# 오픈소스SW 과제중심수업 보고서

ICT융합학부 미디어테크 전공

2020079343 우지영

Github repository 주소 : https://github.com/zi0w/osw

## 1. 각 함수들의 역할

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


FPS = 25
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
BOXSIZE = 20
BOARDWIDTH = 10
BOARDHEIGHT = 20
BLANK = '.'
```

테트리스 게임에 사용되는 상수들이다. FPS, WINDOWWIDTH, WINDOWHEIGHT, BOXSIZE, BOARDWIDTH, BOARDHEIGT, BLANK 등 실행 창의 크기, 박스의 크기, 테트리스 게임이 실행되는 칸의 크기 등을 정해놓은 것이다.

```
MOVESIDEWAYSFREQ = 0.15
MOVEDOWNFREQ = 0.1
```

플레이어가 좌,우 화살표 키를 이용해 테트리스 블록을 이동할 때 0.15초마다 다른 공간으로 이동할 수 있고, 아래쪽 화살표 키를 누른 상태에서 얼마나 조각이 떨어질 수 있는지 정해놓은 상수들이다.

```
XMARGIN = int((WINDOWWIDTH - BOARDWIDTH * BOXSIZE) / 2)
TOPMARGIN = WINDOWHEIGHT - (BOARDHEIGHT * BOXSIZE) - 5
```

전체 창 너비의 총 픽셀수, 박스 폭, 게임 보드의 크기 등을 계산하여 적절한 크기의 게임 창과 테트리스가 진행될 박스의 크기를 정한 것이다.

```
#               R    G    B
WHITE       = (255, 255, 255)
GRAY        = (185, 185, 185)
BLACK       = (  0,   0,   0)
RED         = (155,   0,   0)
LIGHTRED    = (175,  20,  20)
GREEN       = (  0, 155,   0)
LIGHTGREEN  = ( 20, 175,  20)
BLUE        = (  0,   0, 155)
LIGHTBLUE   = ( 20,  20, 175)
YELLOW      = (155, 155,   0)
LIGHTYELLOW = (175, 175,  20)
PURPLE      = (105,   0, 255)
LIGHTPURPLE = (115,  95, 255)
ORANGE      = (255, 135,   0)
LIGHTORANGE = (255, 150,  60)
PINK        = (255,   0, 205)
LIGHTPINK   = (255, 110, 255)

BORDERCOLOR = BLUE
BGCOLOR = BLACK
TEXTCOLOR = LIGHTYELLOW
TEXTSHADOWCOLOR = YELLOW
COLORS      = (     BLUE,      GREEN,      RED,      YELLOW,      PURPLE,      PINK,      ORANGE)
LIGHTCOLORS = (LIGHTBLUE, LIGHTGREEN, LIGHTRED, LIGHTYELLOW, LIGHTPURPLE, LIGHTPINK, LIGHTORANGE)
assert len(COLORS) == len(LIGHTCOLORS) # each color must have light color
```

테트리스 게임에서 사용될 색깔들을 정의해 놓았다. 테트리스 블록에 명암을 주기 위해 연한 색깔도 같이 사용했기에 같이 정의되어 있다. 나는 과제를 수행하기 위해 세가지 색깔을 추가로 정의해주었다. 색상들은 일반 색상과, 연한 색상이라는 튜플에 저장되어 있다.

```
S_SHAPE_TEMPLATE = [['.....',
                     '.....',
                     '..OO.',
                     '.OO..',
                     '.....'],
                    ['.....',
                     '..O..',
                     '..OO.',
                     '...O.',
                     '.....']]

Z_SHAPE_TEMPLATE = [['.....',
                     '.....',
                     '.OO..',
                     '..OO.',
                     '.....'],
                    ['.....',
                     '..O..',
                     '.OO..',
                     '.O...',
                     '.....']]

