Object-Oriented Game Development: Tank Crossing

Introduction

이 과제는 주어진 게임 시나리오를 바탕으로, 객체지향 프로그래밍(OOP) 원칙을 적용하여 **Pygame** 기반의 탱크 게임을 완성하는 것입니다. 이 과정을 통해 상속 등 클래스 간의 상호작용을 이해하고, 실제 문제 해결에 OOP 개념을 적용하는 능력을 기를 수 있습니다.

게임 시나리오

이 게임의 목표는 장애물을 피하거나 파괴하면서 탱크를 최상단까지 가장 빠르게 도달시키는 것입니다.

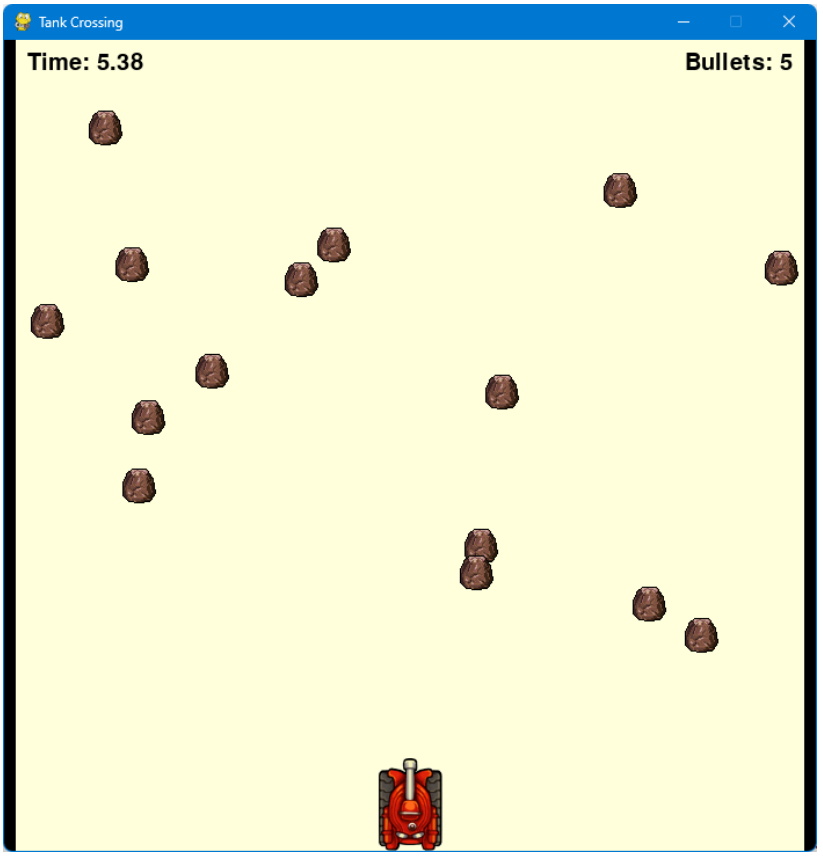
- 플레이어와 조작
  - 플레이어는 화면 하단 중앙에서 출발하는 탱크 한 대를 조종합니다.
  - 화살표 키**(←, →, ↑)**를 사용하여 탱크를 이동시킬 수 있습니다. 후진(↓)은 불가능합니다.
  - 스페이스바를 눌러 포탄을 발사할 수 있습니다.
- 규칙 및 환경
  - 장애물 **(**돌**)**: 게임이 시작되면 15개의 돌이 서로 충돌하지 않는 임의의 위치에 생성됩니다. 각 돌은 좌우로 계속 움직이며, 양쪽 벽에 닿으면 방향을 바꿉니다.
  - 충돌 **(**탱크와 돌**)**: 만약 탱크가 돌과 충돌하면, 탱크는 즉시 처음 시작 위치로 되돌아갑니다.
  - 포탄 발사:
    - 포탄으로 돌을 맞추면, 포탄과 돌은 함께 화면에서 사라집니다.
    - 돌이 사라질 때, **50%**의 확률로 새로운 돌 1개가 임의의 위치에 다시 생성됩니다. 돌은 기존 돌과 충돌하지 않는 위치에 생성되지만 탱크에 충돌하는 위치에 생성될 수도 있습니다.
    - 포탄은 한 번 발사하면 **500ms(0.5**초**)**의 재장전 시간이 지나야 다시 발사할 수 있습니다.
    - 한 게임당 사용할 수 있는 포탄은 총 **5**개로 제한됩니다.
    - 포탄의 생성 위치는 아래 그림처럼 탱크의 중앙(x방향), 앞쪽(y방향)입니다. 탱크의 맨 위와 포탄의 아래 위치가 맞닿습니다.

