오픈소스SW 과제중심수업 보고서

ICT융합학부 미디어테크놀리지 전공

2018045623 양재혁

GitHub repository 주소: https://github.com/gndksakxk10/osw

1. 각 함수들의 역할

```
def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT, BIGFONT
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
    BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
    BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
    pygame.display.set_caption('2018045623 YANGJAEHYUK')



    showTextScreen('MY TETRIS')
    while True: # game loop
        if random.randint(0, 2) == 0:
            pygame.mixer.music.load('Hover.mp3')
        elif random.randint(0, 2) == 1:
            pygame.mixer.music.load('Our_Lives_Past.mp3')
        else:
            pygame.mixer.music.load('Platform_9.mp3')
        pygame.mixer.music.play(-1, 0.0)
        runGame()
        pygame.mixer.music.stop()
        showTextScreen('Over :(')
```

위 함수는 main함수입니다. main함수에서는 실제 게임을 수행하는 역할을 맡고 있습니다. 전역변수들과 게임을 실행하는데 필요한 조건들이 있습니다.

```
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)
    start = time.time()



    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('Get a rest!') # 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)
    drawtimer(start)
    drawNextPiece(nextPiece)
```

```
    if fallingPiece != None:
        drawPiece(fallingPiece)

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

위 함수는 runGame함수입니다. runGame함수는 새로운 게임을 시작하는데 필요한 부분들을 설정하는 역할을 맡고 있습니다.

```
def makeTextObjs(text, font, color):
    surf = font.render(text, True, color)
    return surf, surf.get_rect()
```

위 함수는 makeTextObjs함수입니다. makeTextObjs함수는 텍스트를 만드는 단축함수 역할을 맡고 있습니다.

```
def terminate():
    pygame.quit()
    sys.exit()
```

위 함수는 terminate함수입니다. terminate함수는 게임을 종료하는 역할을 맡고 있습니다.

```
def checkForKeyPress():
    # Go through event queue looking for a KEYUP event.
    # Grab KEYDOWN events to remove them from the event queue.
    checkForQuit()

    for event in pygame.event.get([KEYDOWN, KEYUP]):
        if event.type == KEYDOWN:
            continue
        return event.key
    return None
```

위 함수는 checkForKeyPress함수입니다. KEYUP이벤트가 발생하였는지 이벤트 큐에서 찾고 KEYDOWN이벤트를 찾아 이벤트 큐에서 제거하는 역할을 맡고 있습니다.

```
def showTextScreen(text):
    # This function displays large text in the
    # center of the screen until a key is pressed.
    # Draw the text drop shadow
    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTSHADOWCOLOR)
    titleRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))
    DISPLAYSURF.blit(titleSurf, titleRect)

    # Draw the text
    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTCOLOR)
    titleRect.center = (int(WINDOWWIDTH / 2) - 3, int(WINDOWHEIGHT / 2) - 3)
    DISPLAYSURF.blit(titleSurf, titleRect)

    # Draw the additional "Press a key to play! pause key is p" text.
    pressKeySurf, pressKeyRect = makeTextObjs('Press a key to play! pause key is p', BASICFONT, TEXTCOLOR)
    pressKeyRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2) + 100)
    DISPLAYSURF.blit(pressKeySurf, pressKeyRect)

    while checkForKeyPress() == None:
        pygame.display.update()
        FPSCLOCK.tick()
```

위 함수는 showTextScreen함수입니다. 위 함수는 플레이어가 키를 누를 때까지 글자를 그려주는 역할을 합니다.

```
def checkForQuit():
    for event in pygame.event.get(QUIT): # get all the QUIT events
        terminate() # terminate if any QUIT events are present
    for event in pygame.event.get(KEYUP): # get all the KEYUP events
        if event.key == K_ESCAPE:
            terminate() # terminate if the KEYUP event was for the Esc key
        pygame.event.post(event) # put the other KEYUP event objects back
```

위 함수는 checkForQuit함수입니다. 위 함수는 프로그램을 종료하는 특정 이벤트들을 처리하는 역할을 맡고 있습니다.

