과제-3 **(Tetris) 블록 회전과 충돌 처리**

① 수행 과업

- 테트리스 블록의 회전과 충돌 처리 구현.
- 블록이 게임 보드 경계와 이미 쌓인 블록과 충돌하지 않도록 로직 추가.

② 요구사항

1. 블록 회전

- 블록을 90도 회전시키는 기능 구현.
- 기능 요구사항:
  - UP 키를 누르면 현재 블록이 회전.
  - 회전 시 경계를 벗어나지 않도록 처리.

2. 충돌 감지

- 현재 블록이 다른 블록이나 바닥과 충돌하면 멈추고 새로운 블록 생성.
- 기능 요구사항
  - 충돌 발생 시 블록을 고정하고 새로운 블록을 화면 상단에 생성.

③ 스스로 학습

1. 블록 회전

- 블록 회전이란?
  - 블록을 90도 회전시키는 기능으로, 블록의 모양이 바뀌도록 배열의 행과 열을 변환.
- 회전 구현 방법:
  - 블록을 2D 배열로 표현한 경우, 배열을 전치(transpose)하고 행을 뒤집어 회전을 구현.
- 구현 예시:

원래 블록	90도 회전:
[[1, 0], [1, 1]]	[[1, 1], [1, 0]]

```
def rotate_block(block):
    return [list(row) for row in zip(*block[::-1])]
```

2. 충돌 처리

- 충돌 처리란?
  - 블록이 게임 보드의 경계나 다른 블록과 겹치지 않도록 제한하는 로직.
- 충돌 조건:
  - 블록의 일부가 보드의 왼쪽, 오른쪽, 아래쪽 경계를 넘는 경우.
  - 블록의 일부가 이미 보드에 고정된 블록과 겹치는 경우.
- 충돌 감지 구현 아이디어
  - 게임은 반복적으로 화면을 업데이트하며 동작.
  - 블록의 각 셀(1)을 게임 보드의 좌표로 변환하여 겹치는지 확인

3. 스스로 학습을 위한 질문

- 블록 회전
  - zip()과 리스트 내포를 사용해 배열을 회전시키는 방식은 무엇인가?
  - 블록 회전 후 경계를 벗어나지 않도록 처리하려면 어떻게 해야 할까?
- 충돌 처리
  - 블록의 좌표를 게임 보드와 비교하여 충돌을 감지하려면 어떤 알고리즘이 필요할까?
  - 충돌 발생 시, 회전을 취소하거나 블록을 제자리에 두려면 어떻게 처리할 수 있을까?

