Henry Miller
A-Level Computer Science
Year 12

Mastermind project
(Pygame)

What is Mastermind?

Mastermind is a code breaking game which involves two people to play. It starts with deciding which role the two people will play as, the codebreaker or the code maker. Once the roles have been decided the code maker can begin to create the code. The code is made on a board using coloured pins and is hidden from the code breaker. This code is made up from 6 different coloured pins but only 4 pins can be used for the code which is then covered by a shield so that the codebreaker cannot see the code. Once the code is made the codebreaker gets 8-12 turns in order to get the right code using the decoder board. Here the code breaker will use the coloured pins in different configurations to try and get the code makers code. The code maker gives feedback for the code by placing 4 pins back to represent the codebreakers code, red for correct and white for incorrect. If the code breaker runs out of attempts then the code maker wins and the roles swap until one of the people wins the majority of the games that was determined at the start of the game.

Rules-

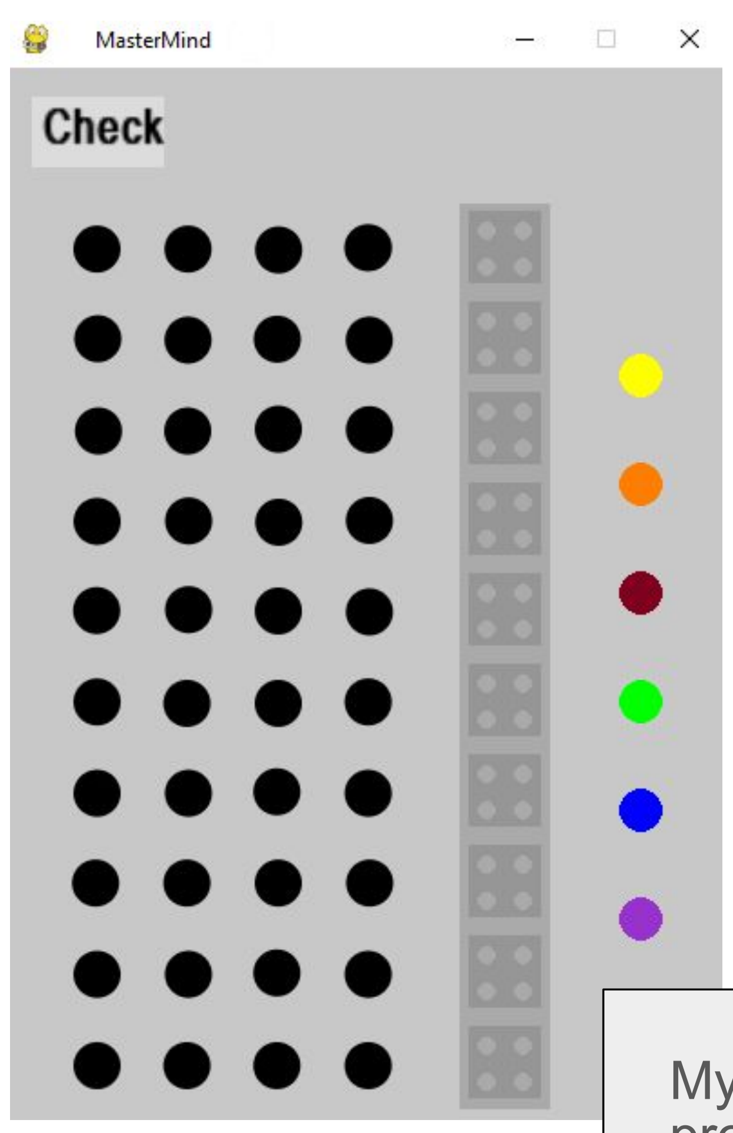
- The two players must decide to how many games they will play in order to find a winner (this has to be even and will swap each time the code is either guessed or the codebreaker runs out of attempts).
- The decoding board is used to create the code of pins and for the codebreaker to input their attempts at cracking the code.
- The code maker must give input back to the codebreaker each attempt to help aid the code breaker into getting the correct order.
- Only 6 main colours must be used to create the 4 pin code.
- The code must be 4 pins long
- The code maker must make the code on the board to make sure they don't change the code mid-game.