  - **UI**: 화면 왼쪽 위에는 경과 시간이, 오른쪽 위에는 남은 포탄 수가 표시됩니다.
- 승리 조건

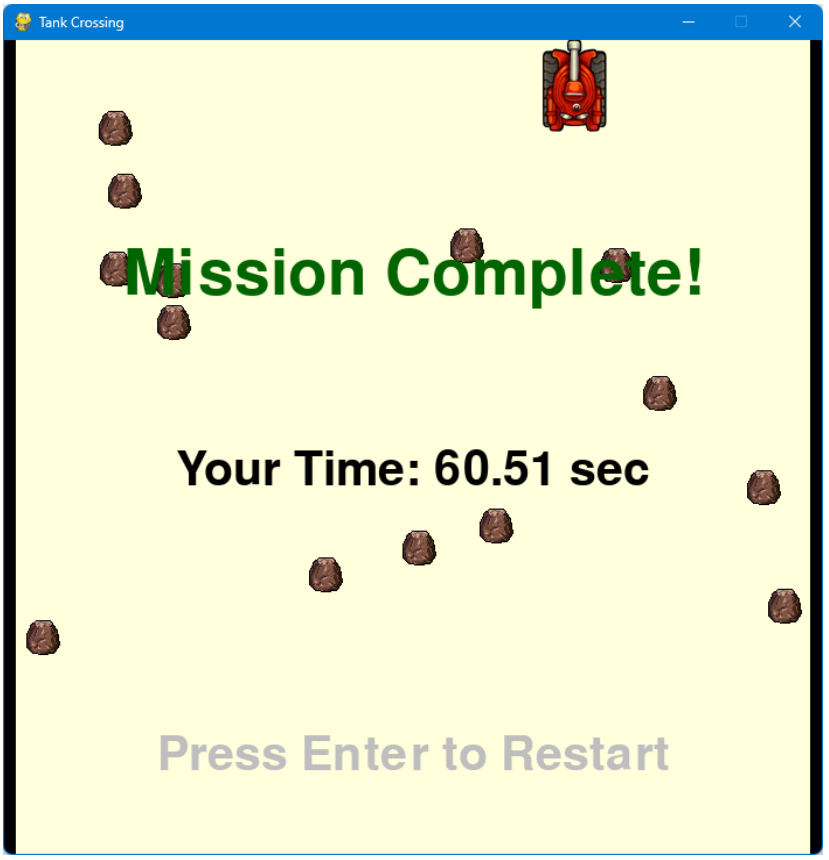
- 탱크가 화면 최상단에 도달하면 게임이 종료되고, **'Mission Complete!'** 메시지와 함께 최종 기록(도달 시간)이 화면에 나타납니다.
- 게임 종료 후 **Enter** 키를 누르면 언제든지 게임을 다시 시작할 수 있습니다.

Test Output

[초기 화면: 시작 후 3.16초 경과한 상황]



[종료 화면: 탱크가 화면 최상단에 도달한 후 상황]



주요 클래스 및 기능

1. Game

게임의 전체 흐름을 총괄하는 메인 클래스입니다. 게임 창을 생성하고, 메인 루프를 실행하며, 모든 게임 객체(스프라이트 그룹)를 관리합니다. 이벤트 처리, 상태 업데이트, 화면 그리기 등 게임의 핵심 로직을 담당합니다.