④ 코드 구현 예제를 이해하여 설명하시오.

```
import pygame
import random

# Pygame 초기화
pygame.init()

# 화면 크기 설정
WIDTH, HEIGHT = 300, 600
ROWS, COLS = 20, 10
BLOCK_SIZE = WIDTH // COLS

# 색상 정의
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
BLUE = (0, 0, 255)
RED = (255, 0, 0)

# 블록 모양 정의 (I, O, T, S, Z, L, J)
SHAPES = [
    [[1, 1, 1, 1]],  # I
    [[1, 1], [1, 1]],  # O
    [[0, 1, 0], [1, 1, 1]],  # T
    [[0, 1, 1], [1, 1, 0]],  # S
    [[1, 1, 0], [0, 1, 1]],  # Z
    [[1, 0, 0], [1, 1, 1]],  # L
    [[0, 0, 1], [1, 1, 1]],  # J
]

# 게임 보드 초기화
def create_board():
    return [[0 for _ in range(COLS)] for _ in range(ROWS)]

# 블록 회전 함수
def rotate_block(block):
    return [list(row) for row in zip(*block[::-1])]

# 블록 그리기 함수
def draw_block(screen, shape, x, y):
    for row_idx, row in enumerate(shape):
        for col_idx, cell in enumerate(row):
            if cell:
                pygame.draw.rect(
                    screen,
                    BLUE,
                    (x + col_idx * BLOCK_SIZE, y + row_idx * BLOCK_SIZE, BLOCK_SIZE,
BLOCK_SIZE),
                )

# 보드 그리기 함수
```

```
def draw_board(screen, board):
    for row_idx, row in enumerate(board):
        for col_idx, cell in enumerate(row):
            if cell:
                pygame.draw.rect(
                    screen,
                    RED,
                    (col_idx * BLOCK_SIZE, row_idx * BLOCK_SIZE, BLOCK_SIZE,
BLOCK_SIZE),
                )

# 충돌 감지 함수
def check_collision(board, shape, x, y):
    for row_idx, row in enumerate(shape):
        for col_idx, cell in enumerate(row):
            if cell:
                board_x = (x // BLOCK_SIZE) + col_idx
                board_y = (y // BLOCK_SIZE) + row_idx

                if board_x < 0 or board_x >= COLS or board_y >= ROWS:
                    return True
                if board_y >= 0 and board[board_y][board_x] != 0:
                    return True
    return False

# 블록 고정 함수
def place_block(board, shape, x, y):
    for row_idx, row in enumerate(shape):
        for col_idx, cell in enumerate(row):
            if cell:
                board_x = (x // BLOCK_SIZE) + col_idx
                board_y = (y // BLOCK_SIZE) + row_idx
                if 0 <= board_x < COLS and 0 <= board_y < ROWS:
                    board[board_y][board_x] = 1

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Tetris - 블록 회전과 충돌 처리")

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

    current_block = random.choice(SHAPES)
    block_x, block_y = 4 * BLOCK_SIZE, 0

    while running:
        screen.fill(BLACK)

        # 이벤트 처리
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:  # 블록 회전
                    rotated_block = rotate_block(current_block)
                    if not check_collision(board, rotated_block, block_x, block_y):
                        current_block = rotated_block

        # 키 입력 처리
        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            if not check_collision(board, current_block, block_x - BLOCK_SIZE, block_y):
                block_x -= BLOCK_SIZE
        if keys[pygame.K_RIGHT]:
            if not check_collision(board, current_block, block_x + BLOCK_SIZE, block_y):
                block_x += BLOCK_SIZE
        if keys[pygame.K_DOWN]:
            if not check_collision(board, current_block, block_x, block_y + BLOCK_SIZE):
                block_y += BLOCK_SIZE

        # 블록 자동 하강
        block_y += BLOCK_SIZE if not check_collision(board, current_block, block_x,
block_y + BLOCK_SIZE) else 0

        # 충돌 시 블록 고정 및 새로운 블록 생성
        if check_collision(board, current_block, block_x, block_y + BLOCK_SIZE):
            place_block(board, current_block, block_x, block_y)
            current_block = random.choice(SHAPES)
            block_x, block_y = 4 * BLOCK_SIZE, 0

        # 보드 및 블록 그리기
        draw_board(screen, board)
        draw_block(screen, current_block, block_x, block_y)

        pygame.display.flip()
        clock.tick(10)

    pygame.quit()

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

【Peer 평가 Check List】(Tetris) 블록 회전과 충돌 처리

설명자 Explainer		평가자 Evaluator		날짜 date	
☐ 전혀 설명하지 못함 ☐ 기본적인 설명은 했으나 부족함 ☐ 완벽하게 이해하고 명확히 설명함					

【기능 이해 및 코드 해석 능력】

1. 블록 회전 구현 이해

☐ 블록의 회전 방식(rotate_block 함수)이 어떻게 작동하는지 명확히 설명했는가?
- 예: zip(*block[::-1])이 배열의 행과 열을 변환하는 원리를 이해했는가?

☐ 회전 후 블록의 좌표와 크기가 게임 보드 내에 유지되도록 처리한 방식(경계 조건)을 설명했는가?

2. 충돌 감지 이해

☐ 블록이 게임 보드의 경계(왼쪽, 오른쪽, 아래쪽)를 벗어났는지 확인하는 로직을 정확히 설명했는가?

☐ 블록이 고정된 다른 블록과 충돌했는지 확인하는 방식(check_collision 함수)을 이해하고 설명했는가?

☐ 충돌 시 블록이 멈추고 새로운 블록이 생성되는 흐름을 명확히 설명했는가?

3. 블록 고정 로직 이해

☐ 충돌 발생 시 블록을 게임 보드에 "고정"시키는 로직(place_block 함수)을 정확히 설명했는가?

☐ 블록이 고정된 후 새로운 블록이 생성되는 과정을 이해했는가?

【설명 능력】

4. 명확성과 논리성

☐ 동료가 이해하기 쉽도록 프로그램의 주요 기능과 동작 원리를 논리적으로 설명했는가?

⑤ [도전과제] (Tetris) 라인 삭제와 블록 속도 증가

- 요구사항:

1. 라인 삭제 구현
   - 게임 보드에서 한 줄이 완전히 채워진 경우 해당 줄 삭제.
   - 삭제된 라인 위의 블록은 모두 한 칸씩 아래로 내려오도록 구현.

【Hint 1: 라인 삭제】

- 행의 모든 셀이 채워졌는지 확인
  - 보드의 각 행(row)을 반복문으로 검사하여 모든 셀이 1인 경우 해당 행을 삭제.
- 삭제 후 행 이동:
  - 삭제된 행 위의 모든 블록을 한 칸 아래로 이동.
- 구현 예시:

