上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Guess IT:

A Pythonic Approach

INTRODUCTION TO BUSINESS COMPUTING

FINAL PROJECT REPORT

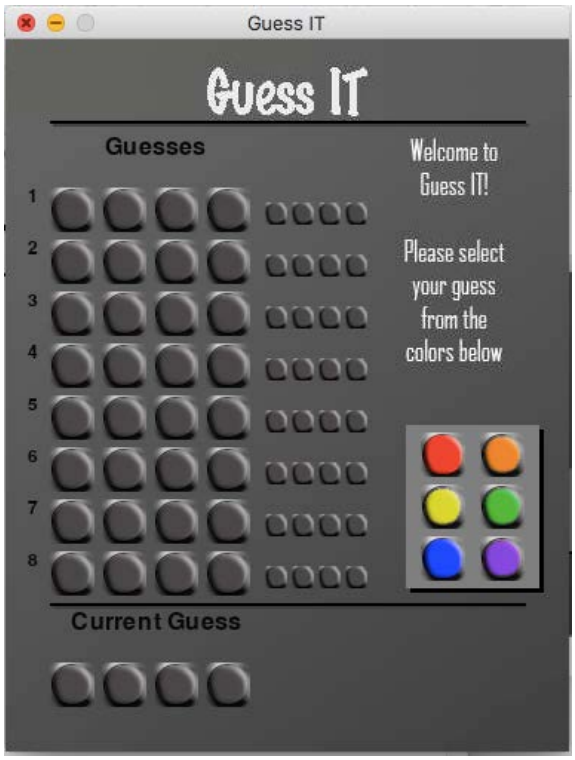
CHIA JUN JIE | 714120290007
CHONG KA SHING CLARIS | 714120290006
IGOR IVANKIN | 714120290004
SHANE BRADLEY | 714120290005

Abstract

This project encapsulates the formation of a logic puzzle, Guess IT, using Object-Oriented Programming, including Pygame and other important Python tools. The report is intended to cover the processes our group took when developing the final game, a description of the lessons learnt during the course of the project as well as the possible future development to improvise the game.

Introduction

About the Game



Guess IT is a deductive logic puzzle that involves the player to guess a randomly-generated combination of 4 colours as the code is initially ran. After each round, the player is given a hint based on the number of correct answers the player selected in the round. The hint is given through a selection of black and white colours, where black indicates a correct colour

in a correct position, and white indicates a correct colour in the wrong position. An empty blank indicates a wrong colour. However, to increase the difficulty of the game, the hint does not indicate the position of the correct colour, only the existence of the correct colour. In total, the player is given 8 rounds to guess the colour combination and the results of the game (Win or Lose) will then be displayed.

Objective of the Project

We choose to design Guess IT because it is a fantastic board game our group members played when we were young. Interestingly, this game is not well-known in China and many local students have not seen it before. Hence, we decide to design this game and introduce to the class.

About the Code

For the coding we chose to take an Object Oriented approach. We defined a class variable and broke down our game into numerous functions which would execute when they were called. We have a lot of elements in our code that we hadn't learned previously primarily because of the Pygame package we had to download and install. Since this was the case, when we were programming our interface, it took us a while to understand the design and location codes associated with the visual display. After consulting our resources and testing different codes, we are left with our final results and a successful game.

Coding Process (i.e Code Interpretation)

Class

In order to design the game, we have used pygame library and functions of it.

```
class Button(pygame.sprite.Sprite):
    def __init__(self,image):
        pygame.sprite.Sprite.__init__(self)
        self.image, self.rect = load_image(image)
    def setCords(self,x,y):
        self.rect.topleft = x,y
        screen.blit(self.image, (x,y))
    def pressed(self,mouse):
        if mouse[0] > self.rect.topleft[0]:
            if mouse[1] > self.rect.topleft[1]:
                if mouse[0] < self.rect.bottomright[0]:
                    if mouse[1] < self.rect.bottomright[1]:
                        return True
                    else: return False
                else: return False
            else: return False
        else: return False
```