I_SHAPE_TEMPLATE = [['..O..',
                     '..O..',
                     '..O..',
                     '..O..',
                     '.....'],
                    ['.....',
                     '.....',
                     'OOOO.',
                     '.....',
                     '.....']]

O_SHAPE_TEMPLATE = [['.....',
                     '.....',
                     '.OO..',
                     '.OO..',
                     '.....']]

J_SHAPE_TEMPLATE = [['.....',
                     '.O...',
                     '.OOO.',
                     '.....',
                     '.....'],
                    ['.....',
                     '..OO.',
                     '..O..',
                     '..O..',
                     '.....'],
                    ['.....',
                     '.....',
                     '.OOO.',
                     '...O.',
                     '.....'],
                    ['.....',
                     '..O..',
                     '..O..',
                     '.OO..',
                     '.....']]

L_SHAPE_TEMPLATE = [['.....',
                     '...O.',
                     '.OOO.',
                     '.....',
                     '.....'],
                    ['.....',
                     '..O..',
                     '..O..',
                     '..OO.',
                     '.....'],
                    ['.....',
                     '.....',
                     '.OOO.',
                     '.O...',
                     '.....'],
                    ['.....',
                     '.OO..',
                     '..O..',
                     '..O..',
                     '.....']]

T_SHAPE_TEMPLATE = [['.....',
                     '..O..',
                     '.OOO.',
                     '.....',
                     '.....'],
                    ['.....',
                     '..O..',
                     '..OO.',
                     '..O..',
                     '.....'],
                    ['.....',
                     '.....',
                     '.OOO.',
                     '..O..',
                     '.....'],
                    ['.....',
                     '..O..',
                     '.OO..',
                     '..O..',
                     '.....']]
```

테트리스 블록이 가능한 모든 회전을 포함해서 어떻게 모양이 형성될 수 있는지 정의해놓았다. 문자열 목록을 만들고 내부 목록들은 블록의 단일 회전 모양을 나타내고 있다.

```
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}
```

딕셔너리를 통해 PIECES 변수에 테트리스 블럭 모양 템플릿을 넣어주었다.

```
def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT, BIGFONT, STARTIME
    pygame.init()
    STARTIME = time.time()
    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('2020079343 WOOJIYOUNG')

    showTextScreen('MY TETRIS')
```

메인 함수에서는 다양한 글로벌 상수를 좀 더 추가해주었고, 나는 과제의 play time 부분을 수행하기 위해 STARTIME 이라는 상수를 추가해주었다. 또한 과제 수행을 위해 pygame.display.set_caption, showTextScreen 부분을 수정해주었다.

```
while True: # game loop
    STARTIME = time.time()
    if random.randint(0, 2) == 0:
        pygame.mixer.music.load('Platform_9.mp3')
    elif random.randint(0, 2) == 1:
        pygame.mixer.music.load('Our_Lives_Past.mp3')
    else:
        pygame.mixer.music.load('Hover.mp3')

    pygame.mixer.music.play(-1, 0.0)
    runGame()
    pygame.mixer.music.stop()
    showTextScreen('Over :(')
```

게임 실행을 위해 계속 돌아가는 반복문이며, 나는 이 지점에 STARTIME을 넣어 게임을 재시작하는 시점에 시간이 초기화 될 수 있도록 했다. 과제를 수행하기 위해 조건문으로 음악이 랜덤으로 설정되며, 제공받은 음악을 세 개 넣어주었다. 게임이 종료되면 음악이 중지되고 게임 종료 화면을 띄우게 되어있는데 과제 수행을 위해 showTextScreen의 문구를 변경했다.

```
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()
```

새 게임이 시작되는 함수이다. 게임이 시작되고 테트리스 블록들이 떨어지기 전에 여러 변수를 초기화 시키고 맨 밑 두 줄에서 블록을 회전할 수 있는 현재의 피스로 변경시키고 다음 피스가 화면에 next 부분에 나타나는 피스로 설정되게 한다.