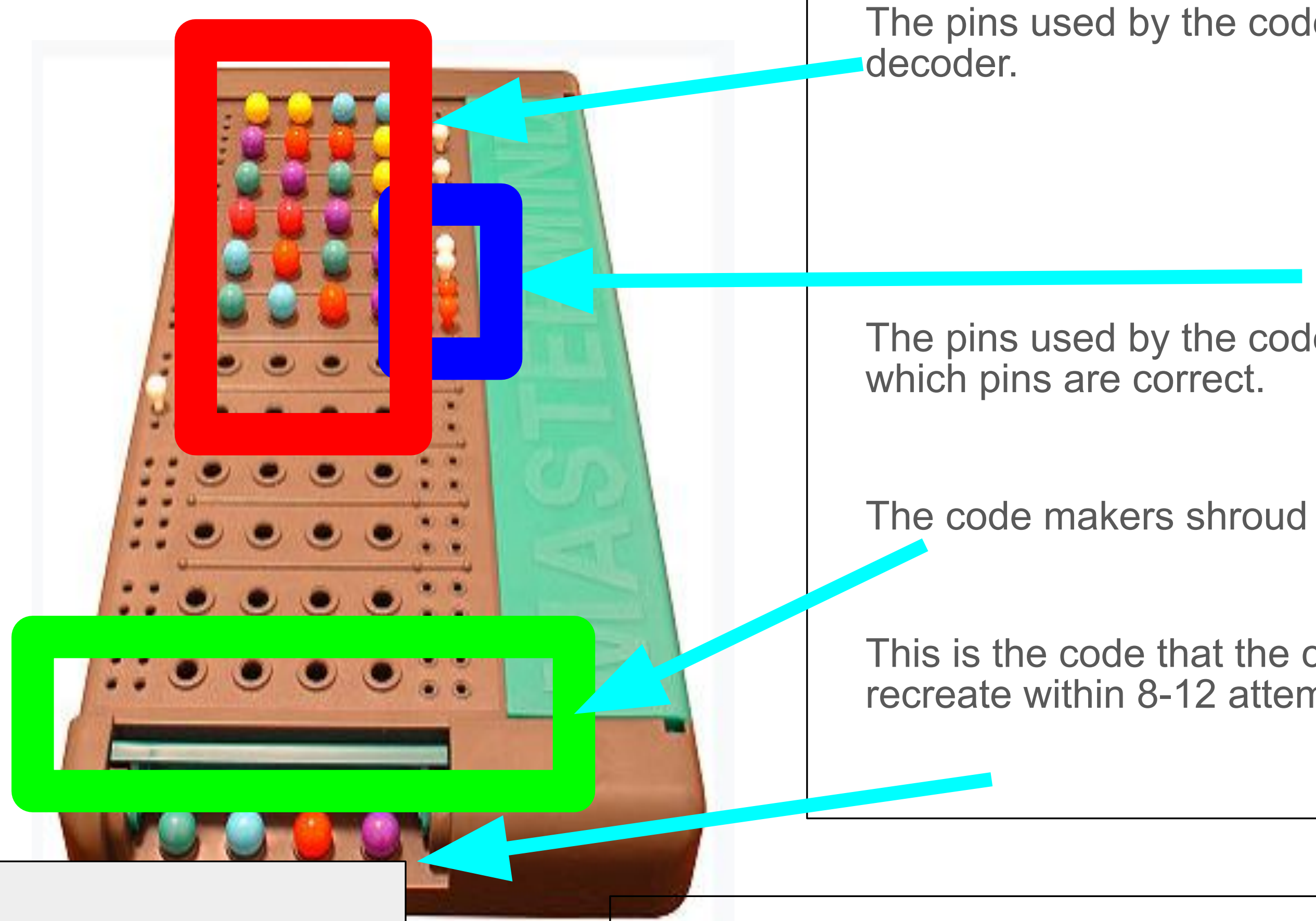
Conclusions-

What I would do differently next time would be to create a simple python version of Mastermind as it would allow me to think of the logic behind it without having to implement it into a new language with visuals. This would speed the process up a lot and make me more comfortable with what I would have to do in Pygame rather than figuring it out along the way. I would also have made the game two player but where I was making it on the fly it was an oversight so the game runs as the user vs computer.

However, by using lots of functions and separate documents to control different parts of the code I was able to make a reliable version of Mastermind in Pygame whilst keeping the code clean and easily readable. This is something that I will take forward when on whatever language.

The board and pins-





This is the board (decoder) that is used to play the game. The code breaker sits facing the top and the code maker sits facing the bottom of the board.

The pins used by the code breaker placed into the decoder.

The pins used by the code maker to give feedback on which pins are correct.

