```
import random, pygame, sys
from pygame.locals import *




FPS = 30 # кадры в секунду
WINDOWWIDTH = 640 #  ширина окна в пикселях
WINDOWHEIGHT = 480 # высота окна
REVEALSPEED = 3 # скорость открывания пазлов
BOXSIZE = 40 # высота и ширина поля в пикселях
GAPSIZE = 10 # промежуток между пазлами
BOARDWIDTH = 3 # кол-во столбцов иконок
BOARDHEIGHT = 4 # кол-во рядов иконок
assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'На поле должно быть четное
количество пазлов.'
XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE + GAPSIZE))) / 2)

HARD_BOARDWIDTH = 10
HARD_BOARDHEIGHT = 7
def hard_game():
    global BOARDWIDTH, BOARDHEIGHT, WINDOWWIDTH, WINDOWHEIGHT
    BOARDWIDTH = HARD_BOARDWIDTH
    BOARDHEIGHT = HARD_BOARDHEIGHT
    WINDOWWIDTH = BOARDWIDTH * (BOXSIZE + GAPSIZE) + XMARGIN * 2
    WINDOWHEIGHT = BOARDHEIGHT * (BOXSIZE + GAPSIZE) + YMARGIN * 2
    main()



#параметры кнопок
BUTTON_WIDTH = 200
BUTTON_HEIGHT = 50
BUTTON_COLOR = (150, 150, 150)
TEXT_COLOR = (255, 255, 255)
BUTTON_TEXT_SIZE = 30
def draw_button(screen, msg, x, y, w, h, ic, ac):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(screen, ac, (x, y, w, h))
        if click[0] == 1:
            return True
    else:
        pygame.draw.rect(screen, ic, (x, y, w, h))

    small_text = pygame.font.Font(None, BUTTON_TEXT_SIZE)
    text_surf = small_text.render(msg, True, TEXT_COLOR)
    text_rect = text_surf.get_rect()
    text_rect.center = ((x + (w / 2)), (y + (h / 2)))
    screen.blit(text_surf, text_rect)
    return False


def display_confirmation(screen):
    confirmation_box = pygame.Rect(150, 150, 400, 200)
    pygame.draw.rect(screen, (200, 200, 200), confirmation_box)

    font = pygame.font.Font(None, 36)
    text = font.render("Выйти из игры?", True, (0, 0, 0))
    text_rect = text.get_rect(center=confirmation_box.center)
    screen.blit(text, text_rect)
```

```
    # Кнопка Да
    yes_button = pygame.Rect(confirmation_box.left + 30,
confirmation_box.bottom - 70, 100, 40)
    pygame.draw.rect(screen, (100, 100, 100), yes_button)
    yes_text = font.render("Да", True, (255, 255, 255))
    yes_text_rect = yes_text.get_rect(center=yes_button.center)
    screen.blit(yes_text, yes_text_rect)

    # Кнопка Нет
    no_button = pygame.Rect(confirmation_box.right - 130,
confirmation_box.bottom - 70, 100, 40)
    pygame.draw.rect(screen, (100, 100, 100), no_button)
    no_text = font.render("Нет", True, (255, 255, 255))
    no_text_rect = no_text.get_rect(center=no_button.center)
    screen.blit(no_text, no_text_rect)

    return yes_button, no_button

def main_menu():
    pygame.init()
    screen = pygame.display.set_mode((640, 480))
    pygame.display.set_caption('Memory Game')

    clock = pygame.time.Clock()
    running = True

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()

        screen.fill((0, 255, 255))  # заполняем экран цветом

        hard_button_clicked = draw_button(screen, "Hard", 220, 400,
BUTTON_WIDTH, BUTTON_HEIGHT, BUTTON_COLOR,
                                          (100, 100, 100))
        if hard_button_clicked:
            hard_game()  # запуск игры с уровнем сложности



        # рисуем кнопку "играть"
        play_button_clicked = draw_button(screen, "Начать", 220, 200,
BUTTON_WIDTH, BUTTON_HEIGHT, BUTTON_COLOR, (100, 100, 100))
        if play_button_clicked:
            running = False  # Break the loop to start the game


        # рисуем кнопку "выйти"
        exit_button_clicked = draw_button(screen, "Выйти", 220, 300,
BUTTON_WIDTH, BUTTON_HEIGHT, BUTTON_COLOR, (100, 100, 100))
        if exit_button_clicked:
            confirmation_displayed = True
            while confirmation_displayed:
                screen.fill((0, 0, 0))  # Fill the screen with black
                yes_button, no_button = display_confirmation(screen)
                pygame.display.update()
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        pygame.quit()
                        quit()
                    elif event.type == pygame.MOUSEBUTTONDOWN:
```

```
                        mouse_pos = pygame.mouse.get_pos()
                        if yes_button.collidepoint(mouse_pos):
                            pygame.quit()
                            sys.exit()
                        elif no_button.collidepoint(mouse_pos):
                            confirmation_displayed = False

        pygame.display.update()
        clock.tick(30)

    # если нажали кнопку Играть
    if play_button_clicked:
        main()  # старт игры

#             R    G    B
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)

BGCOLOR = NAVYBLUE
LIGHTBGCOLOR = GRAY
BOXCOLOR = WHITE
HIGHLIGHTCOLOR = BLUE

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

ALLCOLORS = (RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, CYAN)
ALLSHAPES = (DONUT, SQUARE, DIAMOND, LINES, OVAL)
assert len(ALLCOLORS) * len(ALLSHAPES) * 2 >= BOARDWIDTH * BOARDHEIGHT, "Поле
слишком велико для заданного количества фигур."



def main():
    global FPSCLOCK, DISPLAYSURF
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    total_moves=0
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))

    mousex = 0 # хренение координаты x событий мыши
    mousey = 0 # хренение координаты y событий мыши
    pygame.display.set_caption('Memory Game')

    mainBoard = getRandomizedBoard()
    revealedBoxes = generateRevealedBoxesData(False)

    firstSelection = None # сохраняет (x, y) первого пазла, на который
щелкнули

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