```
def calculateLevelAndFallFreq(score):
    # Based on the score, return the level the player is on and
    # how many seconds pass until a falling piece falls one space.
    level = int(score / 10) + 1
    fallFreq = 0.27 - (level * 0.02)
    return level, fallFreq
```

위 함수는 calculateLevelAndFallFreq함수입니다. 위 함수는 플레이어의 점수에 따라 레벨을 올리고 떨어지는 속도를 조정해주는 역할을 맡고 있습니다.

```
def getNewPiece():
    shape = random.choice(list(PIECES.keys()))
    newPiece = {'shape': shape,
                'rotation': random.randint(0, len(PIECES[shape]) - 1),
                'x': int(BOARDWIDTH / 2) - int(TEMPLATEWIDTH / 2),
                'y': -2, # start it above the board (i.e. less than 0)
                'color': (list(PIECES.keys()).index(shape))%len(COLORS)}
    return newPiece
```

위 함수는 getNewPiece함수입니다. 위 함수는 새로운 피스를 하나 만드는 역할을 맡고 있습니다.

```
def addToBoard(board, piece):
    # fill in the board based on piece's location, shape, and rotation
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            if PIECES[piece['shape']][piece['rotation']][y][x] != BLANK:
                board[x + piece['x']][y + piece['y']] = piece['color']
```

위 함수는 addToBoard함수입니다. 위 함수는 보드 데이터 구조에 피스를 추가하는 역할을 맡고 있습니다. 피스의 위치, 형태, 회전 여부에 따라 보드를 채우게 됩니다.

```
def getBlankBoard():
    # create and return a new blank board data structure
    board = []
    for i in range(BOARDWIDTH):
        board.append([BLANK] * BOARDHEIGHT)
    return board
```

위 함수는 getBlankBoard함수입니다. 비어 있는 새로운 보드 데이터 구조를 만들어 반환하는 역할을 맡고 있습니다.

```
def isOnBoard(x, y):
    return x >= 0 and x < BOARDWIDTH and y < BOARDHEIGHT
```

위 함수는 isOnBoard함수입니다. x와 y좌표가 보드에 있는지 검사하는 역할을 맡고 있습니다.

```
def isValidPosition(board, piece, adjX=0, adjY=0):
    # Return True if the piece is within the board and not colliding
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            isAboveBoard = y + piece['y'] + adjY < 0
            if isAboveBoard or PIECES[piece['shape']][piece['rotation']][y][x] == BLANK:
                continue
            if not isOnBoard(x + piece['x'] + adjX, y + piece['y'] + adjY):
                return False
            if board[x + piece['x'] + adjX][y + piece['y'] + adjY] != BLANK:
                return False
    return True
```

위 함수는 isValidPosition함수입니다. 위 함수는 board 데이터 구조와 piece 데이터 구조를 제공하고, piece가 board위에 있고 겹치지 않았다면 true값을 return해주는 역할을 맡고 있습니다.

```
def isCompleteLine(board, y):
    # Return True if the line filled with boxes with no gaps.
    for x in range(BOARDWIDTH):
        if board[x][y] == BLANK:
            return False
    return True
```

위 함수는 isCompleteLine함수입니다. 위 함수는 줄에 상자가 채워져 있어 빈칸이 존재하는지 검사하는 역할을 맡고 있습니다.

```
def removeCompleteLines(board):
    # Remove any completed lines on the board, move everything above them down, and return the number of complete lines.
    numLinesRemoved = 0
    y = BOARDHEIGHT - 1 # start y at the bottom of the board
    while y >= 0:
        if isCompleteLine(board, y):
            # Remove the line and pull boxes down by one line.
            for pullDownY in range(y, 0, -1):
                for x in range(BOARDWIDTH):
                    board[x][pullDownY] = board[x][pullDownY-1]
            # Set very top line to blank.
            for x in range(BOARDWIDTH):
                board[x][0] = BLANK
            numLinesRemoved += 1
            # Note on the next iteration of the loop, y is the same.
            # This is so that if the line that was pulled down is also
            # complete, it will be removed.
        else:
            y -= 1 # move on to check next row up
    return numLinesRemoved
```

위 함수는 removeCompleteLines함수입니다. 보드가 완벽하게 완성된 줄을 제거하고 그 줄 위에 있던 상자들을 내립니다. 그리고 몇 개의 줄이 없어졌는지 반환하여 score에 더해주는 역할을 합니다.

