코드쌤: 지혜야, 슬기야! 우리가 지난 시간에는 Player와 Monster를 각각 '스프라이트'로 만들어서 화면에서 활약하게 했었지?

지혜: 네! 몬스터들이 각자 다른 모습과 속도로 화면을 통통 튕겨 다니는 게 정말 신기했어요! 👾💨

슬기: 맞아요! 근데 선생님, 게임에는 점수판도 있고, 지금 몇 라운드인지, 남은 시간은 얼마인지 같은 정보들도 보여줘야 하잖아요. 그런 건 어디서 관리해요? 그리고 몬스터를 그냥 피하기만 하는 게 아니라, 뭔가 특별한 미션이 있으면 더 재미있을 것 같아요!

코드쌤: 오호! 슬기가 게임의 '정보 표시'와 '미션'이라는 아주 중요한 부분을 이야기했네! 오늘은 바로 그런 게임 전체의 상태(점수, 라운드, 시간 등)를 관리하고, 게임의 흐름을 지휘하는 **'게임 지휘관' 역할을 하는 Game 클래스**를 자세히 살펴볼 거야. 그리고 이 Game 클래스가 "이번 라운드에는 이 몬스터를 찾아야 해!" 하고 '수배 몬스터' 미션도 내준단다! 또, 몬스터들이 함부로 침범할 수 없는 '안전지대'도 화면에 그려줄 거야!

