ПНВИ

Прв колоквиум

Memory Puzzle

измена во анимацијата што се појавува кога играчот успешно ќе ги отвори сите полиња така што ќе се појават квадрати околу секое поле и ќе се менува нивната боја во две бои по ваш избор.

```
def gameWonAnimation(board):
    . . . code . . .
    Animation(board)
    . . . code . . .

def Animation(board):
    colors = [RED, GREEN]  # Alternating colors for the animation
    animation_duration = 3000  # Animation duration in milliseconds
    start_time = pygame.time.get_ticks()

    while pygame.time.get_ticks() - start_time < animation_duration:
        current_color = colors[(pygame.time.get_ticks() // 500) % 2]  # Alternate colors every 500ms

        for boxx in range(BOARDWIDTH):
            for boxy in range(BOARDHEIGHT):
                left, top = leftTopCoordsOfBox(boxx, boxy)
                # Draw a rectangle slightly larger than the box for the animation
                pygame.draw.rect(DISPLAYSURF, current_color, (left - 5, top - 5, BOXSIZE + 10, BOXSIZE + 10), 3)

        pygame.display.update()
        FPSCLOCK.tick(FPS)
```

Сменете ја насоката на покривање/откривање на секое од полињата.

```
def drawBoxCovers(board, boxes, coverage):
    for box in boxes:
        left, top = leftTopCoordsOfBox(box[0], box[1])

        # Clear previous box content and draw the icon
        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:  # Only draw the cover if there is coverage
            # Create cover options for left-to-right or top-to-bottom animations
            cover_left = (left, top, coverage, BOXSIZE)  # Cover growing horizontally
            cover_top = (left, top, BOXSIZE, coverage)  # Cover growing vertically

            # Randomly choose one of the cover animations to draw
            cover = random.choice([cover_left, cover_top])
            pygame.draw.rect(DISPLAYSURF, BOXCOLOR, cover)

    pygame.display.update()
    FPSCLOCK.tick(FPS)
```

Slide Puzzle

Зголемете ги димензиите на коцките со бројки за 10 точки во секоја насока, и сменете ги сите соодветни вредности за да се задржи правилен приказ на таблата. Прикачете документ со слика и кратко објаснување за ова и секое од прашањата што следуваат, нумерирани според редоследот на појавување.

```
TILESIZE = 80+10
```

Направете случајно да се избира бројот на случајни чекори кои се прават на почеток за да се измеша таблата. Бројот треба да биде помеѓу 1 и 100, и треба да биде различен при секое ново почнување на играта.

```
def main():
    #Generate random number of steps between 1 and 100 to shuffle the board
    numSlides = random.randint(1, 100)
    mainBoard, solutionSeq = generateNewPuzzle(numSlides)
```

Додадете бројач на поместувањата што ги прави играчот. Ако корисникот успешно ја заврши играта, информирајте го за бројот на чекори што му бил потребен да ја заврши играта и бројот на чекори што би биле потребни за решавање на играта од страна на компјутерот на почетокот (имајќи ја предвид логиката што за таа цел се користи при кликнување на копчето "Solve").

```
moveCounter = 0  # додаден е бројач за потези

def main():
    while True:  # main game loop
. . .
      if mainBoard == SOLVEDBOARD:
            # додаден е бројот на потези во пораката
            solvedSteps = len(solutionSeq)
            msg = f"Solved in {moveCounter} moves! Computer solved it in {solvedSteps} moves."
```

Напишете ги командите кои што треба да бидат променети за да се забрза анимацијата на лизгање на коцките.

```
 slideAnimation(mainBoard, slideTo, 'Click tile or press arrow keys to slide.', 2) # show slide
on screen
 slideAnimation(board, move, 'Generating new puzzle...', animationSpeed=int(TILESIZE / 1))
```

Додадете можност за UNDO потег (ново копче), со тоа што ќе ги направите сите потребни промени за понатамошно правилно функционирање на играта.

```
def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT, RESET_SURF, RESET_RECT, NEW_SURF, NEW_RECT, SOLVE_SURF, SOLVE_RECT, UNDO_SURF,
UNDO_RECT, HELP_SURF, HELP_RECT

. . .
UNDO_SURF, UNDO_RECT = makeText('Undo', TEXTCOLOR, TILECOLOR, WINDOWWIDTH - 120, WINDOWHEIGHT -120)
    HELP_SURF, HELP_RECT = makeText('Help', TEXTCOLOR, TILECOLOR, WINDOWWIDTH - 240, WINDOWHEIGHT - 120)

 while True: # main game loop
. . .
 for event in pygame.event.get(): # event handling loop
. . .
if UNDO_RECT.collidepoint(event.pos):
                        if allMoves:
                            lastMove = allMoves.pop()
                            if lastMove == UP:
                                oppositeMove = DOWN
                            elif lastMove == DOWN:
                                oppositeMove = UP
                            elif lastMove == RIGHT:
                                oppositeMove = LEFT
                            elif lastMove == LEFT:
                                oppositeMove = RIGHT
                            slideAnimation(mainBoard, oppositeMove, '', animationSpeed=int(TILESIZE / 2))
                            makeMove(mainBoard, oppositeMove)
                            playerMovesCount -= 1
                    if HELP_RECT.collidepoint(event.pos):
                        print('HELP')
                        print('You can move the tiles by clicking on them or by using the arrow keys.')
                        print('You can reset the puzzle by clicking on the Reset button.')
                        print('You can generate a new puzzle by clicking on the New Game button.')
                        print('You can solve the puzzle by clicking on the Solve button.')
                        print('You can undo the last move by clicking on the Undo button.')
                        print('You can close the game by clicking on the close button.')
                        highlightNeighboringCells(mainBoard)

. . .

def drawBoard(board, message):
. . .
    DISPLAYSURF.blit(UNDO_SURF, UNDO_RECT)
    DISPLAYSURF.blit(HELP_SURF, HELP_RECT)
```

Simulate

Change the pattern list so that:
- the number of the elements in the next iteration will be calculated as 2^i, where i starts from 0 and increases by 1 in each iteration until the player loses the game.
- in each iteration, the pattern list is composed of new 2^i elements.

```
def main():
    # Initialize some variables for a new game
    pattern = [] # stores the pattern of colors
    currentStep = 0 # the color the player must push next
    lastClickTime = 0 # timestamp of the player's last button push
    score = 0
    iteration = 0 # the current iteration the player is on

 while True: # main game loop
     if not waitingForInput:
            # play the pattern
            pygame.display.update()
            pygame.time.wait(1000)
            num_elements = 2 ** iteration # number of elements for this round
            . . .
        else:
            # wait for the player to enter buttons
            if clickedButton and clickedButton == pattern[currentStep]:
                # pushed the correct button
                flashButtonAnimation(clickedButton)
                currentStep += 1
                lastClickTime = time.time()

                if currentStep == len(pattern):
                    # pushed the last button in the pattern
                    iteration += 1
                    changeBackgroundAnimation()
                    score += 1
                    waitingForInput = False
                    currentStep = 0 # reset back to first step

            elif (clickedButton and clickedButton != pattern[currentStep]) or (currentStep != 0 and time.time() - TIMEOUT >
lastClickTime):
                . . .
                iteration = 0

        pygame.display.update()
        FPSCLOCK.tick(FPS)
```