오픈소스SW 12주차 보고서

2023090137 변우석

기존의 코드에서 해당 체크리스트를 해결하기 위해 노력했습니다.

* 체크리스트 (과제)

1. 현재 테트리스 게임의 배경음악을 주어진 3개의 음악 중 1개가 재생되도록 수정

- 2개의 다른 곡이 아닌 3개의 다른 곡 중 하나로 무작위 재생되게 설정

2. 상태창 이름을 학번_이름으로 수정

- 기존의 이름에서 학번_이름으로 설정

3. 게임시작화면의 문구를 MY TETRIS으로 변경

- 기존의 문구에서 MY TETRIS로 설정

4. 게임시작화면의 문구 및 배경색을 노란색으로 변경

- 기존의 검은색에서 노란색으로 설정

5. 게임 경과 시간을 초 단위로 표시 (새 게임 시작 시 0으로 초기화 되어야 함)

- 시작 시간 변수를 추가하고 해당 값을 이용해 새 게임 시작 시 시간을 0으로 초기화

6. 7개의 블록이 각각 고유의 색을 갖도록 코드를 수정하거나 추가

- 각 블록별 색을 매칭시켜놓고, 기존의 랜덤 코드에서 고정 코드로 변경

주소

https://github.com/Woos0219/week12.git

1번 부분의 기존의 코드와 수정된 코드

- while True: # game loop

```
if random.randint(0, 1) == 0:
    pygame.mixer.music.load('tetrisb.mid')
else:
    pygame.mixer.music.load('tetrisc.mid')
```

➔ while True: # game loop

```
RN = random.randint(0, 2)
If RN == 0:
    pygame.mixer.music.load('Hover.mp3')
elif RN == 1:
    pygame.mixer.music.load('Our_Lives_Past.mp3')
else:
```

pygame.mixer.music.load('Platform_9.mp3')

2번 부분의 기존의 코드와 수정된 코드

- pygame.display.set_caption('Tetromino')
  ➔ pygame.display.set_caption('2023090137_변우석')

3번 부분의 기존의 코드와 수정된 코드

- showTextScreen('Tetromino')
  ➔ showTextScreen('MY TETRIS')