```
def convertToPixelCoords(boxx, boxy):
    # Convert the given xy coordinates of the board to xy
    # coordinates of the location on the screen.
    return (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))
```

위 함수는 convertToPixelCoords함수입니다. 위 함수는 주어진 보드 좌표계를 픽셀 좌표계로 변환하는 역할을 합니다.

```
def drawBox(boxx, boxy, color, pixelx=None, pixely=None):
    # draw a single box (each tetromino piece has four boxes)
    # at xy coordinates on the board. Or, if pixelx & pixely
    # are specified, draw to the pixel coordinates stored in
    # pixelx & pixely (this is used for the "Next" piece).
    if color == BLANK:
        return
    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))
```

위 함수는 drawBox함수입니다. 보드 x, y좌표에 상자를 그리는 역할을 합니다.

```
def drawBoard(board):
    # draw the border around the board
    pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (XMARGIN - 3, TOPMARGIN - 7, (BOARDWIDTH * BOXSIZE) + 8, (BOARDHEIGHT * BOXSIZE) + 8), 5)

    # fill the background of the board
    pygame.draw.rect(DISPLAYSURF, BGCOLOR, (XMARGIN, TOPMARGIN, BOXSIZE * BOARDWIDTH, BOXSIZE * BOARDHEIGHT))
    # draw the individual boxes on the board
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            drawBox(x, y, board[x][y])
```

위 함수는 drawBoard함수입니다. 보드 주위에 테두리를 그리고 보드 배경을 채우고 보드에 각 상자를 그리는 역할을 합니다.

```
def drawStatus(score, level):
    # draw the score text
    scoreSurf = BASICFONT.render('Score: %s' % score, True, TEXTCOLOR)
    scoreRect = scoreSurf.get_rect()
    scoreRect.topleft = (WINDOWWIDTH - 150, 20)
    DISPLAYSURF.blit(scoreSurf, scoreRect)

    # draw the level text
    levelSurf = BASICFONT.render('Level: %s' % level, True, TEXTCOLOR)
    levelRect = levelSurf.get_rect()
    levelRect.topleft = (WINDOWWIDTH - 150, 50)
    DISPLAYSURF.blit(levelSurf, levelRect)
```

위 함수는 drawStatus함수입니다. 위 함수는 점수와 현재 레벨 텍스트를 그립니다.

```
def drawtimer(start):
    # draw the time text
    timer=time.time()-start
    timerSurf = BASICFONT.render('Play Time: %s sec' % int(timer), True, TEXTCOLOR)
    timerRect = timerSurf.get_rect()
    timerRect.topleft = (WINDOWWIDTH - 630, 10)
    DISPLAYSURF.blit(timerSurf,timerRect)
```

위 함수는 drawtimer함수입니다. 과제를 위해 제가 추가한 함수이며 play Time을 그립니다.

```
def drawPiece(piece, pixelx=None, pixely=None):
    shapeToDraw = PIECES[piece['shape']][piece['rotation']]
    if pixelx == None and pixely == None:
        # if pixelx & pixely hasn't been specified, use the location stored in the piece data structure
        pixelx, pixely = convertToPixelCoords(piece['x'], piece['y'])

    # draw each of the boxes that make up the piece
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            if shapeToDraw[y][x] != BLANK:
                drawBox(None, None, piece['color'], pixelx + (x * BOXSIZE), pixely + (y * BOXSIZE))
```

위 함수는 drawPiece함수입니다. 전달되는 조각 데이터 구조에 따라 조각의 상자를 그리는 역할을 합니다. 이는 떨어지는 피스나 다음에 나올 피스를 그릴 때 사용됩니다.

```
def drawNextPiece(piece):
    # draw the "next" text
    nextSurf = BASICFONT.render('Next:', True, TEXTCOLOR)
    nextRect = nextSurf.get_rect()
    nextRect.topleft = (WINDOWWIDTH - 120, 80)
    DISPLAYSURF.blit(nextSurf, nextRect)
    # draw the "next" piece
    drawPiece(piece, pixelx=WINDOWWIDTH-120, pixely=100)



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

위 함수는 drawNextPiece함수입니다. 화면 우측에 다음 피스를 그리는 역할을 합니다.