(코드 조각 1: 'Game' 클래스 - 게임의 모든 것을 지휘한다!)

```
import pygame, random

pygame.init()

WINDOW_WIDTH = 1200
WINDOW_HEIGHT = 700

FPS = 60
clock = pygame.time.Clock()

window_surface = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
pygame.display.set_caption("수배 몬스터를 찾아라!") # 캡션 변경!

# --- Player 클래스 정의 (lives, safe 속성 추가!) ---
class Player(pygame.sprite.Sprite):
  def __init__(self):
    super().__init__()
    self.image = pygame.image.load("hero.png")
    self.rect = self.image.get_rect()
    self.rect.centerx = WINDOW_WIDTH/2
    self.rect.bottom = WINDOW_HEIGHT
    self.velocity = 8
    self.lives = 5    # 플레이어 목숨 추가!
    self.safe = 3     # '안전' 기회 횟수인가? (이름만으로는 추측!)

  def update(self): # (Player의 update는 이전과 동일)
```

```
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and self.rect.left > 0:
      self.rect.x -= self.velocity
    elif keys[pygame.K_RIGHT] and self.rect.right < WINDOW_WIDTH:
      self.rect.x += self.velocity
    elif keys[pygame.K_UP] and self.rect.top > 0: # (화면 위쪽 경계는 이제 안전지대와 관련될 수도 있겠네요
      self.rect.y -= self.velocity
    elif keys[pygame.K_DOWN] and self.rect.bottom < WINDOW_HEIGHT:
      self.rect.y += self.velocity

# --- Monster 클래스 정의 (monster_type 속성 추가!) ---
class Monster(pygame.sprite.Sprite):
  # 몬스터를 만들 때 x, y, 이미지, 그리고 '몬스터 종류(type)' 정보도 받네!
  def __init__(self, x, y, image, monster_type):
    super().__init__()
    self.image = image
    self.rect = self.image.get_rect()
    self.rect.topleft = (x, y)
    self.velocity = random.randint(1, 5)
    self.monster_type = monster_type # 이 몬스터가 어떤 종류인지 기억! (0:파랑, 1:초록 등)

    self.dx = random.choice([-1, 1])
    self.dy = random.choice([-1, 1])

  def update(self):
    self.rect.x += self.velocity * self.dx
    self.rect.y += self.velocity * self.dy

    # 몬스터가 튕기는 영역이 조금 바뀌었네! 위아래 100픽셀은 못 들어오게!
    if self.rect.left <= 0 or self.rect.right >= WINDOW_WIDTH:
      self.dx *= -1
    # 화면 위쪽 100픽셀 라인 또는 화면 아래쪽에서 100픽셀 위 라인에 닿으면 튕김!
    elif self.rect.top <= 100 or self.rect.bottom >= WINDOW_HEIGHT - 100:
      self.dy *= -1


# --- 여기가 오늘의 주인공! Game 클래스! ---
class Game():
  # Game 객체를 만들 때는 'player' 객체를 꼭 전달해줘야 하네! (플레이어 정보를 알아야 하니까)
```

```
def __init__(self, player):
  # 게임의 기본 정보들을 초기화!
  self.score = 0
  self.round_number = 1
  self.player = player # 전달받은 플레이어 객체를 저장! (나중에 player.lives 같은 정보 쓰려고)
  self.round_time = 0    # 현재 라운드 진행 시간
  self.frame_count = 0   # 프레임 수를 세어서 시간을 계산하는 데 사용

  # 글꼴 설정 (CrotahFreeVersionItalic-z8Ev3.ttf 파일 필요!)
  self.font = pygame.font.Font("CrotahFreeVersionItalic-z8Ev3.ttf", 24)

  # --- "수배 몬스터" 관련 설정! ---
  # 몬스터 이미지들을 미리 리스트에 담아두자 (Game 객체가 직접 관리!)
  blue_monster = pygame.image.load("blue_monster.png")
  green_monster = pygame.image.load("green_monster.png")
  orange_monster = pygame.image.load("orange_monster.png")
  purple_monster = pygame.image.load("purple_monster.png")
  self.monster_images = [blue_monster, green_monster, orange_monster, purple_monster]

  # 이번 라운드에 찾아야 할 "수배 몬스터" 종류를 무작위로 선택! (0~3 사이 숫자)
  self.wanted_monster_type = random.randint(0, 3)
  # 선택된 종류에 해당하는 실제 몬스터 이미지를 가져옴!
  self.wanted_monster_image = self.monster_images[self.wanted_monster_type]
  # 수배 몬스터 이미지를 화면 상단 중앙에 표시하기 위한 rect 설정!
  self.wanted_monster_rect = self.wanted_monster_image.get_rect()
  self.wanted_monster_rect.centerx = WINDOW_WIDTH/2
  self.wanted_monster_rect.top = 35 # 위에서 35픽셀 떨어진 곳에!

# Game 객체의 상태를 업데이트하는 메소드! (지금은 시간만 업데이트)
def update(self):
  self.frame_count += 1 # 매 프레임마다 카운트 1 증가!
  if self.frame_count == FPS: # 프레임 카운트가 FPS(60)와 같아지면? (즉, 1초가 지나면!)
    self.round_time += 1    # 라운드 진행 시간 1초 증가!
    self.frame_count = 0    # 프레임 카운트는 다시 0으로 초기화!

# Game 관련 정보들을 화면에 그리는 메소드!
def draw(self):
  # 색깔 정의 (draw 메소드 안에서만 쓰니까 여기에 있어도 괜찮아)
  WHITE = (255, 255, 255)
```

```
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
ORANGE = (255, 128, 0)
PURPLE = (102, 0, 204)
colors = [BLUE, GREEN, ORANGE, PURPLE] # 수배 몬스터 테두리 색깔용!

# --- 화면에 표시할 각종 텍스트 정보들 만들기! ---
# self.score, self.round_number, self.player.lives, self.round_time, self.player.safe
# 이런 식으로 Game 객체가 가진 정보나, Game 객체가 알고 있는 player 객체의 정보를 가져와서 텍스트로 만
score_text = self.font.render(f"Score: {self.score}", True, WHITE)
score_rect = score_text.get_rect()
score_rect.topleft = (10, 10)

round_text = self.font.render(f"Round: {self.round_number}", True, WHITE)
round_rect = round_text.get_rect()
round_rect.topleft = (10, 40)

lives_text = self.font.render(f"Lives: {self.player.lives}", True, WHITE)
lives_rect = lives_text.get_rect()
lives_rect.topleft = (10, 70)

wanted_text = self.font.render("Wanted Monster", True, WHITE)
wanted_rect = wanted_text.get_rect()
wanted_rect.centerx = WINDOW_WIDTH/2
wanted_rect.top = 10

round_time_text = self.font.render(f"Round Time: {self.round_time}", True, WHITE)
round_time_rect = round_time_text.get_rect()
round_time_rect.topright = (WINDOW_WIDTH - 10, 10)

safe_zone_text = self.font.render(f"Safe Zone: {self.player.safe}", True, WHITE)
safe_zone_rect = safe_zone_text.get_rect()
safe_zone_rect.topright = (WINDOW_WIDTH - 10, 40)

# --- 만들어진 텍스트와 "수배 몬스터" 이미지를 화면에 그리기! ---
window_surface.blit(score_text, score_rect)
window_surface.blit(round_text, round_rect)
window_surface.blit(lives_text, lives_rect)
window_surface.blit(wanted_text, wanted_rect)
```

```
        window_surface.blit(self.wanted_monster_image, self.wanted_monster_rect) # 수배 몬스터 이미지!
        window_surface.blit(round_time_text, round_time_rect)
        window_surface.blit(safe_zone_text, safe_zone_rect)

        # --- "수배 몬스터" 주변에 테두리 그려주기! (색깔은 몬스터 종류에 맞게!) ---
        pygame.draw.rect(window_surface, colors[self.wanted_monster_type], self.wanted_monster_rect, 2)
        # --- 화면 위아래에 "안전지대" 테두리 그려주기! (몬스터 못 들어오는 곳) ---
        # (0, 100) 위치에서 가로 WINDOW_WIDTH, 세로 (WINDOW_HEIGHT - 200) 크기의 사각형!
        pygame.draw.rect(window_surface, colors[self.wanted_monster_type], (0, 100, WINDOW_WIDTH, WINDOW_HEIG

# ... (플레이어 그룹 및 몬스터 그룹 생성, Game 객체 생성, 게임 루프 코드는 아래에) ...
```

지혜: Game 클래스의 __init__ 메소드를 보니까 self.score, self.round_number, self.round_time 처럼 게임 점수, 라운드, 시간 같은 정보들을 다 가지고 있네요! 그리고 self.player = player 이렇게 플레이어 객체도 알고 있고요! 진짜 게임 지휘관 같아요!