4번 부분의 기존의 코드와 수정된 코드

- BGCOLOR = BLACK
  ➔ BGCOLOR = YELLOW

5번 부분의 기존의 코드와 수정된 코드

- def runGame():

```
    # setup variables for the start of the game
    board = getBlankBoard()
    lastMoveDownTime = time.time()
    lastMoveSidewaysTime = time.time()
    lastFallTime = time.time()
    movingDown = False # note: there is no movingUp variable
    movingLeft = False
    movingRight = False
    score = 0
    level, fallFreq = calculateLevelAndFallFreq(score)

    fallingPiece = getNewPiece()
    nextPiece = getNewPiece()

    while True: # game loop
        if fallingPiece == None:
            # No falling piece in play, so start a new piece at the top
            fallingPiece = nextPiece
            nextPiece = getNewPiece()
            lastFallTime = time.time() # reset lastFallTime

            if not isValidPosition(board, fallingPiece):
```

```
                return # can't fit a new piece on the board, so game over

        checkForQuit()
        for event in pygame.event.get(): # event handling loop
            if event.type == KEYUP:
                if (event.key == K_p):
                    # Pausing the game
                    DISPLAYSURF.fill(BGCOLOR)
                    pygame.mixer.music.stop()
                    showTextScreen('Paused') # pause until a key press
                    pygame.mixer.music.play(-1, 0.0)
                    lastFallTime = time.time()
                    lastMoveDownTime = time.time()
                    lastMoveSidewaysTime = time.time()
                elif (event.key == K_LEFT or event.key == K_a):
                    movingLeft = False
                elif (event.key == K_RIGHT or event.key == K_d):
                    movingRight = False
                elif (event.key == K_DOWN or event.key == K_s):
                    movingDown = False

            elif event.type == KEYDOWN:
                # moving the piece sideways
                if (event.key == K_LEFT or event.key == K_a) and isValidPosition(board, fallingPiece,
adjX=-1):
                    fallingPiece['x'] -= 1
                    movingLeft = True
                    movingRight = False
                    lastMoveSidewaysTime = time.time()

                elif (event.key == K_RIGHT or event.key == K_d) and isValidPosition(board,
fallingPiece, adjX=1):
                    fallingPiece['x'] += 1
                    movingRight = True
                    movingLeft = False
                    lastMoveSidewaysTime = time.time()
```

```
                # rotating the piece (if there is room to rotate)
                elif (event.key == K_UP or event.key == K_w):
                    fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])
                    if not isValidPosition(board, fallingPiece):
                        fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
                elif (event.key == K_q): # rotate the other direction
                    fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
                    if not isValidPosition(board, fallingPiece):
                        fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])

                # making the piece fall faster with the down key
                elif (event.key == K_DOWN or event.key == K_s):
                    movingDown = True
                    if isValidPosition(board, fallingPiece, adjY=1):
                        fallingPiece['y'] += 1
                    lastMoveDownTime = time.time()

                # move the current piece all the way down
                elif event.key == K_SPACE:
                    movingDown = False
                    movingLeft = False
                    movingRight = False
                    for i in range(1, BOARDHEIGHT):
                        if not isValidPosition(board, fallingPiece, adjY=i):
                            break
                    fallingPiece['y'] += i - 1

        # handle moving the piece because of user input
        if (movingLeft or movingRight) and time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ:
            if movingLeft and isValidPosition(board, fallingPiece, adjX=-1):
                fallingPiece['x'] -= 1
            elif movingRight and isValidPosition(board, fallingPiece, adjX=1):
```

```
                fallingPiece['x'] += 1
            lastMoveSidewaysTime = time.time()

        if movingDown and time.time() - lastMoveDownTime > MOVEDOWNFREQ and isValidPosition(board, fallingPiece, adjY=1):
            fallingPiece['y'] += 1
            lastMoveDownTime = time.time()

        # let the piece fall if it is time to fall
        if time.time() - lastFallTime > fallFreq:
            # see if the piece has landed
            if not isValidPosition(board, fallingPiece, adjY=1):
                # falling piece has landed, set it on the board
                addToBoard(board, fallingPiece)
                score += removeCompleteLines(board)
                level, fallFreq = calculateLevelAndFallFreq(score)
                fallingPiece = None
            else:
                # piece did not land, just move the piece down
                fallingPiece['y'] += 1
                lastFallTime = time.time()

        # drawing everything on the screen
        DISPLAYSURF.fill(BGCOLOR)
        drawBoard(board)
        drawStatus(score, level)
        drawNextPiece(nextPiece)
        if fallingPiece != None:
            drawPiece(fallingPiece)
```

