4
        elif num == orange:
            num =2
        elif num == pink:
            num =3
        elif num ==purple:
            num =6
        elif num == black:
            num =5
        elif num == blue:
            num = 1
        DisplayColouredPeg(num, locationx[(h - 1)], locationy[(h - 1)])
        CongratsMessage()
        number = number + 12
        name = str(input("Please enter your name for the leaderboard"))
        score = counter
        with open ("Leaderboard.txt","a") as f:
            writeto = str(score) + "," + name +"\n"
            f.write(writeto)
            f.close()
        scores,names = read_scores("Leaderboard.txt")
        zips = sort_scores(scores,names)
        print_scores(zips)
        pygame.quit()

    display_red_pegs(checkanswer,x4,y1)
    num_white_pegs = displaying_white_pegs(used, Answer, Guess)
    display_white_pegs(num_white_pegs,x4,y1)
    pygame.display.update()



for h in range(4):
    num =Answer[(h-1)]
    if num == yellow:
        num  = 4
    elif num == orange:
        num =2
    elif num == pink:
        num =3
    elif num ==purple:
        num =6
    elif num == black:
        num =5
    elif num == blue:
        num = 1
    DisplayColouredPeg(num, locationx[(h - 1)], locationy[(h - 1)])

time.sleep(5)
GoodAttemptMessage()
pygame.quit()
```

12 and 13. Code to display welcome messages and the code for the leader board(functions can be found in the copy of the whole program.)

```python
#Importing the necessary libraries into python.
import pygame
import random
import time

locationx = [121, 180, 242, 302]
locationy = [557, 558, 560, 560]

def DisplayColouredPeg(num, locationx, locationy):
    if num == 1:
        pygame.draw.circle(gameDisplay, blue, (locationx, locationy), 19, 0)
        pygame.display.update()
    elif num == 2:
        pygame.draw.circle(gameDisplay, orange, (locationx, locationy), 19, 0)
        pygame.display.update()
    elif num == 3:
        pygame.draw.circle(gameDisplay, pink, (locationx, locationy), 19, 0)
        pygame.display.update()
    elif num ==4:
        pygame.draw.circle(gameDisplay, yellow , (locationx, locationy), 19, 0)
        pygame.display.update()
    elif num ==5:
        pygame.draw.circle(gameDisplay, black , (locationx, locationy), 19, 0)
        pygame.display.update()
    else:
        pygame.draw.circle(gameDisplay, purple, (locationx, locationy), 19, 0)
        pygame.display.update()

#Initialising the game screen and captioning it as Master Mind
pygame.init()
gameDisplay = pygame.display.set_mode((600, 637))
pygame.display.set_caption('Mastermind.')

#Setting the screen to a colour to display a welcome message and updating the screen.
colour = (145, 184, 255)
gameDisplay.fill(colour)
pygame.display.flip()
number = 0
#Creating A function to display a welcome message.
def WelcomeMessage():
#Setting the font, size and text as well as the positioning of the text.
    Welcome_font = pygame.font.Font('freesansbold.ttf',36)
    Welcome_text = Welcome_font.render("Welcome to MasterMind, ",True,(0,0,0))
    Welcome_text1 = Welcome_font.render("I hope you enjoy the game!",True,(0,0,0))
    gameDisplay.blit(Welcome_text,((70), (317)))
    gameDisplay.blit(Welcome_text1,((50), (357)))
    pygame.display.update()
#Using the time module setting how long the message displays for and then resetting the scre
    time.sleep(4)
    gameDisplay.fill(colour)
    pygame.display.flip()

#Creating A function to display a congratulations message.
def CongratsMessage():
#Setting the font, size and text as well as the positioning of the text.
    colour = (145, 184, 255)
    gameDisplay.fill(colour)
    pygame.display.flip()
    Welcome_font = pygame.font.Font('freesansbold.ttf',36)
    Welcome_text = Welcome_font.render("Congratulations!",True,(0,0,0))
    gameDisplay.blit(Welcome_text,((140), (317)))
    pygame.display.update()
#Using the time module setting how long the message displays for and then resetting the scre
    time.sleep(4)
    gameDisplay.fill(colour)
    pygame.display.flip()


def GoodAttemptMessage():
#Setting the font, size and text as well as the positioning of the text.
    colour = (145, 184, 255)
    gameDisplay.fill(colour)
    pygame.display.flip()
    Welcome_font = pygame.font.Font('freesansbold.ttf',36)
    Welcome_text = Welcome_font.render("Good Attempt!",True,(0,0,0))
    gameDisplay.blit(Welcome_text,((140), (317)))
    pygame.display.update()
#Using the time module setting how long the message displays for and then resetting the scree
    time.sleep(4)
    gameDisplay.fill(colour)
    pygame.display.flip()