```
def clear_lines(board):
    cleared_rows = 0
    new_board = []
    for row in board:
        if all(cell == 1 for cell in row):  # 해당 행이 꽉 찬 경우
            cleared_rows += 1
        else:
            new_board.append(row)

    # 위쪽에 빈 행 추가
    for _ in range(cleared_rows):
        new_board.insert(0, [0] * COLS)
    return new_board, cleared_rows
```

2. 점수 시스템 추가
   - 삭제된 라인의 개수에 따라 점수를 부여:
     - 1줄 삭제: +100점
     - 2줄 삭제: +300점
     - 3줄 삭제: +500점
     - 4줄 삭제: +800점
   - 화면에 현재 점수 표시.

【Hint 2: 점수 시스템】

- 점수는 삭제된 라인의 개수에 따라 결정.
  - 블록이 일정 시간마다 자동으로 아래로 내려와야 하지만, 게임 루프는 초당 여러 번 실행됨.
- 점수 규칙:
  - 1줄 삭제: +100점
  - 2줄 삭제: +300점
  - 3줄 삭제: +500점

- 4줄 삭제: +800점

- 구현 예시:

```
def calculate_score(cleared_rows):
    score_table = {1: 100, 2: 300, 3: 500, 4: 800}
    return score_table.get(cleared_rows, 0)
```

3. 블록 속도 증가
- 삭제된 줄의 개수(점수)에 따라 블록이 내려오는 속도 점진적으로 증가.
- 게임이 진행될수록 난이도 상승.

【Hint 3: 블록 속도 증가】
- 속도 증가 방식:
  - 점수 또는 삭제된 줄의 누적 개수에 따라 속도를 증가.
- 구현 아이디어:
  - 삭제된 줄의 개수를 기준으로 속도 조정
- 구현 예시:

```
speed = max(50, 500 - (cleared_rows * 20))  # 최소 50ms까지 제한
```

【Peer 평가 Check List】(Tetris) 블록 회전과 충돌 처리

설명자 Explainer		평가자 Evaluator		날짜 date	
☐ 전혀 설명하지 못함 ☐ 기본적인 설명은 했으나 부족함 ☐ 완벽하게 이해하고 명확히 설명함					

【기능 구현 확인】

1. **타임아웃 처리**
   - ☐ HTTP 요청 타임아웃 설정이 올바르게 구현되었는가?
   - ☐ 타임아웃 시 적절한 오류 메시지가 출력되는가?