```
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()
```

이 루프는 테트리스 블록이 위에서 바닥으로 떨어질 때와 관련된 코드를 처리한다. 블록이 바닥에 떨어지면 없음으로 처리되어 새로운 피스가 떨어지고, 테트리스 게임 창이 블록으로 가득 찬 경우 false로 반환해 게임오버가 되게 한다.

```
for event in pygame.event.get(): # event handling loop
    if event.type == KEYUP:
```

이 루프는 테트리스 블록을 회전하거나 이동하거나 게임을 일시 중지할 때 처리된다.

```
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()
```

p키를 누르면 게임이 일시정지 되게 한다. 게임을 중단하고 블록 위치를 생각하는 경우를 예방하기 위해 창을 가리고 showTextScreen을 사용해 텍스트를 표시해준다. 이 부분에서

과제 수행을 위해 문구를 변경했다. 플레이어가 키를 누르면 텍스트는 반환되고 배경음악이 다시 재생되며 오랜 시간 정지를 했을 수도 있으니 여러 변수들을 다시 초기화 해준다.

```
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
```

화살표키(좌,우,아래)나 a,d,s 키를 사용해 블록을 움직일 수 있게 한다.

```
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()
```

왼쪽 화살표 키나 a를 눌렀을 때 피스의 값을 계산하고 데이터의 위치를 확인해 블록을 왼쪽으로 옮길 수 있게 한다. movingright를 false로 설정해 오른쪽으로는 움직이지 않게 한다. 마지막 줄에 저장된 0.15초가 지나면 다시 왼쪽으로 움직이게 되어있다.

```
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()
```

오른쪽 화살표 키나 d를 눌렀을 때 오른쪽으로 이동하는 것을 처리한다는 점을 제외하면 위의 설명과 거의 동일하다.

```
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']])
```

블럭의 회전과 관련된 부분이다. 1을 더하고 빼줌으로써 테트리스 블록이 정상적으로 회전할 수 있도록 해준다. 윗쪽 방향키 또는 w키를 사용해 회전시킬 수 있고, q키를 사용해 반대 방향으로 회전시킬 수 있도록 해준다.

```
elif (event.key == K_DOWN or event.key == K_s):
    movingDown = True
    if isValidPosition(board, fallingPiece, adjY=1):
        fallingPiece['y'] += 1
    lastMoveDownTime = time.time()
```

아래쪽 화살표나 s키를 누르면 블록이 아래로 떨어지게 하는데 플레이어가 평소보다 블록을 빠르게 떨어뜨리길 원할 때 키를 지속해서 누른 상태면 블록을 더 빠른 속도로 떨어지게 한다.

```
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
```

스페이스 키를 누르면 테트리스 블럭이 테트리스 게임 창의 가장 밑 부분으로 즉시 떨어지도록 하는 기능을 한다.

```
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()
```

플레이어가 화살표 키를 반복적으로 누르지 않아도 MOVESIDEWAYSFPEQ에 저장된 값인 0.15초 이상을 누르고 있을 경우 플레이어가 키를 놓기 전까지 자동으로 이동하게 해주는 기능을 한다.

```
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()
```

테트리스 블록이 자연스럽게 낙하할 수 있도록 해주는 코드이다. 블록이 떨어진 경우 블록을 테트리스 창에서 보이게하고 블록이 한 줄이 되어 지워진 경우에는 그에 맞는 점수를 올려주고 레벨이 변경되는지 여부도 확인한다. 블록이 다 낙하되지 않은 경우에는 낙하되게 두는 기능을 한다.

```
DISPLAYSURF.fill(BGCOLOR)
drawBoard(board)
drawStatus(score, level, time)
drawNextPiece(nextPiece)
if fallingPiece != None:
    drawPiece(fallingPiece)

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

위의 코드들을 실제 컴퓨터 화면에 나타나게 하는 역할을 한다. 과제 수행을 위해 drawStatus에 time 부분을 추가해주었다.