- 주요 관계:
    - `Player`, `Stone`, `Bullet`, `Wall` 객체 인스턴스를 생성하고 관리합니다 (Composition 관계).
    - 모든 게임 객체를 `all_sprites`라는 스프라이트 그룹에 담아 한 번에 업데이트하고 그립니다.

2. Player

플레이어가 조종하는 탱크 객체를 생성하기 위한 클래스입니다.

- 주요 기능:
    - 키보드 입력에 따라 좌, 우, 위로 이동합니다.
    - `shoot()` 메서드를 통해 `Bullet` 객체를 생성합니다.

3. Stone

장애물인 돌 객체를 생성하기 위한 클래스입니다.

- 주요 기능:
    - 생성 시 무작위 위치와 속도를 가집니다.
    - 좌우로 움직이다가 `Wall`에 닿으면 방향을 바꿉니다.

4. Bullet

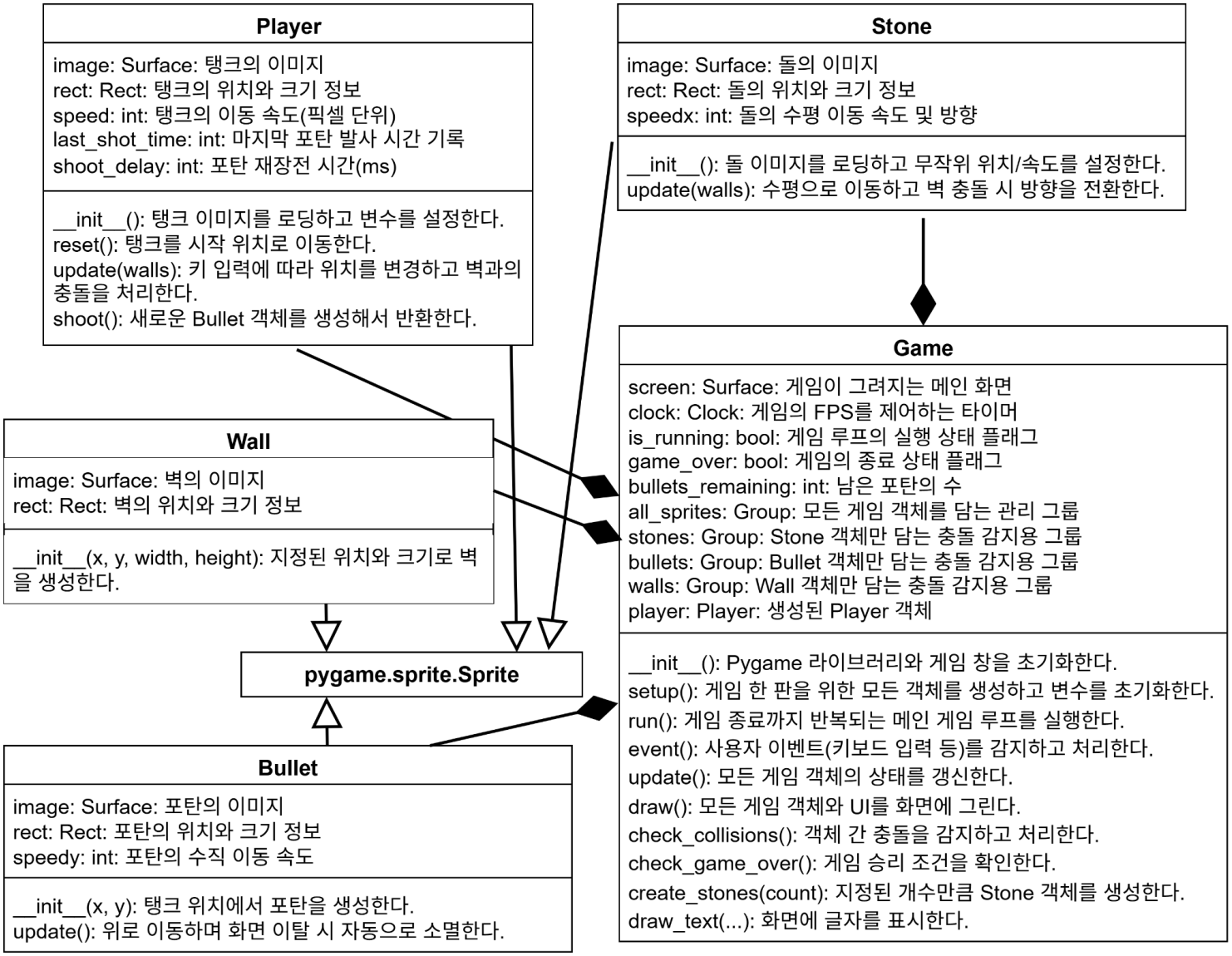
플레이어가 발사하는 포탄 객체를 생성하기 위한 클래스입니다.