In order to work with a group of images, we had to use “class”. It is used to group data and functions. In our case, we have to work with a group of images, from background to colours. Also, in order to identify, when the certain colour is pressed, we used the above lines of code (def pressed...)

```
class Board:
    def __init__(self):
        self.board = ["e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",
                      "e  e  e  e",]
        self.bwboard = ["e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",
                        "e  e  e  e",]
        self.guess = ["e", "e", "e", "e"]
        self.gx = 270
        self.gy = 260
        self.gby = 420
        self.bluebut = Button('/Users/ivankinigor/image/bluepeg.png')
        self.redbut = Button('/Users/ivankinigor/image/redpeg.png')
        self.yelbut = Button('/Users/ivankinigor/image/yellowpeg.png')
        self.orbut = Button('/Users/ivankinigor/image/orangepeg.png')
        self.purpbut = Button('/Users/ivankinigor/image/purppeg.png')
        self.greenbut = Button('/Users/ivankinigor/image/greenpeg.png')
        self.submitbut = Button('/Users/ivankinigor/image/submit.png')
        self.font = pygame.font.SysFont('agencyfb', 18)
        self.font2 = pygame.font.SysFont('agencyfb', 24)
```

In the game, we have a few boards. In the above you can see how the board is settled in terms of the colours. We have a few images corresponding to certain colour and we have used the images in order to fill in the boards.

```
def drawboard(self):
    self.guessline = 1
    self.bx = 30
    self.by = 100
    for row in self.board:
        self.g = str(self.guessline)
        self.text = self.font.render(self.g, 1, (10, 10, 10))
        screen.blit(self.text, (self.bx - 15, self.by))
        self.guessline += 1
        pygame.display.update()
        for col in row:
            if col == "e":
                screen.blit(empty_peg, (self.bx, self.by))
            elif col == "r":
                screen.blit(red_peg, (self.bx, self.by))
            elif col == "b":
                screen.blit(blue_peg, (self.bx, self.by))
            elif col == "g":
                screen.blit(green_peg, (self.bx, self.by))
            elif col == "p":
                screen.blit(purple_peg, (self.bx, self.by))
            elif col == "y":
                screen.blit(yellow_peg, (self.bx, self.by))
            elif col == "o":
                screen.blit(orange_peg, (self.bx, self.by))
            else:
                continue
            self.bx += 35
        self.by += 35
        self.bx = 30
        pygame.display.flip()
```

The above lines of codes are responsible for setting the location of the guesses board according to the coordinates given and the parameters for the text font. It also defines movement of the images (colours) from current guess board to guesses board.

The letters “e”, “r”, “b”, “g”, “p”, “y”, “o” correspond to certain colour. It matches certain images pressed to the correspondent colour.

```
def drawbw(self):
    self.bwx = 175
    self.bwy = 110
    for row in self.bwboard:
        for col in row:
            if col == "e":
                screen.blit(bw_empty, (self.bwx, self.bwy))
            elif col == "b":
                screen.blit(bw_black, (self.bwx, self.bwy))
            elif col == "w":
                screen.blit(bw_white, (self.bwx, self.bwy))
            else:
                continue
            self.bwx += 18
        self.bwy += 35
        self.bwx = 175
        pygame.display.flip()
```

The above lines of code set the location of the guesses board according to coordinates gives. Also, defines movement of the images (colours) from current guess board to guesses board.

```
def colorbin(self):
    pygame.draw.rect(screen, BLACK, (self.gx + 3, self.gy + 3, 90,110))
    pygame.draw.rect(screen, GREY, (self.gx,self.gy,90,110))
    self.redbut.setCords(self.gx+10,self.gy+5)
    self.orbut.setCords(self.gx+50,self.gy+5)
    self.yelbut.setCords(self.gx+10, self.gy +40)
    self.greenbut.setCords(self.gx+50, self.gy+40)
    self.bluebut.setCords(self.gx+10, self.gy+75)
    self.purpbut.setCords(self.gx+50, self.gy+75)
    pygame.display.update()
```

The above code is used in order to set the coordinates for colour board.

```
def guessdisplay(self):
    self.bx = 30
    for row in self.guess:
        if row == "e":
            screen.blit(empty_peg, (self.bx, self.gby))
        elif row == "r":
            screen.blit(red_peg, (self.bx, self.gby))
        elif row == "b":
            screen.blit(blue_peg, (self.bx, self.gby))
        elif row == "g":
            screen.blit(green_peg, (self.bx, self.gby))
        elif row == "p":
            screen.blit(purple_peg, (self.bx, self.gby))
        elif row == "y":
            screen.blit(yellow_peg, (self.bx, self.gby))
        elif row == "o":
            screen.blit(orange_peg, (self.bx, self.gby))
        else:
            continue
        self.bx += 35
        pygame.display.flip()
```