2. 함수의 호출 순서 또는 호출 조건

```
def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT, BIGFONT
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
    BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
    BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
    pygame.display.set_caption('2018045623 YANGJAEHYUK')




    showTextScreen('MY TETRIS')
    while True: # game loop
        if random.randint(0, 2) == 0:
            pygame.mixer.music.load('Hover.mp3')
        elif random.randint(0, 2) == 1:
            pygame.mixer.music.load('Our_Lives_Past.mp3')
        else:
            pygame.mixer.music.load('Platform_9.mp3')
        pygame.mixer.music.play(-1, 0.0)
        runGame()
        pygame.mixer.music.stop()
        showTextScreen('Over :(')
```

위 함수는 main함수입니다. 전역변수 fpsclock, displaysurf, basicfont, bigfont를 설정하고 pygame.init()으로 초기화합니다. 루프문을 돌고 게임이 실행되는 동안 3개의 음악 중 하나를 무작위로 틀고, runGame함수를 호출해야 게임을 시작합니다. 게임이 끝나면 음악을 멈추고 Over:(를 뜨게 해야 합니다.

```
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)
    start = time.time()



    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
```

위 함수는 runGame함수입니다. 시작 전에 변수들을 초기화 합니다. fallingPiece변수는 현재 떨어지고 있는 피스를 플레이어가 회전할 수 있는 피스로 설정합니다. nextPiece변수는 다음에 떨어뜨릴 피스입니다. 호출 순서는 while true가 게임 루프로써 만약 떨어지는 피스가 없다면 nextPiece변수를 fallingPiece변수에 복사하여 임의의 피스를 넣습니다. getNewPiece()함수에서 새로운 피스를 생성할 수 있습니다. lastFallTime변수는 현재시간으로 재설정됩니다. isValidPosition이 없다면 새로운 피스가 board에 들어갈 수 없게 되므로 게임이 끝나게 됩니다. 즉 다시 runGame()함수를 반환하게 됩니다.

```
        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('Get a rest!') # 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
```

위 함수는 runGame함수의 연장선입니다. 호출 조건으로는 플레이어가 p키를 누르면 보드를 가리고 음악을 멈추고 스크린에 Ger a rest!를 띄웁니다. 이는 p키를 누른 상태이며 다른 키를 누를 때까지 멈추어야 합니다. 플레이어가 다른 키를 누르면 showTextScreen이 반환되며 음악이 재개되고 lastFallTime, lastMoveDownTime, lastMoveSidewaysTime변수는 현재시간으로 재설정됩니다. elif 부분부터는 움직임관련 변수를 사용하여 플레이어의 입력을 처리하는 부분입니다. 플레이어가 키에서 손을 땐 경우 그 방향으로 움직이려고 하지 않음을 의미하므로 False값이 되야 합니다.

```
        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
```

위 함수는 runGame함수의 연장선입니다. 플레이어가 여러 키를 눌렀을 때 피스들이 밀거나 회전하는 것이 유효한지 확인하는 부분입니다. 호출 조건으로는 방향키 왼쪽 버튼이나 a키를 입력하고 adjx값에 -1을 대입하면 피스가 왼쪽을 한 칸 움직여야 합니다. 방향키 오른쪽 버튼이나 d키를 입력하고 adjx값에 +1을 대입하면 피스가 오른쪽으로 한 칸 움직여야 합니다. 회전을 시킬 때 회전하지 못하는 상황이라면 -1을 해서 원래의 값으로 돌려놓아야 합니다. q를 누르면 반대 방향으로 돌려야 하고 space를 누르면 아래로 빠르게 내려가야 합니다. 여기서 아래로 착지하기 위해 얼마나 많은 공간이 있어야 하는지 파악해야 합니다. isValidPosition함수가 False를 반환하면 더 이상 아래로 내려갈 수 없고 True를 반환해야 피스가 한 칸 아래로 내려갈 수 있습니다.

```
        # 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)
        drawtimer(start)
        drawNextPiece(nextPiece)


        if fallingPiece != None:
            drawPiece(fallingPiece)

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

위 함수는 runGame함수의 연장선입니다. 처음 나오는 if문을 보면 (movingLeft or movingRight) and time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ: 인 경우 그 안의 if문을 보면 movingLeft and isValidPosition(board, fallingPiece, adjX=-1): 이면 fallingPiece['x']에 -1을 합니다. elif문을 보면 adjX값이 1이라면 fallingPiece['x']에 +1을 합니다. lastMoveSidewaysTime을 time.time()으로 덮어씁니다. 그 다음 if문을 보면 떨어지는 피스가 마지막으로 한 공간에 떨어진 이후 충분한 시간이 경과한 경우를 나타낸 것입니다. if not문은 떨어지는 피스가 착지했으면 보드에 둔다는 내용입니다. removeCompleteLines() 함수는 제거된 줄 수의 정수 값을 반환하므로 score에 이 숫자를 더합니다. 따라서 떨어지고 있는 피스는 none이 됩니다. else문을 보면 피스가 아직 착지하지 않았다면 피스를 아래로 움직인다는 내용입니다. 떨어지고 있는 피스 'y에 +1을 하고 lastFallTime에 time.time()으로 덮어씁니다. 그 다음으로 화면에 무언가 그리는 기능은 다른 함수에서 처리하므로 해당 함수를 호출만 하면 됩니다. 그런 다음 pygame.display.update()에 대한 호출로 인해 디스플레이 Surface가 실제 컴퓨터 화면에 표시되고, FPSCLOCK함수로 게임 실행 시간을 조정해줍니다.