WelcomeMessage()

#Importing the edited graphic of the Mastermind board and updating the screen.
image = pygame.image.load('MasterMindBoard.jpg')
gameDisplay.blit(image, (0, 0))
pygame.display.update()

#Generating the 4 ball sequence.
#Setting Colours for the future
blue = (112, 206, 226)
orange = (242, 101, 36)
pink = (242, 110, 113)
purple = (114, 61, 151)
red =(205,92,92)
white = (255,255,255)
black = (0,0,0)
yellow = (255,255,0)

#Setting up an array to store the answer
Answer = []

#Setting up num of white and red pegs to 0.
num_white_pegs = 0
num_red_pegs = 0

for j in range(4):
    num = random.randint(1, 6)
    if num == 1:
        num = blue
    elif num ==2:
        num = orange
    elif num ==3:
        num = pink
    elif num ==4:
        num = yellow
    elif num ==5:
        num = black
    else:
        num = purple
    Answer.append(num)


#Drawing circles onto the game board.
#X and Y co-ordinates of the peg holes
Xco_ords1 = [188,228,268,308,186,226,266,306,186,226,266,306,179,219,259,299,170,214,258,302,
Yco_ords1 = [44,69,94,119,148,177,215,251,291,337,385,436]

#Setting the counter to 0
counter = 0
correct = False
#Function needed to print the circles to the screen in the right position. Yellow as the defa
def printRow(counter):
    circles = counter * 4
    circle_rect1 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[cou
    x1 = Xco_ords1[circles]
    circles += 1
    circle_rect2 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[cou
    x2 = Xco_ords1[circles]
    circles += 1
    circle_rect3 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[cou
    x3 = Xco_ords1[circles]
    circles += 1
    circle_rect4 = pygame.draw.circle(gameDisplay, yellow, (Xco_ords1[circles], Yco_ords1[cou
    x4 = Xco_ords1[circles]
    circles += 1
    y1 = Yco_ords1[counter]
    pygame.display.update()
    return circle_rect1,x1,circle_rect2,x2,circle_rect3,x3,circle_rect4,x4,y1

#Creating a loop of colours for the button
def getcolour(click):
    colour = [yellow, blue, orange, pink, purple,black]
    length = len(colour)
    if click>(length-1):
        click = 0
    col = colour[click]
    return click, col

#Function to establish if the answer is correct.
def checkAnswer(Answer, Guess, counter):
    used =[False,False,False,False]
#Setting the necessary variables to 0.
    num_white_pegs = 0
    num_red_pegs = 0
    ArrayCounter = 0
#Checking if they are identical.
    if Answer[0] == Guess[0] and Answer[1]==Guess[1] and Answer[2]==Guess[2] and Answer[3] ==
        correct = True
        return correct
#Beginning to calculate the pegs if they are not identical.
    else:
        for w in range(4):
            if Answer[w] == Guess[w]:
                num_red_pegs += 1
                used[w] = True
        return num_red_pegs,used

#Drawing the submit button onto the screen.
circle_rect5 = pygame.draw.circle(gameDisplay, black, (450,500),20, 0)
pygame.display.update()

#Making it so that when a circle is clicked it changes colour.
circle_rect1,x1, circle_rect2,x2, circle_rect3,x3, circle_rect4,x4,y1 = printRow(counter)
'''Guess = []
click1 = 0
click2 = 0
click3 = 0
click4 = 0
colour1 = yellow
colour2 = yellow
colour3 = yellow
colour4 = yellow
'''
#Displaying Red Pegs:
def display_red_pegs(checkanswer,x4, y1):
    x4 = x4+2
    num_red_pegs = checkanswer
    if num_red_pegs ==1:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        pygame.display.update()
    elif num_red_pegs ==2:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, red, ((x4+(27+9)), (y1-3)),6,0)
        pygame.display.update()
    elif num_red_pegs==3:
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, red, ((x4+(27+9)), (y1-3)),6,0)
        x4 = x4 -10
        pygame.draw.circle(gameDisplay, red, ((x4+27), (y1+10)),6,0)
        pygame.display.update()