- 주요 기능:
    - `Player`에 의해 생성됩니다.
    - 위쪽으로 직선으로 날아가며, 화면을 벗어나면 자동으로 제거됩니다.

5. Wall

게임 화면 양쪽에 위치하여 탱크와 돌이 밖으로 나가지 못하게 막는 벽 객체를 생성하기 위한 클래스입니다. 별도의 업데이트 로직은 없으며, 다른 객체들의 충돌 감지 기준으로 사용됩니다.

UML 다이어그램


Player
image: Surface: 탱크의 이미지
rect: Rect: 탱크의 위치와 크기 정보
speed: int: 탱크의 이동 속도(픽셀 단위)
last_shot_time: int: 마지막 포탄 발사 시간 기록
shoot_delay: int: 포탄 재장전 시간(ms)
__init__(): 탱크 이미지를 로딩하고 변수를 설정한다.
reset(): 탱크를 시작 위치로 이동한다.
update(walls): 키 입력에 따라 위치를 변경하고 벽과의 충돌을 처리한다.
shoot(): 새로운 Bullet 객체를 생성해서 반환한다.
Stone
image: Surface: 돌의 이미지
rect: Rect: 돌의 위치와 크기 정보
speedx: int: 돌의 수평 이동 속도 및 방향
__init__(): 돌 이미지를 로딩하고 무작위 위치/속도를 설정한다.
update(walls): 수평으로 이동하고 벽 충돌 시 방향을 전환한다.
Game
screen: Surface: 게임이 그려지는 메인 화면
clock: Clock: 게임의 FPS를 제어하는 타이머
is_running: bool: 게임 루프의 실행 상태 플래그
game_over: bool: 게임의 종료 상태 플래그
bullets_remaining: int: 남은 포탄의 수
all_sprites: Group: 모든 게임 객체를 담는 관리 그룹
stones: Group: Stone 객체만 담는 충돌 감지용 그룹
bullets: Group: Bullet 객체만 담는 충돌 감지용 그룹
walls: Group: Wall 객체만 담는 충돌 감지용 그룹
player: Player: 생성된 Player 객체
__init__(): Pygame 라이브러리와 게임 창을 초기화한다.
setup(): 게임 한 판을 위한 모든 객체를 생성하고 변수를 초기화한다.
run(): 게임 종료까지 반복되는 메인 게임 루프를 실행한다.
event(): 사용자 이벤트(키보드 입력 등)를 감지하고 처리한다.
update(): 모든 게임 객체의 상태를 갱신한다.
draw(): 모든 게임 객체와 UI를 화면에 그린다.
check_collisions(): 객체 간 충돌을 감지하고 처리한다.
check_game_over(): 게임 승리 조건을 확인한다.
create_stones(count): 지정된 개수만큼 Stone 객체를 생성한다.
draw_text(...): 화면에 글자를 표시한다.
Wall
image: Surface: 벽의 이미지
rect: Rect: 벽의 위치와 크기 정보
__init__(x, y, width, height): 지정된 위치와 크기로 벽을 생성한다.
pygame.sprite.Sprite
Bullet
image: Surface: 포탄의 이미지
rect: Rect: 포탄의 위치와 크기 정보
speedy: int: 포탄의 수직 이동 속도
__init__(x, y): 탱크 위치에서 포탄을 생성한다.
update(): 위로 이동하며 화면 이탈 시 자동으로 소멸한다.