```
def makeTextObjs(text, font, color):
    surf = font.render(text, True, color)
    return surf, surf.get_rect()
```

위 함수는 makeTextObjs함수입니다. 호출 조건으로는 파라미터로 text, font, color객체를 받아야

합니다. 호출 순서로는 render method를 호출한 후 값을 surf에 저장하고 surf와 surf.get_rect()객체를 반환합니다.

```
def terminate():
    pygame.quit()
    sys.exit()
```

위 함수는 terminate함수입니다. 호출 순서로는 pygame이 끝나고 시스템이 종료되어야 합니다.

```
def checkForKeyPress():
    # Go through event queue looking for a KEYUP event.
    # Grab KEYDOWN events to remove them from the event queue.
    checkForQuit()

    for event in pygame.event.get([KEYDOWN, KEYUP]):
        if event.type == KEYDOWN:
            continue
        return event.key
    return None
```

위 함수는 checkForKeyPress함수입니다. 함수 호출 조건으로는 QUIT이벤트가 없으면 큐에서 모든 KEYUP/DOWN 이벤트를 가져와야 합니다. 호출 순서로는 KEYDOWN이 이벤트면 무시하고 이벤트 키가 없으면 None을 반환합니다.

```
def showTextScreen(text):
    # This function displays large text in the
    # center of the screen until a key is pressed.
    # Draw the text drop shadow
    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTSHADOWCOLOR)
    titleRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))
    DISPLAYSURF.blit(titleSurf, titleRect)

    # Draw the text
    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTCOLOR)
    titleRect.center = (int(WINDOWWIDTH / 2) - 3, int(WINDOWHEIGHT / 2) - 3)
    DISPLAYSURF.blit(titleSurf, titleRect)

    # Draw the additional "Press a key to play! pause key is p" text.
    pressKeySurf, pressKeyRect = makeTextObjs('Press a key to play! pause key is p', BASICFONT, TEXTCOLOR)
    pressKeyRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2) + 100)
    DISPLAYSURF.blit(pressKeySurf, pressKeyRect)

    while checkForKeyPress() == None:
        pygame.display.update()
        FPSCLOCK.tick()
```

위 함수는 showTextScreen함수입니다. 함수 호출 조건으로는 Surf, Rect을 makeTextObjs함수로부터 받아야합니다. 이로써 글자를 그릴 수 있게 됩니다. 그 다음에는 루프문을 돕니다. checkForKeyPress()값이 none이면 화면을 업데이트하고 프레임을 조정해줍니다.

```
def checkForQuit():
    for event in pygame.event.get(QUIT): # get all the QUIT events
        terminate() # terminate if any QUIT events are present
    for event in pygame.event.get(KEYUP): # get all the KEYUP events
        if event.key == K_ESCAPE:
            terminate() # terminate if the KEYUP event was for the Esc key
        pygame.event.post(event) # put the other KEYUP event objects back
```

위 함수는 checkForQuit함수입니다. 첫번째 for문에서 QUIT이벤트를 가져오고 QUIT이벤트가 있다면 terminate합니다. 다음 for문에서 KEYUP이벤트를 가져오고 ESC key에 KEYUP이벤트가 있는 경우 terminate합니다.