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

text, font, color 객체가 주어졌을 때 render()를 호출함으로써 필요할 때마다 코드를 입력하지 않고 쉽게 사용할 수 있게 도와준다.

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

종료시키는 기능이다

```
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
```

checkForQuit()을 호출하여 모든 QUIT 이벤트를 처리하고 프로그램이 있으면 종료한다. 그런 다음 이벤트 대기열에서 모든 keyup,keydown 이벤트를 꺼내고 모든 keydown 이벤트를 무시한다.

```
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." text.
    pressKeySurf, pressKeyRect = makeTextObjs('Press any 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()
```

위에 언급되던 showTextScreen 함수이다. 화면에 띄울 MY TETRIS, OVER :( 등의 텍스트에 폰트와 어두운 색상과, 밝은 색상을 적용하여 입체적으로 만들게 한다. 또한 타이틀 텍스트 밑에 추가적으로 문구를 달 수 있게 한다. 과제 수행을 위해 추가적인 문구를 여기에서 변경해주었다. 맨 밑에 와일문은 플레이어가 키를 누를 때까지 텍스트가 화면에 유지되게 하는 역할을 한다.

```
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
```

프로그램을 종료시키는 모든 이벤트를 처리하게 한다. 플레이어가 esc키를 누르면 언제든지 게임이 종료되게 한다.

```
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
```

점수가 1점씩 올라가다가 10점이 될 때마다 게임이 한 단계씩 올라가며, 한 단계 올라갈 때마다 블록이 점점 더 빨리 떨어지게 하는 기능을 한다.

```
def getNewPiece():
    # return a random new piece in a random rotation and color
    shape = random.choice(list(PIECES.keys()))

    if shape == 'S':
        a = 0
    elif shape == 'Z':
        a = 1
    elif shape == 'J':
        a = 2
    elif shape == 'L':
        a = 3
    elif shape == 'I':
        a = 4
    elif shape == 'O':
        a = 5
    elif shape == 'T':
        a = 6
    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': a}
    return newPiece
```

PIECES.keys()를 호출하여 가능한 모든 모양의 목록을 리스트에 넣고 그 중 랜덤으로 임의의 피스를 생성한다. 하단의 newPiece는 단순히 랜덤으로 생성된 shape, rotation, x, y, color를 가지고 있는 딕셔너리이다. 원래의 코드는 색깔을 랜덤하게 지정했지만 과제 수행을 위해 그 부분을 수정해 블럭의 모양마다 고유의 색깔을 가질 수 있도록 설정하였다.

```
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']
```

테트리스 블록의 데이터 구조를 가져와 테트리스 게임 판의 데이터 구조에 추가하여 블록이 착지한 후 화면 상에 기록되어 있도록 해주는 역할을 한다.

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

반복문을 통해 blank를 테트리스 판 모든 블록에 넣어 빈 테트리스 게임 판을 만드는 역할을 한다.

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

전달된 xy 좌표가 테트리스 게임 판에 존재하는 유효한 값을 나타내는지 확인하는 역할을 한다.

```
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
```

테트리스 블록이 테트리스 게임 판 안에 있는 블록과 겹치지 않으면 true를 반환해서 겹치는 블록이 없도록 만든다.

```
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
```

y매개 변수로 지정된 행에서 단순 검사를 수행한다. 테트리스 게임 판 안에 모든 공간이 블록으로 채워질 때 완료로 간주하는 역할을 하고, 비어있으면 false를 반환한다.

```
def removeCompleteLines(board):
    # Remove any completed lines on the board, move everythin
    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-
            # 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 th
            # This is so that if the line that was pulled dow
            # complete, it will be removed.
        else:
            y -= 1 # move on to check next row up
    return numLinesRemoved
```

테트리스 게임 판에서 블록으로 한 줄이 가득 차면 가득 찬 줄의 수 만큼 지우는 역할을 한다.

