과제-3 **(Tetris) 기본 구조 설계**

① 수행 과업

- 테트리스의 기본 게임 구조와 로직 이해.
- Pygame을 사용하여 게임 창 생성, 기본 블록 출력.

② 요구사항

1. 테트리스 게임 창 생성

- Pygame을 사용하여 10x20 크기의 테트리스 게임 보드 생성.
- 기능 요구사항:
  - 게임 창 표시 및 창 닫기 이벤트 처리.
  - 10x20 그리드 구현(블록의 위치를 표현).

2. 테트리스 블록 설계

- 테트리스 블록(테트로미노)을 데이터로 정의하고 게임 보드에 표시.
- 기능 요구사항:
  - 각 테트로미노를 list로 정의.
  - 랜덤한 블록이 화면 상단에 나타나도록 구현.

③ 스스로 학습

1. Pygame 기초

1) Pygame의 역할

- Pygame이란?
  - 파이썬을 사용해 간단한 2D 게임과 멀티미디어 프로그램을 개발할 수 있는 라이브러리.
- 활용 사례:
  - 화면 생성, 이벤트 처리, 이미지 및 사운드 출력 등.

2) 주요 개념

- Pygame 초기화:
  - pygame.init(): Pygame 라이브러리를 초기화하여 사용 가능 상태로 만듦.
- 게임 화면 설정:
  - pygame.display.set_mode(): 게임 창 크기를 설정.
  - pygame.display.set_caption(): 게임 창의 제목을 설정.
- 게임 루프:
  - 게임은 반복적으로 화면을 업데이트하며 동작.
  - while 루프를 사용해 게임 상태를 지속적으로 처리.

2. Pygame 이벤트 처리

1) 이벤트란?

- 사용자의 입력(키보드, 마우스 등)에 반응하여 특정 동작을 수행하는 기능.
- 예: 키보드로 블록을 이동, ESC 키로 게임 종료.

2) 주요 이벤트

- pygame.event.get(): 발생한 이벤트 목록을 가져옴.
- 이벤트 종류:
  - 종료 이벤트: pygame.QUIT
  - 키 입력 이벤트: pygame.KEYDOWN (키를 누를 때), pygame.KEYUP (키를 뗄 때)

3) 키 입력 처리

- 키 코드: pygame.K_LEFT, pygame.K_RIGHT, pygame.K_DOWN
- 키 상태 확인: pygame.key.get_pressed()를 사용해 특정 키가 눌려 있는지 확인.

3. 화면 좌표와 블록의 위치 계산

1) 화면 좌표계

- 화면의 (0, 0)은 왼쪽 위 모서리를 의미.
- x 좌표는 오른쪽으로 갈수록 증가, y 좌표는 아래로 갈수록 증가.

2) 블록 위치 계산

- 블록의 위치는 그리드 좌표를 기반으로 계산.
  - 각 블록의 위치 = (그리드 열 × 블록 크기, 그리드 행 × 블록 크기)
- 코드에서의 예:

```
pygame.draw.rect(screen, BLUE, (x, y, BLOCK_SIZE, BLOCK_SIZE))
```

4. 테트리스 블록 설계와 데이터 구조

1) 블록 정의 방식

- 테트리스 블록은 2D 배열로 정의:
  - 1: 블록이 있는 부분.
  - 0: 빈 부분.
- 예: T 모양 블록

```
[[0, 1, 0],
 [1, 1, 1]]
```

2) 블록 표시 방법

- 블록의 각 셀에 대해 화면에 사각형(rect)을 그려 표시.
- 반복문 사용: 배열의 행과 열을 반복하며 1인 부분만 그리기.

5. 스스로 학습을 위한 질문

1) Pygame 관련

- Pygame 초기화와 화면 설정은 어떻게 이루어질까?
- pygame.event.get()와 pygame.key.get_pressed()의 차이점은 무엇일까?

2) 블록 그리기

- 블록의 위치를 그리드 좌표와 화면 좌표로 변환하는 방법은 무엇일까?
- draw_block 함수에서 배열의 각 셀을 화면에 출력하려면 어떻게 해야 할까?

3) 테트리스 블록 설계

- 블록의 회전은 2D 배열에서 어떻게 구현할 수 있을까?
- 게임 보드와 블록 간의 충돌을 어떻게 감지할 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)

# 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 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),
                )

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Tetris")

    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)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # 블록 이동 로직
        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            block_x -= BLOCK_SIZE
        if keys[pygame.K_RIGHT]:
            block_x += BLOCK_SIZE
        if keys[pygame.K_DOWN]:
            block_y += BLOCK_SIZE

        # 게임 보드와 블록 그리기
        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) 기본 구조 설계

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

【기능 이해 및 코드 해석 능력】

1. Pygame 초기화 및 화면 설정

☐ Pygame을 초기화하고, 게임 화면(WIDTH, HEIGHT)과 그리드(ROWS, COLS)를 설정하는 부분을 이해했는가?

