Memory Puzzle

Making Game with Python

Last time

- Pygame Primitive Drawing Functions
- Drawing code

Today

- Python advance
  - Nested for loop
  - List of list
  - Random shuffle and list operations
- Memory puzzle code
  - Main function

Nested for loop

```
colors = ['red', 'green', 'blue']
shapes = ['donut', 'square', 'diamond', 'oval']

for c in colors:
    for s in shapes:
        print(c, s)

print()

for s in shapes:
    for c in colors:
        print(c, s)
```

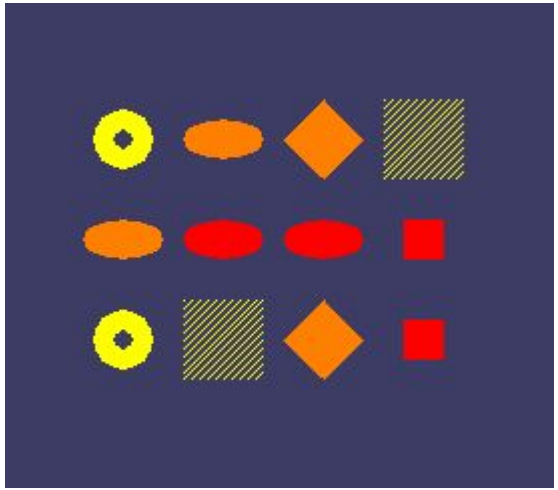
List of lists

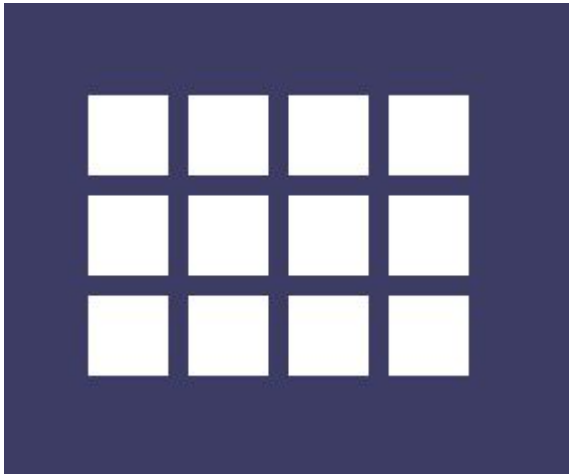
```
board = []
num_column = 3
num_row = 4
for i in range(num_column):
    column = []

    for j in range(num_row):
        column.append(j)

    board.append(column)

print(board)
```

Random Shuffle and list operations

```
import random
A = [1,2,3,4,5]

random.shuffle(A)
print(A)

B = A[:3]
print(B)

C = B * 2
print(C)

random.shuffle(C)
print(C)
```

Memory Puzzle Game

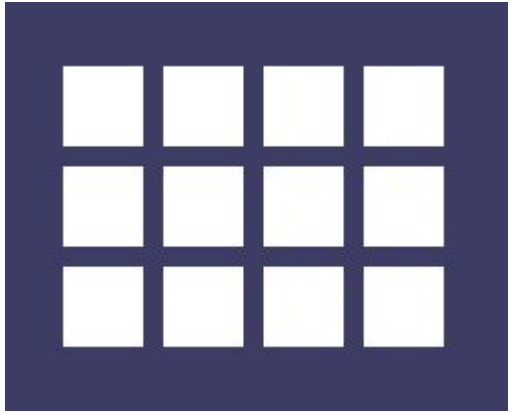




https://github.com/zhihongzeng2002/pythongame

Game Parameters

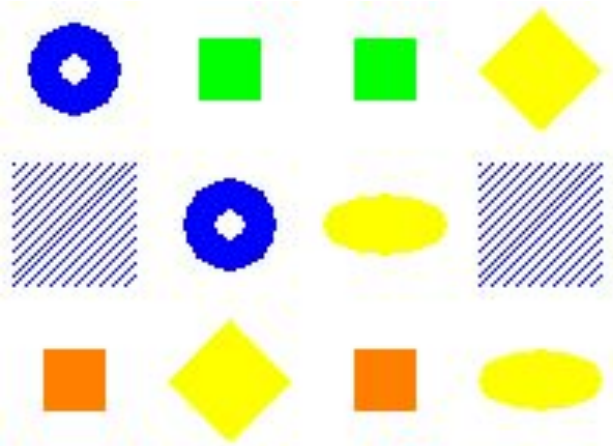
```
FPS = 30 # frames per second, the general speed of the program
WINDOWWIDTH = 640 # size of window's width in pixels
WINDOWHEIGHT = 480 # size of windows' height in pixels
REVEALSPEED = 8 # speed boxes' sliding reveals and covers
BOXSIZE = 40 # size of box height & width in pixels
GAPSIZE = 10 # size of gap between boxes in pixels
BOARDWIDTH = 4 # number of columns of icons
BOARDHEIGHT = 3 # number of rows of icons
assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'Board needs to have an even number
XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE + GAPSIZE))) / 2)
```



