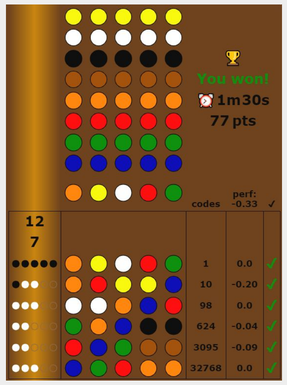
Pygame Mastermind Project

Louis Selwood A-Level Computer Science

Analysis

Rules of Mastermind

Mastermind is a family game that has two players, the code breaker and the code master, in this program the computer will act as the code master, the codemasters roll is to create a code that the code breaker then has to decipher, once the code breaker has made his guess the code master will place red pegs for each of the pegs that the code breaker has guessed correct colour and position and a white peg if they have guessed the correct colour but no the correct position, so in this case the program will play as the code master and create a code for the user(the code breaker) to attempt to break, the program must also determine how many red or white pegs need the player gets. After the code breaker has cracked the code or ran out of guesses they will be given the option to play again, in the case they choose to the code will be changed



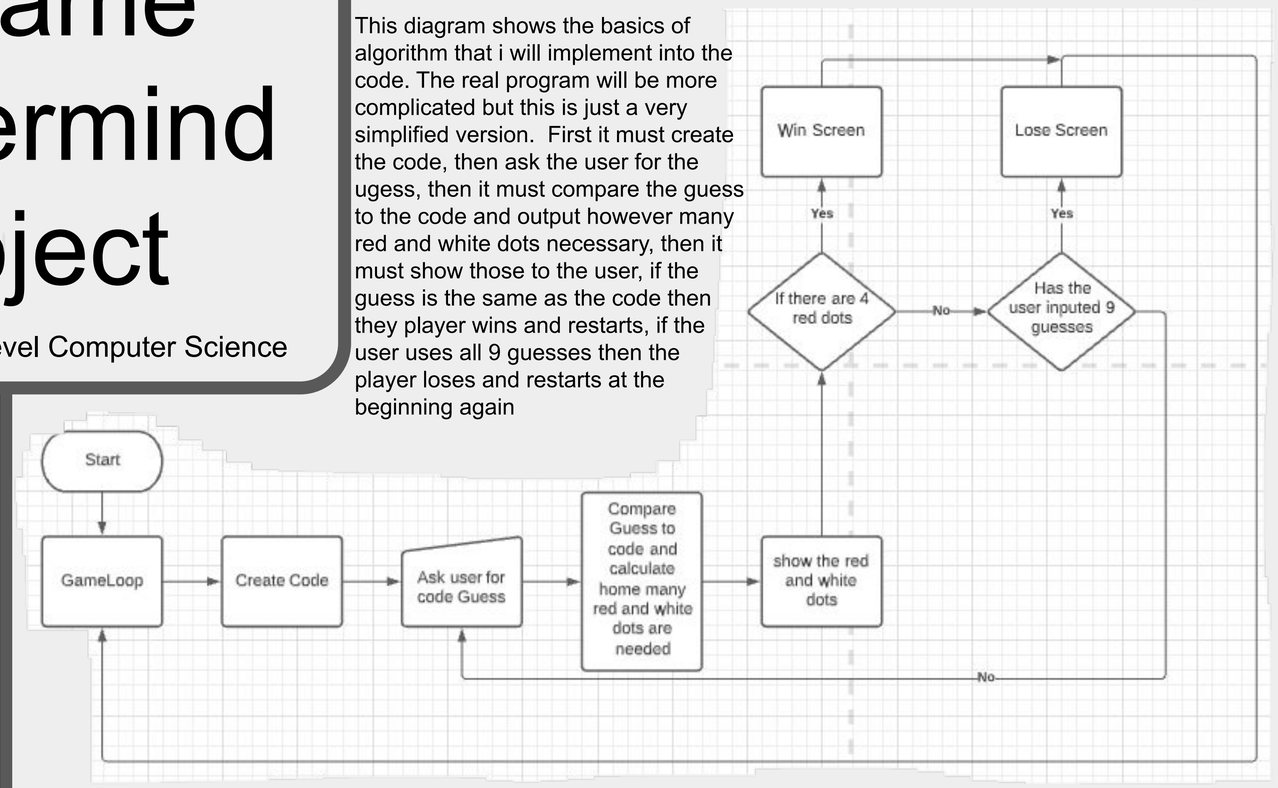
Pygame

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language. Pygame is highly portable and runs on nearly every platform and operating system. Pygame itself has been downloaded millions of times. Pygame is free. Released under the LGPL licence, you can create open source, freeware, shareware, and commercial games with it. See the licence for full details.