☐ 블록 크기(BLOCK_SIZE)와 그리드의 역할을 명확히 설명했는가?

2. 게임 보드 초기화 (create_board)

☐ acreate_board 함수가 게임 보드를 초기화하는 로직을 정확히 설명했는가?

☐ 2D 리스트를 활용하여 보드를 빈 상태로 만드는 이유를 이해했는가?

3. 블록 설계와 데이터 구조

☐ SHAPES 리스트를 사용하여 테트리스 블록(테트로미노)을 정의한 방식을 설명했는가?

☐ 각 블록이 배열로 표현되는 이유와 배열 구조를 명확히 이해했는가?

4. 블록 그리기 (draw_block)

☐ draw_block 함수가 특정 위치에 블록을 그리는 과정을 이해하고 설명했는가?

☐ 블록의 위치(x, y)와 크기(BLOCK_SIZE)가 화면에서 어떻게 결정되는지 설명했는가?

5. 게임 루프와 이벤트 처리

☐ Pygame의 이벤트 루프(for event in pygame.event.get():)를 이해하고, 종료 이벤트(QUIT) 처리 방식을 설명했는가?

☐ 키 입력(K_LEFT, K_RIGHT, K_DOWN)에 따라 블록의 움직임을 처리하는 로직을 명확히 설명했는가?

【설명 능력】

6. 명확성과 논리성

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

⑤ [도전과제] (Tetris) 기본 구조 설계

- 요구사항:

1. 블록 이동 제약 추가:
   - 블록이 화면 경계를 넘어가지 않도록 제한.
   - 좌우 이동(K_LEFT, K_RIGHT) 시 게임 보드의 크기를 초과하지 않게 처리.

【Hint 1: 블록의 위치가 경계를 넘어가지 않도록 제한】

- 가로 경계 조건:
  - 왼쪽 경계: block_x >= 0
  - 오른쪽 경계: block_x + 블록 너비 <= 화면 너비
- 구현 아이디어:
  - 블록이 움직일 때(K_LEFT, K_RIGHT 입력), 이동 후의 좌표를 미리 계산하여 화면 경계를 초과하지 않도록 체크.
- 구현 예시:

```
if keys[pygame.K_LEFT]:
    if block_x > 0:  # 왼쪽 경계를 벗어나지 않도록
        block_x -= BLOCK_SIZE
if keys[pygame.K_RIGHT]:
    if block_x + len(current_block[0]) * BLOCK_SIZE < WIDTH:  # 오른쪽 경계를 벗어나지 않도록
        block_x += BLOCK_SIZE
```

2. 블록 자동 하강:
   - 블록이 일정 시간마다 한 칸씩 아래로 내려오도록 구현.
   - 키보드 입력 없이도 블록이 움직이는 동작을 구현.

【Hint 2: 시간 기반 동작 (자동 하강)】

- 문제 상황
  - 블록이 일정 시간마다 자동으로 아래로 내려와야 하지만, 게임 루프는 초당 여러 번 실행됨.
- 해결 방법:
  - pygame.time.Clock()과 clock.get_rawtime()을 사용하여 일정 시간 간격(예: 500ms)마다 블록을 하강.
- 구현 예시:

```
drop_time += clock.get_rawtime()
if drop_time > 500:  # 500ms마다 블록이 한 칸 하강
    block_y += BLOCK_SIZE
    drop_time = 0
```

3. 블록 바닥 충돌 감지:

- 블록이 보드의 바닥에 닿으면 멈추고 새로운 블록을 생성.

【Hint 3: 게임 보드와 블록의 충돌 체크】

- 충돌 조건:
  - 블록이 바닥에 닿거나, 이미 고정된 블록에 닿는 경우 충돌로 간주.
- 구현 아이디어:
  - 블록의 각 셀이 이동 후 게임 보드의 경계나 이미 고정된 셀과 겹치는지 확인.

- 구현 예시:

```
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
                # 경계 밖으로 나가거나, 이미 고정된 블록이 있는 경우 충돌
                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
```

【Hint 4: 블록 고정】

- 아이디어
  - 충돌이 감지되면 현재 블록을 보드에 "고정"하고 새로운 블록을 생성.
  - board를 2D 리스트로 사용하여 고정된 블록 정보를 저장.

- 구현 예시:

```
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
```

【Hint 5: 새로운 블록 생성】

- 새로운 블록을 생성하려면
  - 충돌이 감지되면 place_block()을 호출해 현재 블록을 고정.
  - 새로운 블록을 랜덤하게 생성.

- 구현 예시:

```
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)  # 새로운 블록 생성
    block_x, block_y = 4 * BLOCK_SIZE, 0  # 초기 위치로 이동
```