➔ def runGame():

```
    # setup variables for the start of the game
    board = getBlankBoard()
    startTime = time.time() #추가됨
    lastMoveDownTime = time.time()
    lastMoveSidewaysTime = time.time()
    lastFallTime = time.time()
    movingDown = False # note: there is no movingUp variable
```

```
movingLeft = False
movingRight = False
score = 0
level, fallFreq = calculateLevelAndFallFreq(score)

fallingPiece = getNewPiece()
nextPiece = getNewPiece()

while True: # game loop
    startTime = time.time()
    if fallingPiece == None:
        # No falling piece in play, so start a new piece at the top
        fallingPiece = nextPiece
        nextPiece = getNewPiece()
        lastFallTime = time.time() # reset lastFallTime

        if not isValidPosition(board, fallingPiece):
            return # can't fit a new piece on the board, so game over

    checkForQuit()
    for event in pygame.event.get(): # event handling loop
        if event.type == KEYUP:
            if (event.key == K_p):
                # Pausing the game
                DISPLAYSURF.fill(BGCOLOR)
                pygame.mixer.music.stop()
                showTextScreen('Paused') # pause until a key press
                pygame.mixer.music.play(-1, 0.0)
                lastFallTime = time.time()
                lastMoveDownTime = time.time()
                lastMoveSidewaysTime = time.time()
            elif (event.key == K_LEFT or event.key == K_a):
                movingLeft = False
            elif (event.key == K_RIGHT or event.key == K_d):
                movingRight = False
            elif (event.key == K_DOWN or event.key == K_s):
                movingDown = False
```

```
            elif event.type == KEYDOWN:
                # moving the piece sideways
                if (event.key == K_LEFT or event.key == K_a) and isValidPosition(board, fallingPiece, adjX=-1):
                    fallingPiece['x'] -= 1
                    movingLeft = True
                    movingRight = False
                    lastMoveSidewaysTime = time.time()

                elif (event.key == K_RIGHT or event.key == K_d) and isValidPosition(board, fallingPiece, adjX=1):
                    fallingPiece['x'] += 1
                    movingRight = True
                    movingLeft = False
                    lastMoveSidewaysTime = time.time()

                # rotating the piece (if there is room to rotate)
                elif (event.key == K_UP or event.key == K_w):
                    fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])
                    if not isValidPosition(board, fallingPiece):
                        fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
                elif (event.key == K_q): # rotate the other direction
                    fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
                    if not isValidPosition(board, fallingPiece):
                        fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])

                # making the piece fall faster with the down key
                elif (event.key == K_DOWN or event.key == K_s):
                    movingDown = True
                    if isValidPosition(board, fallingPiece, adjY=1):
                        fallingPiece['y'] += 1
                    lastMoveDownTime = time.time()
```

```
                # move the current piece all the way down
                elif event.key == K_SPACE:
                    movingDown = False
                    movingLeft = False
                    movingRight = False
                    for i in range(1, BOARDHEIGHT):
                        if not isValidPosition(board, fallingPiece, adjY=i):
                            break
                    fallingPiece['y'] += i - 1

        # handle moving the piece because of user input
        if (movingLeft or movingRight) and time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ:
            if movingLeft and isValidPosition(board, fallingPiece, adjX=-1):
                fallingPiece['x'] -= 1
            elif movingRight and isValidPosition(board, fallingPiece, adjX=1):
                fallingPiece['x'] += 1
            lastMoveSidewaysTime = time.time()

        if movingDown and time.time() - lastMoveDownTime > MOVEDOWNFREQ and isValidPosition(board, fallingPiece, adjY=1):
            fallingPiece['y'] += 1
            lastMoveDownTime = time.time()

        # let the piece fall if it is time to fall
        if time.time() - lastFallTime > fallFreq:
            # see if the piece has landed
            if not isValidPosition(board, fallingPiece, adjY=1):
                # falling piece has landed, set it on the board
                addToBoard(board, fallingPiece)
                score += removeCompleteLines(board)
                level, fallFreq = calculateLevelAndFallFreq(score)
                fallingPiece = None
            else:
                # piece did not land, just move the piece down
                fallingPiece['y'] += 1
```

```
                    lastFallTime = time.time()

                # drawing everything on the screen
                DISPLAYSURF.fill(BGCOLOR)
                drawBoard(board)
                elapsedTime = int(time.time() - startTime)  # 경과 시간을 초 단위로 계산함.
                drawStatus(score, level)
                drawNextPiece(nextPiece)
                if fallingPiece != None:
                    drawPiece(fallingPiece)
```

6번 부분의 기존의 코드와 수정된 코드

- PIECES = {'S': S_SHAPE_TEMPLATE,
  'Z': Z_SHAPE_TEMPLATE,
  'J': J_SHAPE_TEMPLATE,
  'L': L_SHAPE_TEMPLATE,
  'I': I_SHAPE_TEMPLATE,
  'O': O_SHAPE_TEMPLATE,
  'T': T_SHAPE_TEMPLATE}

  'color': random.randint(0, len(COLORS)-1)}

  if pixelx == None and pixely == None:
  pixelx, pixely = convertToPixelCoords(boxx, boxy)
  pygame.draw.rect(DISPLAYSURF, COLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 1, BOXSIZE - 1))
  pygame.draw.rect(DISPLAYSURF, LIGHTCOLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 4, BOXSIZE - 4))

  ➔ PIECES = {'S': S_SHAPE_TEMPLATE,
  'Z': Z_SHAPE_TEMPLATE,
  'J': J_SHAPE_TEMPLATE,
  'L': L_SHAPE_TEMPLATE,
  'I': I_SHAPE_TEMPLATE,
  'O': O_SHAPE_TEMPLATE,
  'T': T_SHAPE_TEMPLATE}

  PIECE_COLORS = {

```
        'S': GREEN,
        'Z': RED,
        'J': BLUE,
        'L': ORANGE,
        'I': CYAN,
        'O': YELLOW,
        'T': MAGENTA
} # 추가

'color': PIECE_COLORS[shape]} # 변경

if pixelx == None and pixely == None:
            pixelx, pixely = convertToPixelCoords(boxx, boxy)
        pygame.draw.rect(DISPLAYSURF, color, (pixelx + 1, pixely + 1, BOXSIZE - 1, BOXSIZE - 1))
        pygame.draw.rect(DISPLAYSURF, LIGHTCOLORS[COLORS.index(color)], (pixelx + 1, pixely + 1,
BOXSIZE - 4, BOXSIZE - 4)) # 변경
```