Mastermind in Pygame

Matthew McPherson
A Level Computer Science
Year 12

The Rules of Mastermind

The board of Mastermind has 4 columns and 8 - 12 rows. These are used for cracking the colour code of the pegs. The cover at the end of the board is used to cover the correct code that you are trying to crack. The *codemaker* makes the pattern to be cracked, which will be cracked by the *codebreaker*.

The codebreaker places 4 coloured pins on the board in the first row. This will then give an indication of the correct and incorrect colours that they have guessed. This is given through the feedback pegs:

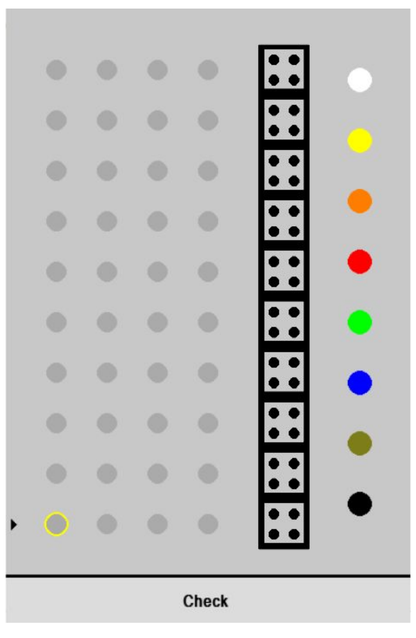
- A coloured or black peg is placed for each peg on the board which is the correct colour and position.
- A white peg indicated the existence of a correct colour peg on the board in the incorrect position.

From this feedback, another guess is made and the cycle continues until the code is either cracked or the codebreaker runs out of guesses.

Interface

Designing the interface was simple. I took the approach of making the most simple to understand interface which would be the easiest to program. This resulted in a simple 2D representation of the board.

I achieved this by simply drawing and moving polygons when required to represent the game. This only took simple draw commands. This turned out very successfully.



If I were to remake this, I would have added more functions and buttons to be able to have some more generated abilities of the game, such as 2 player or repeated colours on and off etc.

Design

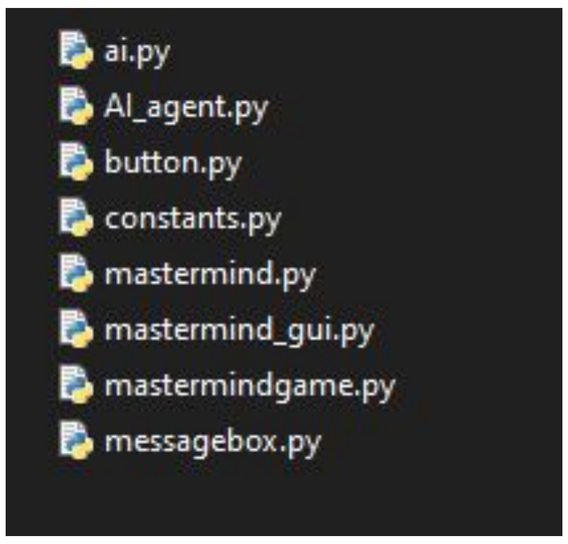
Designing the required algorithms for the game to work was the most challenging part of the whole project. In hindsight, I think programming this regularly in Python would have made life far easier. I was trying to develop the code and the visual implementation at the same time. This made it more difficult than it could have been.

The first step I took was designing the AI to choose a set of random colours. This is done without duplicates and using the Python ‘random’ library. It simply picks a random number between 1 and 8, each representing a colour. These colours are then the colours that will have to be guessed in the set positions.These colours and stored in variables of each position. Every time a guess is made, the position stored colour is compared with the guess of the player. Depending on the guess colour and location, the indicator pins will be updated to display if the answer is correct and in the correct position etc.

This was the main concept of the code. The rest of it lay around debugging, setting the limits of the program, pop-ups, and much of the other ‘boring’ code. Overall, it turned out to be pretty successful. I would attempt to make it 2 player in the future, but I found myself struggling for time and ability to make instances of player switching vs the AI simply choosing the random colours to be used.