```
    while True: # основной игровой цикл
        mouseClicked = False

        DISPLAYSURF.fill(BGCOLOR) # рисование окна
        drawBoard(mainBoard, revealedBoxes)
        font = pygame.font.Font(None, 36)
        text_surface = font.render(f"Moves: {total_moves}", True, WHITE)
        text_rect = text_surface.get_rect()
        text_rect.topleft = (10, 10)  # расположение счета в верхнем левом
углу
        DISPLAYSURF.blit(text_surface, text_rect)




        for event in pygame.event.get(): # цикл обработки событий
            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:
            # В данный момент мышь находится над пазлом
            if not revealedBoxes[boxx][boxy]:
                drawHighlightBox(boxx, boxy)
            if not revealedBoxes[boxx][boxy] and mouseClicked:
                total_moves += 1
                revealBoxesAnimation(mainBoard, [(boxx, boxy)])
                revealedBoxes[boxx][boxy] = True # устанавливает пазл как
открытый
                if firstSelection == None: # текущий пазл был первым, по
которому щелкнули
                    firstSelection = (boxx, boxy)
                else: # текущий пазл был вторым по клику
                    # проверяем, есть ли совпадение между двумя иконками
                    icon1shape, icon1color = getShapeAndColor(mainBoard,
firstSelection[0], firstSelection[1])
                    icon2shape, icon2color = getShapeAndColor(mainBoard,
boxx, boxy)

                    if icon1shape != icon2shape or icon1color != icon2color:
                        # Иконки не совпадают. Снова закрываем оба открытых
                        pygame.time.wait(1000) # 1000 миллисекунд = 1 секунда
                        coverBoxesAnimation(mainBoard, [(firstSelection[0],
firstSelection[1]), (boxx, boxy)])
                        revealedBoxes[firstSelection[0]][firstSelection[1]] =
False
                        revealedBoxes[boxx][boxy] = False


                    elif hasWon(revealedBoxes): # проверяем, все ли пары
найдены
                        gameWonAnimation(mainBoard)
                        pygame.time.wait(2000)
                        total_moves = 0
```