The code makers shroud

This is the code that the code breaker will need to recreate within 8-12 attempts.

My Mastermind project in Pygame (top left image).

Here is a mastermind board, we can see the code makers code at the bottom and the code breakers code is at the top. He's used 6 of his 12 attempts. On the right we can see the code maker giving feedback to whether the pins are correct (white) and incorrect (white). The green, blue, red and purple are correct so the code maker has added the red pins to the board to show they are correct.

```
 for event in pygame.event.get():
       if event.type == pygame.QUIT:
          done = True
       elif event.type == pygame.KEYDOWN:
          if event.key == pygame.K_KP_ENTER or event.key ==
pygame.K_RETURN:
             if run_time:
                event_button_pressed()
             else:
                toggle_message()
          elif event.key in [pygame.K_1, pygame.K_2, pygame.K_3,
pygame.K_4, pygame.K_5, pygame.K_6, pygame.K_7,
pygame.K_8]:
#
             update_grid(event.key-48)
          elif event.key == pygame.K_DELETE or event.key ==
pygame.K_BACKSPACE:
             undo_last()
```

The code to the left shows the main loop. In this you can see that it manages all of the events to do with the buttons the user may press like the X to close the program. Keydown is used to take input from the user using a keyboard for example, the user cna press backspace which will then delete their pin selection.

```
                self.select_username_button = button.Button(left, 50, width, height,
    GREY, BLACK, 'select username', BLACK)
            self.select_ai_button = button.Button(left, 150, width, height, GREY,
    BLACK, 'select AI', BLACK)
            self.start_game_button = button.Button(left, 450, width, height, GREY,
    BLACK, 'start game', BLACK)
            self.stop_game_button = button.Button(left, 550, width, height, GREY,
    BLACK, 'stop game', BLACK)
```

This is the code for creating the buttons. This works by selecting the width and height and colour of the button. These buttons are used for the selection.

This is the code for the colours of the pins. Saving them as a RGB value and then assigning each colour to a number.

```
WHITE = (255, 255, 255)
YELLOW = (255, 255, 0)
ORANGE = (255, 130, 0)
RED = (128, 0, 32)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
Purple = (153, 50, 204)
BLACK = (0, 0, 0)
GREY = (175, 175, 175)
SILVER= (150, 150, 150)
BACKGROUND = (200, 200, 200)
```

```
COLOUR = {}
COLOUR[1] = WHITE
COLOUR[2] = YELLOW
COLOUR[3] = ORANGE
COLOUR[4] = RED
COLOUR[5] = GREEN
COLOUR[6] = BLUE
COLOUR[7] = Purple
COLOUR[8] = BLACK
```

Code for each of the colors need for the pin and backgrounds / buttons.

```
for i,key in
enumerate(self.names):
        if
self.username_list_buttons[i].clic
ked(event):
            self.username =
self.username_list_buttons[i].get
_text()
      if
self.back_button.clicked(event):
          self.window = 'start'
```

This allows the left click of a mouse to get turned into an event so that it can be used throughout the gui's buttons.