Problem 1: `Player.shoot` 메서드 - 재장전 시간을 고려한 포탄 생성

게임 시나리오: "플레이어가 스페이스키를 누르더라도 이전 포탄이 발사된 이후 500ms가 지나야만 발사되며, 그 이전에 누른 스페이스키는 무시된다."

`Player` 클래스의 `shoot` 메서드는 포탄 발사를 담당합니다. 시나리오에 따라, 포탄은 연속해서 발사할 수 없으며 일정한 재장전 시간이 필요합니다. 이 시간 제약 조건을 확인하여 포탄 `Bullet` 객체를 생성하는 로직을 완성하세요.

- 구현 내용:
  - 마지막 발사 시간과 현재 시간을 비교하여 500ms가 지났는지 확인해야 합니다.
  - 발사가 가능하다면, 새로운 `Bullet` 객체를 생성하여 반환하고, 다음 발사를 위해 현재 시간을 기록해야 합니다.
  - Bullet 객체의 생성 위치는 탱크의 중앙(x방향), 앞쪽(y방향)입니다. 탱크의 맨 위와 포탄의 아래 위치가 맞닿습니다. (Bullet 클래스의 init을 먼저 확인하세요. 생성 시 인자로 받은 x와 y를 포탄의 중앙(x방향), 맨아래(y방향) 값으로 설정합니다.)

  - 이 메서드에서는 Bullet 객체를 생성해서 반환하는 것까지만 담당하며, 각 그룹에 추가하는 등의 작업은 메서드를 호출한 곳에서 처리합니다.
  - 발사가 불가능하다면, 아무 일도 일어나지 않아야 합니다.

Problem 2: Stone.update 메서드 **-** 벽에 닿으면 튕겨 나오는 돌

게임 시나리오: "각 돌은 일정한 속도로 왼쪽 또는 오른쪽으로 이동한다. 왼쪽과 오른쪽 끝에 탱크와 돌이 넘어갈 수 없는 벽이 생성된다."

`Stone` 클래스의 `update` 메서드는 돌의 움직임을 책임집니다. 이 메서드를 완성하여, 돌이 자신의 이동 속도에 따라 계속 움직이다가 화면 양쪽의 벽에 부딪혔을 때, 이동 방향을 반대로 바꾸도록 구현하세요.

- 구현 내용:
    - 돌은 매 프레임마다 자신의 속도(`speedx`)만큼 수평으로 이동해야 합니다.
    - 돌이 벽(`walls` 그룹)과 충돌하는지 감지해야 합니다.
    - 충돌이 감지되면, 돌의 수평 이동 방향이 반대로 바뀌어야 합니다.

Problem 3: Game.create_stones 메서드 **-** 서로 겹치지 않는 장애물 배치

게임 시나리오**:** "돌은 ... 랜덤하게 15개가 겹치지 않게 배치되며..."

Game 클래스의 create_stones 메서드는 게임 시작 시 돌을 생성하거나 게임 진행 중 돌이 포탄에 맞았을 때 50%의 확률로 다시 돌을 생성시킬 때 사용합니다. 시나리오의 요구사항에 맞게, 지정된 개수(count)만큼 돌을 생성하되, 생성되는 돌들이 서로 겹치지 않도록 하는 로직을 완성하세요.

- 구현 내용**:**
    - 요청된 개수(count)만큼 돌을 생성해야 합니다.
    - 새로운 돌을 생성할 때마다, 이미 존재하는 다른 돌들과 충돌하는지 확인해야 합니다.
    - 만약 위치가 겹친다면, 겹치지 않는 새 위치를 찾을 때까지 이 과정을 반복해야 합니다.
    - 생성된 돌은 돌 그룹과 모든 스프라이트 그룹에 추가해야 합니다.