```
def calculateLevelAndFallFreq(score):
    # Based on the score, return the level the player is on and
    # how many seconds pass until a falling piece falls one space.
    level = int(score / 10) + 1
    fallFreq = 0.27 - (level * 0.02)
    return level, fallFreq
```

위 함수는 calculateLevelAndFallFreq함수입니다. 호출 순서로는 level값을 지정해주고 그 level값을 토대로 fallFreq값을 지정해줍니다. 그리고 level과 fallfreq를 반환합니다.

```
def getNewPiece():
    shape = random.choice(list(PIECES.keys()))
    newPiece = {'shape': shape,
                'rotation': random.randint(0, len(PIECES[shape]) - 1),
                'x': int(BOARDWIDTH / 2) - int(TEMPLATEWIDTH / 2),
                'y': -2, # start it above the board (i.e. less than 0)
                'color': (list(PIECES.keys()).index(shape))%len(COLORS)}
    return newPiece
```

위 함수는 getNewPiece함수입니다. 호출 순서로는 먼저 임의로 피스의 모양을 설정하기 위해 PIECES.keys()를 호출하여 가능한 모든 shape 목록을 만듭니다. 그 다음에 newPiece값에 rotation, x, y, color값을 설정해줍니다. 그러고 나서 newPiece를 return해줍니다.

```
def addToBoard(board, piece):
    # fill in the board based on piece's location, shape, and rotation
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            if PIECES[piece['shape']][piece['rotation']][y][x] != BLANK:
                board[x + piece['x']][y + piece['y']] = piece['color']
```

위 함수는 addToBoard함수입니다. 호출 순서로는 피스의 바닥 데이터 구조를 받아 board데이터 구조에 상자들을 먼저 추가합니다. 다음으로 피스의 바닥이나 다른 피스 위에 완전히 착지하고 나면 데이터 구조를 갱신하게 됩니다. 결국 이는 중첩 for문으로 피스 데이터 구조의 모든 공간을 살펴본 다음, 상자를 찾으면 이를 보드에 추가하는 순서를 가지고 있습니다.

```
def getBlankBoard():
    # create and return a new blank board data structure
    board = []
    for i in range(BOARDWIDTH):
        board.append([BLANK] * BOARDHEIGHT)
    return board
```

위 함수는 getBlankBoard함수입니다. 함수의 호출 조건으로는 빈 보드를 만들기 위해서는 board.append를 사용하여 BLANK값 목록을 만들어야 합니다. 조건에 맞으면 board를 return합니다.

```
def isOnBoard(x, y):
    return x >= 0 and x < BOARDWIDTH and y < BOARDHEIGHT
```

위 함수는 isOnBoard입니다. 함수의 호출 조건으로는 전달받은 x와 y좌표를 보고, x의 좌표가 BOARDWIDTH보다 작고 y좌표가 BOARDHEIGHT보다 작고 X가 0보다 크거나 같아야 true를 반환합니다.

```
def isValidPosition(board, piece, adjX=0, adjY=0):
    # Return True if the piece is within the board and not colliding
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            isAboveBoard = y + piece['y'] + adjY < 0
            if isAboveBoard or PIECES[piece['shape']][piece['rotation']][y][x] == BLANK:
                continue
            if not isOnBoard(x + piece['x'] + adjX, y + piece['y'] + adjY):
                return False
            if board[x + piece['x'] + adjX][y + piece['y'] + adjY] != BLANK:
                return False
    return True
```

위 함수는 isValidPosition함수입니다. 함수의 호출 순서로는 피스의 x와 y좌표를 받습니다. 그리고 이 피스가 다른 상자들과 겹치는지 알아보기 위해서 for문을 돌립니다. for문안의 if문에서 피스의 공간이 보드위에 있는지 비어 있는지 확인합니다. 두 경우 중 하나의 경우를 만족하면 continue 합니다. if not문에서 피스의 상자가 보드에 없는지 확인하고 다음 if문에서 피스의 상자가 있는 보드공간이 비어 있지 않은 지 확인합니다. 두 조건 중 하나를 만족하면 isValidPosition함수가 false로 반환됩니다. 또한 false를 반환할 이유를 찾지 못하면 isValidPosition함수는 true로 반환이 됩니다.