The above code defines the movement of the colours from the colour board to the current guess. Each letter “e”, “r”, … “o” corresponds to certain colour.

```
def pegcheck(self, guess):
    self.strikes1 = []
    self.strikes2 = []
    self.blackpeg=0
    self.whitepeg=0
    self.bwcount = []
    for i in range(len(guess)):
        if guess[i] == solution[i]:
            self.blackpeg += 1
            self.strikes1.append(i)
            self.strikes2.append(i)
            self.bwcount.append("b")
    for x in range(len(solution)):
        for y in range(len(solution)):
            if x not in self.strikes1 and y not in self.strikes2:
                if guess[x] == solution[y]:
                    self.whitepeg += 1
                    self.strikes1.append(x)
                    self.strikes2.append(i)
                    self.bwcount.append("w")
        self.bwboard[turn] = self.bwcount
```

The above code is important because it is used to see the result of the each guess. It compares the generated answer with the colours in the guesses board. According to the result, it shows black, white or grey colour in the hint board. Black colour means that certain colour is placed

at the right position, white colour means that certain colour is chosen right but not on the right position. Grey colour means that chosen colour is incorrect.

```
def win(self):
    screen.blit(winbg, (0,0))
    pygame.display.flip()
    pygame.time.delay(5000)
    exit()
```

The code shown above simply shows the function that will be executed when the correct combination of colors is chosen. The image will be called from a another function and the time portion is how long the game will allow the screen to be open before the game closes.

```
def lose(self):
    screen.blit(losebg, (0,0))
    self.bx = 115
    for row in solution:
        if row == "e":
            screen.blit(empty_peg, (self.bx, 400))
        elif row == "r":
            screen.blit(red_peg, (self.bx, 400))
        elif row == "b":
            screen.blit(blue_peg, (self.bx, 400))
        elif row == "g":
            screen.blit(green_peg, (self.bx, 400))
        elif row == "p":
            screen.blit(purple_peg, (self.bx, 400))
        elif row == "y":
            screen.blit(yellow_peg, (self.bx, 400))
        elif row == "o":
            screen.blit(orange_peg, (self.bx, 400))
        else:
            continue
        self.bx += 35
    pygame.display.update()
    pygame.time.delay(5000)
    exit()
```

This portion of code shows the function that will be executed when you are done with guessing the 4 colors 8 different times. The image will be displayed and the correct color combination will be shown on the screen. The time limit is the same as the win one above.

```
def getanswer():
    #generates the solution which the player must guess
    availcolors = ("r", "o", "y", "g", "b", "p")
    answer = [random.choice(availcolors) for i in range(4)]
    return answer
```

```
def drawbg():
    background, bg_rect = load_image('/Users/ivankinigor/image/mmbg2.jpg')
    screen.blit(background, (0,0))
    pygame.draw.line(screen, BLACK, (30, 55), (350, 55), 2)
    pygame.draw.line(screen, DKGREY, (32,57), (352, 57), 2)
    pygame.draw.line(screen, BLACK, (30, 380), (350, 380), 2)
    pygame.draw.line(screen, DKGREY, (32,382), (352, 382), 2)
    heading_text = font.render("Guesses", 1, (10, 10, 10))
    heading_textpos = (67, 65)
    current_guess_text = font.render('Current Guess', 1, (10,10,10))
    current_guesspos = (44, 385)
    screen.blit(current_guess_text, current_guesspos)
    screen.blit(heading_text, heading_textpos)
    pygame.display.update()
```

The first function shown above is for when the game generates an answer for the colors. Every time the game is opened this code is automatically ran. The number 4 in the range stands for the number of colors that will be randomly generated in the answer. The next function describes the background generation of the game. Things such as the color background, line colors, line shadows and thickness, location of these lines, etc.

```
empty_peg, empty_rect = load_image('/Users/ivankinigor/image/mmempty.png')
red_peg, red_peg_rect = load_image('/Users/ivankinigor/image/redpeg.png')
blue_peg, blue_peg_rect = load_image('/Users/ivankinigor/image/bluepeg.png')
green_peg, green_peg_rect = load_image('/Users/ivankinigor/image/greenpeg.png')
purple_peg, purple_peg_rect = load_image('/Users/ivankinigor/image/purppeg.png')
yellow_peg, yellow_peg_rect = load_image('/Users/ivankinigor/image/yellowpeg.png')
orange_peg, orange_peg_rect = load_image('/Users/ivankinigor/image/orangepeg.png')
bw_empty, bw_empty_rect = load_image('/Users/ivankinigor/image/bwempty1.png')
bw_white, bw_white_rect = load_image('/Users/ivankinigor/image/bwwhite.png')
bw_black, bw_black_rect = load_image('/Users/ivankinigor/image/bwblack.png')
winbg, winbg_rect = load_image('/Users/ivankinigor/image/mmbgwin.jpg')
losebg, losebg_rect = load_image('/Users/ivankinigor/image/mmbglose.jpg')
```