지워진 줄의 수를 반환하여 점수에 추가할 수 있도록 돕는다.

```
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))
```

테트리스 게임판의 블록 값을 픽셀 좌표로 변환시켜주는 기능을 한다.

```
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))
```

테트리스 게임판에 떨어질 블록과 다음번에 떨어질 블록을 그리는 기능을 한다. 또한 블록 간의 윤곽선을 그리기 위해 왼쪽과 위쪽 파라미터에 +1이 추가되고 너비 및 높이 파라미터에 -1이 추가된다. 또한 블록에 입체감을 주기 위해 어두운색으로 그려진 뒤, 밝은 색상으로 조금 더 작은 상자를 위에 그리는 기능을 한다.

```
def drawStatus(score, level, time):
    # 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)

    # draw the play time text
    timeSurf = BASICFONT.render('Play Time: %s' % round((time.time()-STARTIME))+' sec', True, TEXTCOLOR)
    timeRect = timeSurf.get_rect()
    timeRect.topleft = (WINDOWWIDTH - 600 , 20)
    DISPLAYSURF.blit(timeSurf, timeRect)
```

컴퓨터 화면에 보이는 창의 우측 모서리 상단에 나타나는 “Score:”, “Level:” 정보에 대한 텍스트를 렌더링 하는 역할을 하는 코드만 존재했지만 과제 수행을 위해 time에 관련된 코드를 작성해 좌측 모서리 상단에 나타나게 하는 기능을 하도록 했다.

```
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))
```

피스로 전달되는 피스 데이터 구조에 따라 피스의 상자를 그리는 역할을 한다.

```
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()
```

컴퓨터 화면에 보이는 창의 우측 모서리 상단에 다음 피스를 그리는 역할을 한다.

맨 밑의 두 줄은 모든 기능을 정의 한 후 메인함수를 호출하여 프로그램의 주요 부분을 시작하는 역할을 한다.

## 2. 함수의 호출 순서 또는 호출 조건

우선 main 함수가 호출되고, 메인함수 안에서 runGame함수가 호출된다.

그 다음 getBlankBorad 함수, calculateLevelAndFallFreq함수, getNewPiece함수를 통해 초기화가 진행된다.

drawBoard(-> drawbox함수 호출됨(->converToPixelCoords함수 호출됨)), drawStatus,

,drawNextPiece

(->drawPiece함수 호출됨(->convertToPixelCoords함수+drawBox함수 호출됨)) 함수들이 게임의 초기 화면을 구축해주고 getNewPiece 함수를 통해 게임이 진행되다가 플레이어가 p키를 클릭하면 게임이 중단되고

showTextScreen

(->makeTextObjs함수 호출됨/checkForKeyPress함수 호출됨(->checkForQuit함수 호출됨(->terminate함수 호출됨)))함수가 호출된다.

플레이어가 키를 눌러 다시 게임을 시작하고 여러 키를 사용하여 게임을 하면 isValidPosition함수(-> isOnBoard함수 호출됨)가 반복적으로 호출되며 블록이 겹치지 않게 돕는 역할을 맡는다.

겹치지 않게 된 블럭들을 addToBoard함수가 테트리스 보드 판 위에 보이게 기록을 한다. 그렇게 게임이 진행되다가 한 줄이 블럭으로 가득 차게 되면

removeCompleteLine(-> isCompleteLine함수 호출됨)함수가 호출되어 지워진 줄 만큼 점수를 올리는 역할을 하게 된다.

그렇게 점수가 올라가다가 10점이 되면 calculateLevelAndFallFreq함수가 호출되어 레벨을 1씩 올라가게 하고, 블록이 떨어지는 속도를 빨라지게 한다.

또한 게임 도중 블럭이 끝까지 쌓이거나 플레이어가 esc키를 누르는 경우 checkForQuit함수가 호출(->terminate함수 호출)되어 게임이 종료된다.