```
def isCompleteLine(board, y):
    # Return True if the line filled with boxes with no gaps.
    for x in range(BOARDWIDTH):
        if board[x][y] == BLANK:
            return False
    return True
```

위 함수는 isCompleteLine함수입니다. 함수의 호출 순서로는 y파라미터로 지정한 가로 행에 대한 단순한 검사를 진행하여 모든 상자가 채워졌을 경우 true를 반환하고 그렇지 않으면(if문) false를 반환하는 순서입니다.

```
def removeCompleteLines(board):
    # Remove any completed lines on the board, move everything above them down, and return the number of complete lines.
    numLinesRemoved = 0
    y = BOARDHEIGHT - 1 # start y at the bottom of the board
    while y >= 0:
        if isCompleteLine(board, y):
            # Remove the line and pull boxes down by one line.
            for pullDownY in range(y, 0, -1):
                for x in range(BOARDWIDTH):
                    board[x][pullDownY] = board[x][pullDownY-1]
            # Set very top line to blank.
            for x in range(BOARDWIDTH):
                board[x][0] = BLANK
            numLinesRemoved += 1
            # Note on the next iteration of the loop, y is the same.
            # This is so that if the line that was pulled down is also
            # complete, it will be removed.
        else:
            y -= 1 # move on to check next row up
    return numLinesRemoved
```

위 함수는 removeCompleteLines함수입니다. 넘겨준 보드 데이터 구조에서 완성된 줄을 먼저 찾아 제거하고 다른 줄들을 아래로 내립니다. while루프를 통해 첫 if문을 보면 행을 한 줄 없애고 상자를 한 줄 아래로 내리면 for문을 통해 맨 위의 줄을 빈칸으로 설정합니다. 그 다음 for문을 통해 반복문에서 처음 y는 그대로인데 한 칸 아래로 내린 줄 또한 완성된 줄이면 그 줄도 없앱니다. else에서는 한 줄 씩 올라가면서 검사하는 순서입니다.

```
def convertToPixelCoords(boxx, boxy):
    # Convert the given xy coordinates of the board to xy
    # coordinates of the location on the screen.
    return (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))
```

위 함수는 convertToPixelCoords함수입니다. 호출 순서로는 boxx와 boxy값을 받아 (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE)) 값으로 return 하는 순서입니다.

```
def drawBox(boxx, boxy, color, pixelx=None, pixely=None):
    # draw a single box (each tetromino piece has four boxes)
    # at xy coordinates on the board. Or, if pixelx & pixely
    # are specified, draw to the pixel coordinates stored in
    # pixelx & pixely (this is used for the "Next" piece).
    if color == BLANK:
        return
    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))
```

위 함수는 drawBox함수입니다. 함수 호출 조건으로는 color가 없어야 합니다. 함수 호출 순서로는 보드 좌표에 대한 boxx, boxy매개변수와 pixelx, pixely가 none으로 받습니다. pixelx, pixely가 둘 다 none이면 convertToPixelCoords()의 반환 값으로 none을 덮어씁니다. 이 호출은 boxx 및 boxy에서 지정한 보드 좌표의 픽셀 좌표를 가져옵니다. 그리고 다음 pygame.draw.rect에서 상자들을 그리게 됩니다.

```
def drawBoard(board):
    # draw the border around the board
    pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (XMARGIN - 3, TOPMARGIN - 7, (BOARDWIDTH * BOXSIZE) + 8, (BOARDHEIGHT * BOXSIZE) + 8), 5)

    # fill the background of the board
    pygame.draw.rect(DISPLAYSURF, BGCOLOR, (XMARGIN, TOPMARGIN, BOXSIZE * BOARDWIDTH, BOXSIZE * BOARDHEIGHT))
    # draw the individual boxes on the board
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            drawBox(x, y, board[x][y])
```

위 함수는 drawBoard함수입니다. 함수 호출 순서로는 pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (XMARGIN - 3, TOPMARGIN - 7, (BOARDWIDTH * BOXSIZE) + 8, (BOARDHEIGHT * BOXSIZE) + 8), 5)를 통해 먼저 보드 주변에 테두리를 그립니다. 다음으로는 pygame.draw.rect(DISPLAYSURF, BGCOLOR, (XMARGIN, TOPMARGIN, BOXSIZE * BOARDWIDTH, BOXSIZE * BOARDHEIGHT))를 통해 보드의 배경색을 채웁니다. 이중 for문을 사용해 보드의 각 상자를 그리게 됩니다.