Design

The diagram

This diagram shows the basics of algorithm that i will implement into the code. The real program will be more complicated but this is just a very simplified version. First it must create the code, then ask the user for the ugess, then it must compare the guess to the code and output however many red and white dots necessary, then it must show those to the user, if the guess is the same as the code then they player wins and restarts, if the user uses all 9 guesses then the player loses and restarts at the beginning again



Technical Implementation

Subprograms

```
def checkPegClicked(peg, pegPos, colour, mouse):
    _pegDragged = None
    _pegCarrying = ""
    Dragging = False
    if pygame.Rect(pegPos[0], pegPos[1], 32, 32).collidepoint(mouse):
        _pegDragged = peg
        _pegCarrying = colour
    return _pegDragged, Dragging, _pegCarrying
```

Check Peg Clicked:
This subprogram is called when the mouse button is clicked, it checks if the pegs have been clicked on, if it has then it returns the colour of peg that has been clicked on, if not then nothing happens.

```
def ChooseCode():
    colours = [
        "Red", "Blue", "Orange", "Green", "Purple", "Yellow", "Black", "Pink"
    ]
    Code = []
    for i in range(4):
        randNum = random.randint(0, len(colours) - 1)
        Code.append(colours[randNum])
    return Code
```

Create Code
This subprogram creates a 4 item code from 9 different colours,

```
def showGuesses(guesses, colourPegs, holes):
    x = 0
    for guess in guesses:
        if (guess != None):
            for peg in colourPegs:
                i = 0
                for colour in guess:
                    if (peg.colour == colour):
                        screen.blit(peg.img, (holes[i][0], 740 - (70 * x)))
                    i += 1
        x += 1
```

Show Guesses:
This subprogram is called with every repeat of the main loop, It shuffles through the list Guesses and prints all of the pegs on the mastermind board.

The Game Loop:
There are two main loops in the code, running and while(True), running is within while(True) and stops when the player either wins or loses, at the end of running there is a keyboard.wait("r") which waits until the user has pressed r to restart the code.

```
while(True):
    running = True
while running:
    #Setup
keyboard.wait("r")
```

```
def showPegs(pegs):
    x = 0
    for peg in pegs:
        i = 0
        if(peg != None):
            for pegI in peg:
                if(i == 0):
                    if(pegI == "Red"):
                        screen.blit(CorrectPeg, (10, 732 - (70 * x)))
                    if(pegI == "White"):
                        screen.blit(PartCorrectPeg, (10, 732 - (70 * x)))
                if(i == 1):
                    if(pegI == "Red"):
                        screen.blit(CorrectPeg, (35, 732 - (70 * x)))
                    if(pegI == "White"):
                        screen.blit(PartCorrectPeg, (35, 732 - (70 * x)))
                if(i == 2):
                    if(pegI == "Red"):
                        screen.blit(CorrectPeg, (10, 757 - (70 * x)))
                    if(pegI == "White"):
                        screen.blit(PartCorrectPeg, (10, 757 - (70 * x)))
                if(i == 3):
                    if(pegI == "Red"):
                        screen.blit(CorrectPeg, (35, 757 - (70 * x)))
                    if(pegI == "White"):
                        screen.blit(PartCorrectPeg, (35, 757 - (70 * x)))
                i += 1
        x += 1
```

Show Pegs
This subprogram is called every repeat and takes in the input of RedAndWhitePegList, It reads through the list and prints the Red or White peg depending on the value of the item in the list.

Event Monitoring:
For pygame i had to learn an entirely new set of commands, one of them was Event Monitoring. In pygame there is a list of events that happened since last checked in "pygame.event.get()" and this can be used to check is certain actions have taken place.

```
#EventMonitoring
mouse = pygame.mouse.get_pos()
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        running = False
    elif event.type == pygame.MOUSEBUTTONDOWN:
        dragging = True
        for s in ColourPegs:
            pegImg = s.img
            pegPos = s.pos
            pegColour = s.colour
            PegDragging, Dragging, colourDragging = checkPegClicked(
                pegImg, pegPos, pegColour, event.pos)
            if (PegDragging != None):
                break
    elif event.type == pygame.MOUSEBUTTONUP:
        dragging = False
        if (MouseOverHole != None):
            Guess[MouseOverHole] = colourDragging
            count = 0
```