Problem 4: Game.event 메서드 **-** 사용자 입력**(**발사 및 재시작**)** 처리

게임 시나리오**:** "플레이어는 돌을 부수기 위해 스페이스키를 눌러 포탄을 발사할 수 있다." 또한, "게임이 종료된 후에는 Enter 키를 눌러 게임을 다시 시작할 수 있다."

`Game` 클래스의 `event` 메서드는 플레이어의 키보드 입력을 감지하고 그에 맞는 행동을 실행하는 역할을 합니다. 이 메서드를 완성하여, 게임 상태**(`self.game_over`)**에 따라 두 가지 다른 키 입력(`Space`, `Return`)을 처리하도록 구현하세요.

- 구현 내용**:**
  - 포탄 발사 **(Space** 키**):** 게임이 진행 중일 때(`not self.game_over`) 스페이스바를 누르면 포탄이 발사되어야 합니다. 단, 남은 총알이 있을 때만 발사되어야 합니다.
  - 게임 재시작 **(Return/Enter** 키**):** 게임이 종료되었을 때(`self.game_over`) Return(Enter) 키를 누르면, 게임의 모든 상태가 초기화되고 즉시 새 게임이 시작되어야 합니다. (어떤 메서드를 호출하면 게임이 초기화될지 생각해보세요.)

Problem 5: Game.check_collisions 메서드 **-** 충돌 결과 구현

게임 시나리오**:** "탱크가 돌에 부딪히면 탱크는 초기 위치로 되돌아간다.", "포탄은 ... 돌과 부딪히면 돌과 함께 화면에서 사라진다. ... 50%의 확률로 ... 새로운 돌이 생성된다."

`Game` 클래스의 `check_collisions` 메서드는 게임 내에서 발생하는 모든 충돌 이벤트를 처리합니다. 시나리오에 명시된 두 가지 충돌(탱크와 돌, 포탄과 돌)의 결과를 각각 구현하세요.

- 구현 내용**:**
    - 탱크**-**돌 충돌**:** 플레이어가 돌 그룹과 충돌했는지 감지하고, 충돌 시 플레이어를 시작 지점으로 되돌려야 합니다.
    - 포탄**-**돌 충돌**:** 포탄이 돌 그룹과 충돌했는지 감지해야 합니다. 충돌이 발생하면 해당 포탄과 돌은 모두 화면에서 제거되어야 하며, 50%의 확률로 새로운 돌이 하나 생성되어야 합니다. 이때 포탄이 2개 이상의 돌을 한 번에 맞췄을 때를 고려해서 각 돌에 대해 50%의 확률로 새로운 돌이 하나씩 생성하도록 구현하세요.

Problem 6: Game.draw 메서드 **-** 게임 정보**(UI)** 표시

게임 시나리오**:** "왼쪽 위에 게임이 실행된 이후 경과된 시간이 표시된다." (또한, 게임에는 포탄 개수 제한이 적용되어 UI에 표시되어야 합니다.)

`Game` 클래스의 `draw` 메서드에서, 게임 화면에 필요한 모든 요소를 그리는 부분을 완성하세요. 모든 게임 객체를 그린 후, 게임이 진행되는 동안 화면에 경과 시간과 남은 포탄 개수를 표시해야 합니다.

- 구현 내용**:**
  - 화면 배경과 모든 게임 객체(탱크, 돌 등)를 그려야 합니다.
  - 게임이 진행 중일 때, 화면 좌측 상단에는 시작 후 흐른 시간(초)을, 우측 상단에는 남은 포탄의 개수를 텍스트로 표시해야 합니다. 단, 게임 오버 화면에서는 이 정보들이 보이지 않아야 합니다.
  - 경과 시간과 남은 포탄의 개수를 출력하는 서식은 아래 예시 화면을 참고하세요. x 방향은 각각 왼쪽과 오른쪽에서 20 픽셀 떨어진진 위치에, y 방향은 화면 상단에서 10 픽셀 아래에 배치하세요. 글자 크기는 30으로 지정합니다.

Skeleton code:

수행평가 시 제공