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

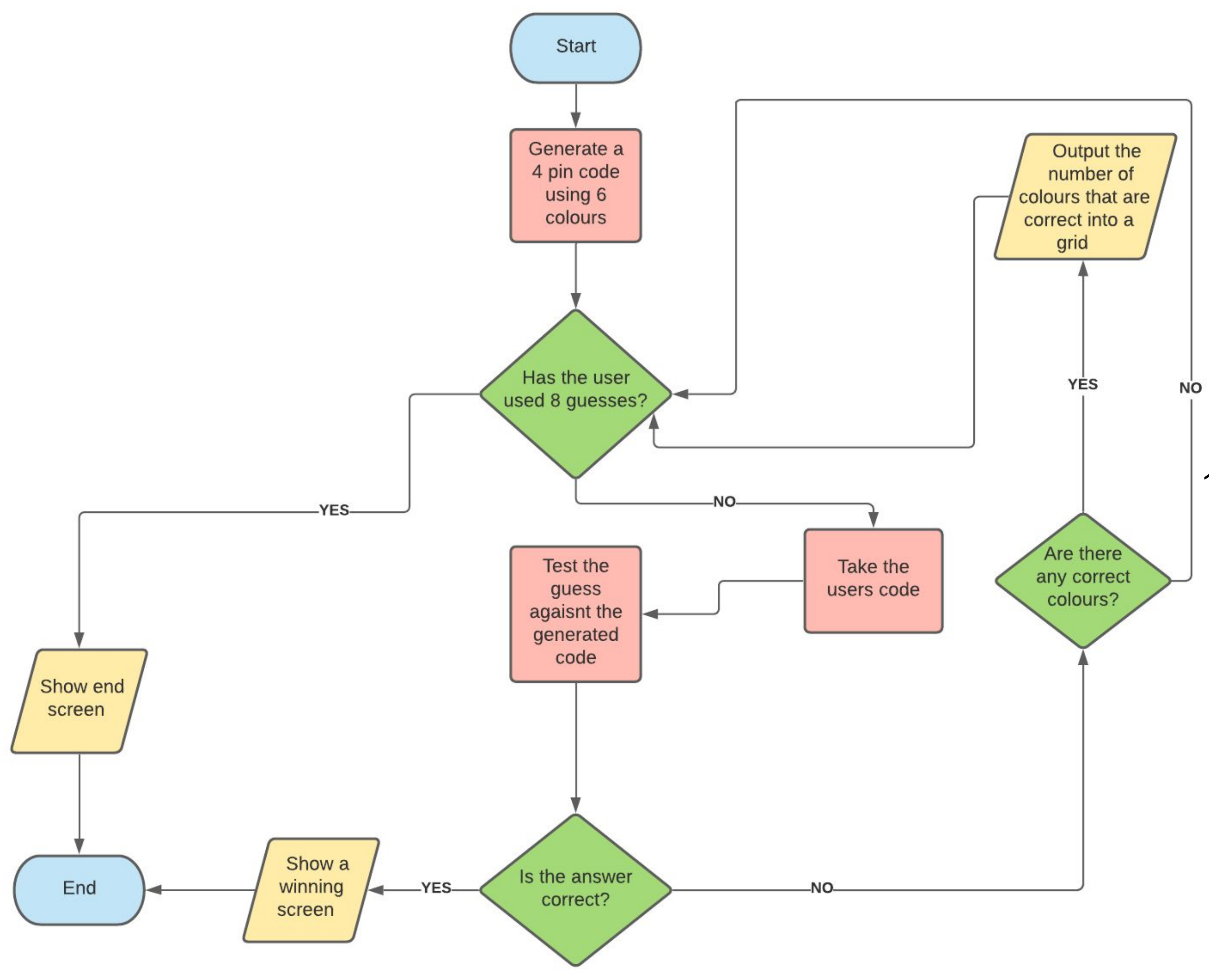
The above code is used to make the randomness of the code that the code breaker needs to crack. This is basically just a complex for loop with the 6 different possible colours being picked at random.

```
screen.fill(BACKGROUND)
   draw_background()
   draw_game()

   clock.tick(60)

pygame.quit()
```

The above code is a bit of visual formatting. The first 3 lines fill in the background and decoder board. The 5th line is used to set the decoder board to 60 FPS and then finally "pygame.quit" is used if the user quits the game.

```
def draw_pin(b, w, x, y):
   pinc = []
   for i in range(b):
      pinc.append(RED)
   for i in range(w):
      pinc.append(WHITE)
   for i in range(4 - b - w):
      pinc.append(GREY)
   pygame.draw.circle(screen,
pinc[0], [x - 10, y - 10], 5)
   pygame.draw.circle(screen,
pinc[1], [x + 10, y - 10], 5)
   pygame.draw.circle(screen,
pinc[2], [x - 10, y + 10], 5)
   pygame.draw.circle(screen,
pinc[3], [x + 10, y + 10], 5)
```

The above code is used to create the pins that are used to drag onto the decoder.


Start
Generate a 4 pin code using 6 colours
Has the user used 8 guesses?
YES
NO
Take the users code
Test the guess agaisnt the generated code
Is the answer correct?
YES
NO
Show a winning screen
Show end screen
End
Are there any correct colours?
YES
NO
Output the number of colours that are correct into a grid

The above image is a flowchart outlining what will need to be done to create a working mastermind game.This is a visual representation of the code that I will need to write in Pygame.

```
def incurrent_row(pos):
   x = pos[0]
   y = pos[1]
   ylim = Y_POS[row]
   return -15 < y - ylim < 15 and
min(X_POS) < x < max(X_POS)
```

The code here is used to check the position of where the user was clicking, here they are in the selection row where the code breaker moves their pin to. I repeated this so that I could know when the user was clicking on other parts of the decoder board and use this to make the buttons function when clicked.