Game Parameters

```
GRAY     = (100, 100, 100)
NAVYBLUE = ( 60,  60, 100)
WHITE    = (255, 255, 255)
RED      = (255,   0,   0)
GREEN    = (  0, 255,   0)
BLUE     = (  0,   0, 255)
YELLOW   = (255, 255,   0)
ORANGE   = (255, 128,   0)
PURPLE   = (255,   0, 255)
CYAN     = (  0, 255, 255)
```

```
DONUT = 'donut'
SQUARE = 'square'
DIAMOND = 'diamond'
LINES = 'lines'
OVAL = 'oval'
```



Main function

```
def main():
    global FPSCLOCK, DISPLAYSURF
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))

    mousex = 0 # used to store x coordinate of mouse event
    mousey = 0 # used to store y coordinate of mouse event
    pygame.display.set_caption('Memory Game')

    mainBoard = getRandomizedBoard()
    revealedBoxes = generateRevealedBoxesData(False)

    firstSelection = None # stores the (x, y) of the first box clicked.

    DISPLAYSURF.fill(BGCOLOR)
    startGameAnimation(mainBoard)
```

```
while True: # main game loop
    mouseClicked = False

    DISPLAYSURF.fill(BGCOLOR) # drawing the window
    drawBoard(mainBoard, revealedBoxes)

    for event in pygame.event.get(): # event handling loop
        if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
            pygame.quit()
            sys.exit()
        elif event.type == MOUSEMOTION:
            mousex, mousey = event.pos
        elif event.type == MOUSEBUTTONUP:
            mousex, mousey = event.pos
            mouseClicked = True
```

```
boxx, boxy = getBoxAtPixel(mousex, mousey)
if boxx != None and boxy != None:
    # The mouse is currently over a box.
    if not revealedBoxes[boxx][boxy]:
        drawHighlightBox(boxx, boxy)
    if not revealedBoxes[boxx][boxy] and mouseClicked:
        revealBoxesAnimation(mainBoard, [(boxx, boxy)])
        revealedBoxes[boxx][boxy] = True # set the box as "revealed"
        if firstSelection == None: # the current box was the first box clicked
            firstSelection = (boxx, boxy)
        else: # the current box was the second box clicked
            # Check if there is a match between the two icons.
            icon1shape, icon1color = getShapeAndColor(mainBoard, firstSelection[0], firstSelection[1])
            icon2shape, icon2color = getShapeAndColor(mainBoard, boxx, boxy)
```

```
if icon1shape != icon2shape or icon1color != icon2color:
    # Icons don't match. Re-cover up both selections.
    pygame.time.wait(1000) # 1000 milliseconds = 1 sec
    coverBoxesAnimation(mainBoard, [(firstSelection[0], firstSelection[1]), (boxx, boxy)])
    revealedBoxes[firstSelection[0]][firstSelection[1]] = False
    revealedBoxes[boxx][boxy] = False
```

```
elif hasWon(revealedBoxes): # check if all pairs found
    gameWonAnimation(mainBoard)
    pygame.time.wait(2000)

    # Reset the board
    mainBoard = getRandomizedBoard()
    revealedBoxes = generateRevealedBoxesData(False)

    # Show the fully unrevealed board for a second.
    drawBoard(mainBoard, revealedBoxes)
    pygame.display.update()
    pygame.time.wait(1000)

    # Replay the start game animation.
    startGameAnimation(mainBoard)
```

```
                firstSelection = None # reset firstSelection variable

# Redraw the screen and wait a clock tick.
pygame.display.update()
FPSCLOCK.tick(FPS)
```