```
                        # сброс поля
                        mainBoard = getRandomizedBoard()
                        revealedBoxes = generateRevealedBoxesData(False)

                        # показываем на секунду все иконки пазлов
                        drawBoard(mainBoard, revealedBoxes)
                        pygame.display.update()
                        pygame.time.wait(1000)

                        # Воспроизведение анимации начала игры
                        startGameAnimation(mainBoard)
                    firstSelection = None #  сброс переменной firstSelection

        # Перерисовываем экран
        pygame.display.update()
        FPSCLOCK.tick(FPS)


def generateRevealedBoxesData(val):
    revealedBoxes = []
    for i in range(BOARDWIDTH):
        revealedBoxes.append([val] * BOARDHEIGHT)
    return revealedBoxes


def getRandomizedBoard():
    # Получаем список всех возможных фигур всех возможных цветов.
    icons = []
    for color in ALLCOLORS:
        for shape in ALLSHAPES:
            icons.append( (shape, color) )

    random.shuffle(icons) # меняет случайный порядок списка значков
    numIconsUsed = int(BOARDWIDTH * BOARDHEIGHT / 2) # подсчитаем, сколько
значков необходимо
    icons = icons[:numIconsUsed] * 2 # делаем по две штуки
    random.shuffle(icons)

    # Создаем структуру данных поля со случайно расположенными иконками
    board = []
    for x in range(BOARDWIDTH):
        column = []
        for y in range(BOARDHEIGHT):
            column.append(icons[0])
            del icons[0] # удаляем значки по мере их присвоения
        board.append(column)
    return board


def splitIntoGroupsOf(groupSize, theList):
    # разбиваем список на список списков, где внутренние списки имеют
    # Максимальное количество элементов groupSize.
    result = []
    for i in range(0, len(theList), groupSize):
        result.append(theList[i:i + groupSize])
    return result


def leftTopCoordsOfBox(boxx, boxy):
    # Преобразование координат поля в координаты пикселей
    left = boxx * (BOXSIZE + GAPSIZE) + XMARGIN
    top = boxy * (BOXSIZE + GAPSIZE) + YMARGIN
    return (left, top)
```

```
def getBoxAtPixel(x, y):
    for boxx in range(BOARDWIDTH):
        for boxy in range(BOARDHEIGHT):
            left, top = leftTopCoordsOfBox(boxx, boxy)
            boxRect = pygame.Rect(left, top, BOXSIZE, BOXSIZE)
            if boxRect.collidepoint(x, y):
                return (boxx, boxy)
    return (None, None)


def drawIcon(shape, color, boxx, boxy):
    quarter = int(BOXSIZE * 0.25)
    half =    int(BOXSIZE * 0.5)

    left, top = leftTopCoordsOfBox(boxx, boxy) # получаем координаты пикселей
из координат поля
    # Рисуем фигуры
    if shape == DONUT:
        pygame.draw.circle(DISPLAYSURF, color, (left + half, top + half),
half - 5)
        pygame.draw.circle(DISPLAYSURF, BGCOLOR, (left + half, top + half),
quarter - 5)
    elif shape == SQUARE:
        pygame.draw.rect(DISPLAYSURF, color, (left + quarter, top + quarter,
BOXSIZE - half, BOXSIZE - half))
    elif shape == DIAMOND:
        pygame.draw.polygon(DISPLAYSURF, color, ((left + half, top), (left +
BOXSIZE - 1, top + half), (left + half, top + BOXSIZE - 1), (left, top +
half)))
    elif shape == LINES:
        for i in range(0, BOXSIZE, 4):
            pygame.draw.line(DISPLAYSURF, color, (left, top + i), (left + i,
top))
            pygame.draw.line(DISPLAYSURF, color, (left + i, top + BOXSIZE -
1), (left + BOXSIZE - 1, top + i))
    elif shape == OVAL:
        pygame.draw.ellipse(DISPLAYSURF, color, (left, top + quarter,
BOXSIZE, half))