```
def drawStatus(score, level):
    # draw the score text
    scoreSurf = BASICFONT.render('Score: %s' % score, True, TEXTCOLOR)
    scoreRect = scoreSurf.get_rect()
    scoreRect.topleft = (WINDOWWIDTH - 150, 20)
    DISPLAYSURF.blit(scoreSurf, scoreRect)

    # draw the level text
    levelSurf = BASICFONT.render('Level: %s' % level, True, TEXTCOLOR)
    levelRect = levelSurf.get_rect()
    levelRect.topleft = (WINDOWWIDTH - 150, 50)
    DISPLAYSURF.blit(levelSurf, levelRect)
```

위 함수는 drawStatus함수입니다. 함수 호출 순서로는 scoreSurf값과 scoreRect값을 설정하고 이 값들을 이용해 DISPLAYSURF.blit를 호출합니다. 그 다음에는 levelSurf값과 levelRect값을 설정하고 이 값들을 이용해 DISPLAYSURF.blit를 호출합니다.

```
def drawtimer(start):
    # draw the time text
    timer=time.time()-start
    timerSurf = BASICFONT.render('Play Time: %s sec' % int(timer), True, TEXTCOLOR)
    timerRect = timerSurf.get_rect()
    timerRect.topleft = (WINDOWWIDTH - 630, 10)
    DISPLAYSURF.blit(timerSurf,timerRect)
```

위 함수는 drawtimer함수입니다. 제가 과제를 하면서 새로 만든 함수입니다. 함수 호출 순서는 timer값을 생성해 time.time()에서 runGame에서 선언한 start값을 뺍니다. 이후에는 drawStatus함수처럼 따로 surf와 rect값을 설정하고 이 값들을 이용해 DISPLAYSURF.blit를 호출합니다.

```
def drawPiece(piece, pixelx=None, pixely=None):
    shapeToDraw = PIECES[piece['shape']][piece['rotation']]
    if pixelx == None and pixely == None:
        # if pixelx & pixely hasn't been specified, use the location stored in the piece data structure
        pixelx, pixely = convertToPixelCoords(piece['x'], piece['y'])

    # draw each of the boxes that make up the piece
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            if shapeToDraw[y][x] != BLANK:
                drawBox(None, None, piece['color'], pixelx + (x * BOXSIZE), pixely + (y * BOXSIZE))
```

위 함수는 drawPiece함수입니다. 호출 순서로 전달되는 피스 데이터 구조에 따라 피스의 상자를 그립니다. pixelx와 pixely의 인수가 none이면 convertToPixelCoord() 호출의 반환 값으로 해당 변수를 덮어씁니다. 다음으로 이중 for문을 이용해 피스를 구성하는 상자를 그립니다. 이 때 그려야 할 각 상자에 대해 drawBox()를 호출합니다.

```
def drawNextPiece(piece):
    # draw the "next" text
    nextSurf = BASICFONT.render('Next:', True, TEXTCOLOR)
    nextRect = nextSurf.get_rect()
    nextRect.topleft = (WINDOWWIDTH - 120, 80)
    DISPLAYSURF.blit(nextSurf, nextRect)
    # draw the "next" piece
    drawPiece(piece, pixelx=WINDOWWIDTH-120, pixely=100)


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

위 함수는 drawNextPiece함수입니다. 함수 호출 순서로는 먼저 next 텍스트를 그려야 하기 때문에, drawStatus함수처럼 nextSurf값과 nextRect값을 설정하고 이 값들을 이용해 DISPLAYSURF.blit를 호출합니다. 다음으로는 next 피스를 그려야 하기 때문에 drawPiece()의 픽셀 및 픽셀 매개 변수에 대한 인수를 전달합니다.