Technical Implementation

Through research of other sources in making this, I found it easier to use multiple files to program the interface and game as a whole. It allowed me to break down and focus the task into different sections. Abstracting the problem like this also meant that my file was not thousands of lines. This made navigating and rewriting parts of the program easier, despite linking them and making them work completely fluidly more difficult at times.



When it comes to the development of the buttons, they are simply linked to the function of code which corresponds to that button. For example, if you press the ‘guess’ button, the button holds the function to execute the code to compare the colours that the player has placed on the board to the colours that the computer has generated. Creating the GUI for the buttons was the more difficult task, as it involved the learning and discovering of how to position and colour a correct button as the correct shape and the correct size.

```
def draw_background():
    for x in X_POS:
        for y in Y_POS:
            pygame.draw.circle(screen, GREY, [x, y], 10)
    pygame.draw.rect(screen, GREY, [250, 75, 50, 500])
    for y in Y_POS:
        pygame.draw.rect(screen, DARKGREY, [255, y - 21, 40, 40])
        draw_pinquarter(0, 0, 275, y)

    pygame.draw.rect(screen, GREY, [25, 25, 200, 40])
    for k in range(1, 9):
        y = 50 + k * 60
        pygame.draw.circle(screen, COLOR[k], [350, y], 12)

    myfont.set_bold(True)
    pygame.draw.circle(screen, COLOR[helpcolor], [350, 40], 26)
    helplabel = myfont.render("?", 1, BLACK)
    textpos = helplabel.get_rect()
    textpos.centerx = 350
    textpos.centery = 40
    screen.blit(helplabel, textpos)
    myfont.set_bold(False)
```

Expanding what has been said above, the algorithm for the AI to decide the colours simply used the random library to get a random number, assign it to a colour which is then passed into the main body of code for the user inputs to be compared to.

```
def __init__(self, colours=8, bins=4, repeat=False):
    self.history = []
    self.bins = bins
    self.colours = [str(k) for k in range(1, colours + 1)]
    self.repeat = repeat
    if not repeat:
        self.solution = random.sample(self.colours, self.bins)
        self.possibles = list(itertools.permutations(self.colours, self.bins))
        self.possibles = [list(sam) for sam in self.possibles]
    else:
        self.solution = [str(random.randint(1, colours)) for _ in range(bins)]
        self.possibles = [[str(i), str(j), str(k), str(l)]
                          for i in range(1, colours + 1)
                            for j in range(1, colours + 1)
                              for k in range(1, colours + 1)
                                for l in range(1, colours + 1)]
    self.practice = False
```

I decided to use more functions and more standard variables than all classes. There are some classes used in this project, but a surprising amount of the project was successful in the use of simple functions. As for the colours, this had to be class based. This is because it would become very confusing and hard to link with the use of just variables. It would also begin to become inefficient and far more bulky. It would have been easier to program as just functions and variables, but as the code develops, the complexity means that it would be very easy to get lost in an almost sea of variables.

```
def draw_game():
    draw_pins()
    draw_buttons()
    draw_current_row()
    draw_current()
    if mes['won'] is True:
        draw_messages('YOU WON')
        if not had_help:
            add_message('%i points (%i sec, %i moves)' % (score, t2-t1, row+1))
    elif mes['pins'] is True:
        draw_messages('Choose four colors')
    elif mes['lost'] is True:
        draw_messages('YOU LOST')
    elif mes['show_leader'] is True:
        show_leaderboard()
    if mes['show_solution'] is True:
        show_solution()
    else:
        poslabel = myfont.render("%i possibilities" % possibilities, 1, BLACK)
        screen.blit(poslabel, (40, 30))
        if run_time:
            t = time.time() - t1
        else:
            t = t2 - t1
        minut = int(t / 60)
        sec = int(t % 60)
        timestring = "{:}:{:0>2d}".format(minut, sec)
        timelabel = myfont.render(timestring, 1, BLACK)
        screen.blit(timelabel, (170, 30))
```

Conclusion

Overall, I would say that this project was successful. It taught me some valuable lessons on problem solving and programming at a higher level than I was before. I still think that my object oriented programming needs some work. Conclusively however, the project was a success!