#Calculating the number of white pegs needed.
def displaying_white_pegs(used, Answer, Guess):
    num_white_pegs = 0
    for z in range(4):
        for e in range(4):
            if not used[e] and Guess[e] == Answer[z]:
                num_white_pegs += 1
                used[e] = True
    return num_white_pegs
print(Answer)
def display_white_pegs(num_white_pegs,x4, y1):
    x4 = x4 + 2
    num_red_pegs = checkanswer
    print(num_red_pegs)
    if num_white_pegs ==1:
        x4+=10
        pygame.draw.circle(gameDisplay, white, ((x4+27+9), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==2:
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1+10)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==3:
        pygame.draw.circle(gameDisplay, white, ((x4+(9+37)), (y1-3)),6,0)
        pygame.draw.circle(gameDisplay, white, ((x4+(27)), (y1+10)),6,0)
        x4 = x4 +10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()
    elif num_white_pegs ==4:
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1-3)),6,0)
        x4 += 10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1-3)),6,0)
        x4 = x4 -10
        pygame.draw.circle(gameDisplay, white, ((x4+27), (y1+10)),6,0)
        x4 = x4 +10
        pygame.draw.circle(gameDisplay, white, ((x4+(27+9)), (y1+10)),6,0)
        pygame.display.update()



def read_scores(filename, splitter=','):
    """reads scores and names from a file, and returns a list of each"""
    scores = []
    names = []

    with open(filename) as f:
        for score in f:
            score, name = score.strip().split(splitter)
            scores.append(int(score))
            names.append(name)
    return scores, names

def sort_scores(scores, names, reverse_bool=True):
    """sorts the scores from greatest to least and returns in a list of tuples format"""
    return sorted(zip(scores,names))

def print_scores(score_list, splitter=' ', top_amount=5):
    """prints the number of leaderboard scores stated"""
    for score_tuple in score_list[:top_amount]:
        print(splitter.join(map(str, score_tuple)))



#Creating the code for the counter = 0 instance to make it so the user can enter at least 1 g

for c in range(0,12):
    Guess = []
    click1 = 0
    click2 = 0
    click3 = 0
    click4 = 0
    colour1 = yellow
colour2 = yellow
colour3 = yellow
colour4 = yellow

circle_rect1,x1, circle_rect2,x2, circle_rect3,x3, circle_rect4,x4,y1 = printRow(counter)
while counter ==c:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            # Set the x, y postions of the mouse click
            x, y = event.pos
            if circle_rect1.collidepoint(x, y):
                click1+=1
                click1, colour1 = getcolour(click1)
                pygame.draw.circle(gameDisplay, colour1, (x1, y1),10, 0)
                pygame.display.update()

            elif circle_rect2.collidepoint(x, y):
                click2+=1
                click2, colour2 = getcolour(click2)
                pygame.draw.circle(gameDisplay, colour2, (x2, y1),10, 0)
                pygame.display.update()
            elif circle_rect3.collidepoint(x, y):
                click3+=1
                click3, colour3 = getcolour(click3)
                pygame.draw.circle(gameDisplay, colour3, (x3, y1),10, 0)
                pygame.display.update()
            elif circle_rect4.collidepoint(x, y):
                click4+=1
                click4, colour4 = getcolour(click4)
                pygame.draw.circle(gameDisplay, colour4, (x4, y1),10, 0)
                pygame.display.update()
            elif circle_rect5.collidepoint(x,y):
                Guess = [colour1, colour2, colour3, colour4]
                    print(Guess)
                    counter +=1
                    next


    try:
        checkanswer,used = checkAnswer(Answer, Guess, counter)
    except:
        for h in range(4):
            num =Answer[(h-1)]
            if num == yellow:
                num  = 4
            elif num == orange:
                num =2
            elif num == pink:
                num =3
            elif num ==purple:
                num =6
            elif num == black:
                num =5
            elif num == blue:
                num = 1
            DisplayColouredPeg(num, locationx[(h - 1)], locationy[(h - 1)])
            CongratsMessage()
            number = number + 12
            name = str(input("Please enter your name for the leaderboard"))
            score = counter
            with open ("Leaderboard.txt","a") as f:
                writeto = str(score) + "," + name +"\n"
                f.write(writeto)
                f.close()
            scores,names = read_scores("Leaderboard.txt")
            zips = sort_scores(scores,names)
            print_scores(zips)
            pygame.quit()

    display_red_pegs(checkanswer,x4,y1)
    num_white_pegs = displaying_white_pegs(used, Answer, Guess)
    display_white_pegs(num_white_pegs,x4,y1)
    pygame.display.update()


for h in range(4):
    num =Answer[(h-1)]
    if num == yellow:
        num  = 4
    elif num == orange:
        num =2
    elif num == pink:
        num =3
    elif num ==purple:
        num =6
    elif num == black:
        num =5
    elif num == blue:
        num = 1
    DisplayColouredPeg(num, locationx[(h - 1)], locationy[(h - 1)])

time.sleep(5)
GoodAttemptMessage()
pygame.quit()
```