This code shows what images are loaded when these variables are called. So in the guessdisplay function for example, we have the peg variables in use so then the color images will be loaded in the displays from these lines of code.

```
while turn <= 8:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        elif turn == 8:
            board.lose()
        elif event.type == MOUSEBUTTONDOWN and guessnum < 4:
            pos = pygame.mouse.get_pos()
            if board.bluebut.pressed(pos) == True:
                board.guess[guessnum] = "b"
                board.guessdisplay()
                guessnum += 1
            elif board.yelbut.pressed(pos) == True:
                board.guess[guessnum] = "y"
                board.guessdisplay()
                guessnum += 1

            elif board.orbut.pressed(pos) == True:
                board.guess[guessnum] = "o"
                board.guessdisplay()
                guessnum += 1

            elif board.purpbut.pressed(pos) == True:
                board.guess[guessnum] = "p"
                board.guessdisplay()
                guessnum += 1

            elif board.greenbut.pressed(pos) == True:
                board.guess[guessnum] = "g"
                board.guessdisplay()
                guessnum += 1

            elif board.redbut.pressed(pos) == True:
                board.guess[guessnum] = "r"
                board.guessdisplay()
                guessnum += 1
            else:
                continue
        elif guessnum == 4:
            board.board[turn] = board.guess
            board.drawboard()
            board.pegcheck(board.guess)
            board.drawbw()
            board.guess = ["e", "e", "e", "e"]
            turn += 1
            board.guessdisplay()
            board.drawbw()
            guessnum = 0
            if board.blackpeg == 4:
                board.win()
                break
        else:
            continue
```

Next is the main loop of code that executes the actual game playing process.This code pretty much shows that you can have no more than 8 guesses at the color combination. Once your choices get to 8 and you don’t have all black pegs then the lose function is called. The nested if/elif pattern within the elif shows all the colors you can choose and doesn’t allow you to pick more than 4. Once you pick 4 colors the last code portion will run and if you get all black pegs then the win function is called and if not then you continue in the loop until you get to 8 and the lose function is called.

Post-Coding Process

Lessons Learnt

Variable Definitions

One important lesson we learnt was the importance of defining the variables and class with a different term. Early on in the coding process, the code continuously ran with errors, and it was not until a few repeated attempts at troubleshooting did we realise that the error was due to Python's inability to run the code as one of our variables had the same term as the class we defined earlier in the process. This was pertinent as after importing Pygame, there were more terms that we were restricted to use as they were previously defined as another term or variable within Pygame.

Code Efficiency

As the game requires a fair amount of coding lines, we realised that the more efficient our codes were, the easier it was for us to pinpoint the errors when it happened. It was first mentioned in class, where we had to find the most pythonic code for a problem. However, we recalled this lesson only after encountering an error that was buried among lines of inefficient code earlier in our coding process.

Challenges Faced

Pygame

Despite Pygame being the crucial component in our project, we were first introduced to in while compiling the initial research for the project proposal. As such, most of the problems

we faced were due to the lack of experience handling pygame and its related tools. However, utilising both the class lecture notes and the online resources enabled us to better understand the basics of Pygame.

Operating System (OS) Problems

Another challenge for the group was in the form of our differing operating systems, as we had a combination of Macintosh users and Windows users. Often the problems stemmed from our inability to port the codes from OS to OS, and as a result, much time was wasted in the process. It hindered the process of developing the final game as we were unable to collaborate as seamlessly as we would have preferred. These problems are more often than not also prevalent among other Python users, and as such, after troubleshooting online, we are able to resolve most of our issues.

Future Developments

After the development of our final version of Guess IT, we felt that in the future, we should improve the game by coding an automation that manages to solve the game within the 8 guesses every round. Another possible development is to include a music soundtrack for the game, involving different jingles depending on which stage of the game the player is at. The last development we intend to attempt in the future would be to include an undo button to improve player comfortability with the game interface and a solve button to reveal the answer should the player wants to end the game halfway.