Code

```
screen.fill((173, 216, 230))
screen.blit(boardImg, (0, 0))
screen.blit(RedPeg, (380, 280))
screen.blit(BluePeg, (380, 320))
screen.blit(OrangePeg, (380, 360))
screen.blit(GreenPeg, (380, 400))
screen.blit(PurplePeg, (380, 440))
screen.blit(YellowPeg, (380, 480))
screen.blit(BlackPeg, (380, 520))
screen.blit(PinkPeg, (380, 560))
screen.blit(Arrow, (310, 740 - (70 * CurrentLevel)))
```

Sprite Placing:
While doing this project i also had to learn how to make shapes appear on screen, i also had to learn how to add pngs and make them appear, this can be done with screen.blit which means you can choose which sprite goes where

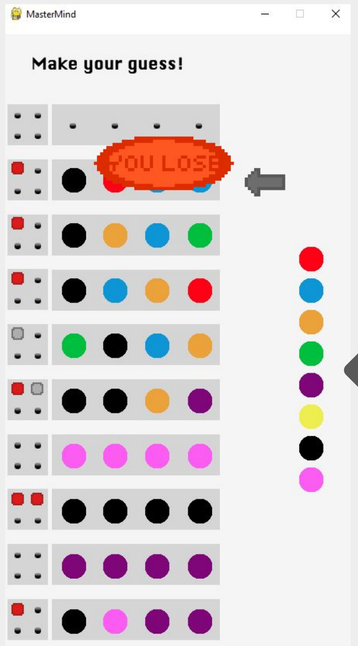
```
if(CurrentLevel == 9):
    won = False
    screen.blit(YouLoseScreen, (100, 100))
    running = False
if(Guess == Code):
    won = True
    screen.blit(YouWinScreen, (100, 100))
    running = False
```

Win and Lose Conditions:
In mastermind you win when your guess is equal to the original code and you lose when you run out of guesses, so for the programming i just turned these conditions into if statements and set running to False when these were met, it also displays win and lose screens.

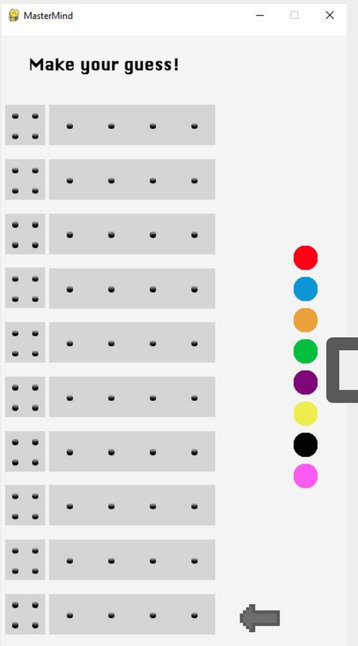
```
class Pegs:
    def __init__(self, _img, _pos, _colour):
        self.img = _img
        self.pos = _pos
        self.colour = _colour
```

Use Of Classes:
In this scenario i used a class to that i could store the position, img name and colour of each peg.

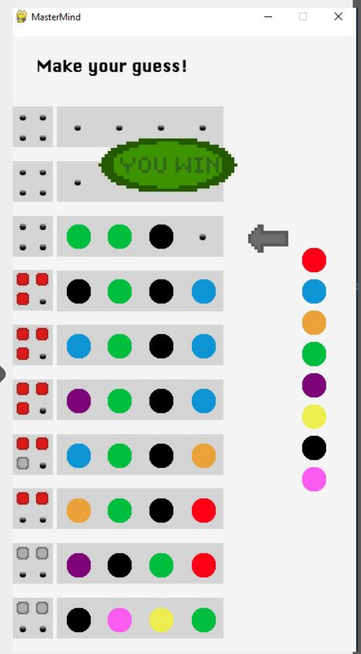
Testing and Evaluation



Lose State
This is what happens when all of the guesses are used up, as you can see in the example that is above the text i did not guess the code in time and therefore the lose screen came up, now if i wanted to reset i would press r and a new code would be created



Start State
This is what the Mastermind board looks like before anything has been altered the arrow that can be seen above signals to the player which strip they are playing on and the red and white dots on the side show how many the player has gotten correct



Win State
This is what happens when the player guesses the correct come in time, as you can see above the player guessed correctly and therefore the win screen appeared in order to restart the player must then press r and a new code will be generated