def getShapeAndColor(board, boxx, boxy):
    #  Значение формы для точки x, y хранится в board[x][y][0]
    #  значение цвета для точки x, y хранится в board[x][y][1]
    return board[boxx][boxy][0], board[boxx][boxy][1]


def drawBoxCovers(board, boxes, coverage):
    # Рисует закрытые/открытые пазлы. «boxes» — это список
    # из двух пунктов, в которых есть точки x и y в ячейке.
    for box in boxes:
        left, top = leftTopCoordsOfBox(box[0], box[1])
        pygame.draw.rect(DISPLAYSURF, BGCOLOR, (left, top, BOXSIZE, BOXSIZE))
        shape, color = getShapeAndColor(board, box[0], box[1])
        drawIcon(shape, color, box[0], box[1])
        if coverage > 0: # рисовать обложку, только если есть крышка
            pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top, coverage,
BOXSIZE))
    pygame.display.update()
    FPSCLOCK.tick(FPS)


def revealBoxesAnimation(board, boxesToReveal):
    # Выполняет анимацию открытия пазла.
```

```
    for coverage in range(BOXSIZE, (-REVEALSPEED) - 1, -REVEALSPEED):
        drawBoxCovers(board, boxesToReveal, coverage)


def coverBoxesAnimation(board, boxesToCover):
    # анимация закрытия "крышки" пазла
    for coverage in range(0, BOXSIZE + REVEALSPEED, REVEALSPEED):
        drawBoxCovers(board, boxesToCover, coverage)


def drawBoard(board, revealed):
    # Рисует все пазлы в закрытом или раскрытом состоянии.
    for boxx in range(BOARDWIDTH):
        for boxy in range(BOARDHEIGHT):
            left, top = leftTopCoordsOfBox(boxx, boxy)
            if not revealed[boxx][boxy]:
                # рисует закрытый пазл
                pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top, BOXSIZE,
BOXSIZE))
            else:
                # рисует иконку открытого пазла
                shape, color = getShapeAndColor(board, boxx, boxy)
                drawIcon(shape, color, boxx, boxy)


def drawHighlightBox(boxx, boxy):
    left, top = leftTopCoordsOfBox(boxx, boxy)
    pygame.draw.rect(DISPLAYSURF, HIGHLIGHTCOLOR, (left - 5, top - 5, BOXSIZE
+ 10, BOXSIZE + 10), 4)


def startGameAnimation(board):
    # случайно открывает пазлы по 8 за раз.
    coveredBoxes = generateRevealedBoxesData(False)
    boxes = []
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            boxes.append( (x, y) )
    random.shuffle(boxes)
    boxGroups = splitIntoGroupsOf(8, boxes)

    drawBoard(board, coveredBoxes)
    for boxGroup in boxGroups:
        revealBoxesAnimation(board, boxGroup)
        coverBoxesAnimation(board, boxGroup)


def gameWonAnimation(board):
    # мигает цвет фона, когда игрок выиграл
    coveredBoxes = generateRevealedBoxesData(True)
    color1 = LIGHTBGCOLOR
    color2 = BGCOLOR


    for i in range(13):
        color1, color2 = color2, color1 # меняет местами цвета
        DISPLAYSURF.fill(color1)
        drawBoard(board, coveredBoxes)
        pygame.display.update()
        pygame.time.wait(300)


def hasWon(revealedBoxes):
    # Возвращает True, если все пары пазлов раскрыты, в противном случае —
```

```
False.
    for i in revealedBoxes:
        if False in i:
            return False # возвращает False, если какие-либо пазлы закрыты.
    return True


if __name__ == '__main__':
    main_menu()
```