코드쌤: 맞아! Game 클래스는 게임 전체의 상태를 나타내는 중요한 변수들을 한 곳에 모아서 관리해. 그리고 self.monster_images 리스트에 파랑, 초록, 주황, 보라 몬스터 이미지를 미리 다 불러와서 저장해두네. 이걸로 뭘 하려는 걸까?

슬기: self.wanted_monster_type = random.randint(0, 3) 여기서 0부터 3까지 숫자 중에 하나를 뽑고, self.wanted_monster_image = self.monster_images[self.wanted_monster_type] 이걸로 그 숫자에 해당하는 몬스터 이미지를 '이번 라운드 수배 몬스터'로 정하는 거네요! 매번 게임 할 때마다 다른 몬스터를 찾아야 하는 미션이 생기겠어요!

코드쌤: 정확해! Game 클래스가 "이번엔 이 녀석을 잡아와!" 하고 임무를 주는 거지. 그리고 Game 클래스의 update() 메소드를 보면 self.frame_count를 이용해서 self.round_time을 1초마다 증가시키고 있어. FPS가 60이니까, 프레임이 60번 그려지면 1초가 지난 거잖아? 그걸로 게임 시간을 정확하게 재는 거야.

지혜: Game 클래스의 draw() 메소드는 정말 많은 걸 그리네요! 점수, 라운드, 목숨, 수배 몬스터 제목, 실제 수배 몬스터 그림, 라운드 시간, 그리고 self.player.safe 값까지! self.player.safe는 뭐예요? 플레이어 클래스에도 self.safe = 3 이렇게 새로 생겼던데!

코드쌤: self.player.safe는 아마도 플레이어가 몬스터와 부딪혔을 때 목숨 대신 깎이는 '안전 보호막' 같은 걸 의미하는 것 같아. (지금 코드에서는 실제로 사용되는 로직은 없지만, 이렇게 변수를 만들어뒀다는 건 나중에 그런 기능을 추가할 수 있다는 뜻이겠지?) 그리고 draw() 메소드 마지막 부분을 보면 pygame.draw.rect로 두 개의 사각형을 그리는데, 하나는 수배 몬스터 이미지 주변에, 다른 하나는 화면 중앙 넓은 영역에 테두리를 그리고 있어.

슬기: 아! 수배 몬스터 테두리는 "얘가 이번 목표야!" 하고 알려주는 거고, 화면 중앙에 그려진 커다란 사각형 테두리가 바로 '안전지대'인가 봐요! 몬스터 클래스의 update 메소드를 보니까, 몬스터들이 화면 위에서 100픽셀, 아래에서 100픽셀 떨어진 곳에서는 튕겨 나가도록 바뀌었거든요! 그럼 저 가운데 안전지대에는 몬스터들이 못 들어오겠네요!

코드쌤: 빙고! Game 클래스가 화면에 UI(사용자 인터페이스)와 게임 규칙을 시각적으로 보여주고, Monster 클래스는 그 규칙(안전지대)에 맞춰서 움직이도록 설계된 거야. 이렇게 각 클래스가 자기 역할을 잘 해내면 복잡한 게임도 체계적으로 만들 수 있단다.

(코드 조각 2: 객체 생성 및 게임 루프에서의 조화로운 합주!)

```
# ... (Player, Monster, Game 클래스 정의는 위와 동일) ...

my_player_group = pygame.sprite.Group()
my_player = Player() # Player 객체 먼저 만들고!
my_player_group.add(my_player)

# Game 객체를 만들 때, 위에서 만든 my_player를 전달해줘야 해!
my_game = Game(my_player) # 게임 지휘관 등장!

# 몬스터 군단 생성!
my_monster_group = pygame.sprite.Group()
# monster_images 리스트는 이제 Game 객체가 가지고 있으니, 그걸 사용해야 할 것 같아!
# (원래 코드에서는 전역 변수처럼 monster_images를 바로 썼는데, Game 객체의 것을 쓰는게 더 깔끔하겠죠?)
# 여기서는 원래 코드의 흐름을 따르기 위해 전역 monster_images가 있다고 가정하고 진행할게요.
# 하지만 더 좋은 설계는 my_game.monster_images를 사용하는 것입니다!
monster_images_for_creation = [pygame.image.load("blue_monster.png"), pygame.image.load("green_monster.pn
monster_type_index = 0 # 몬스터 종류를 순서대로 부여하기 위한 임시 변수 (원래 코드엔 없음)

for img in monster_images_for_creation: # 각 이미지마다
  # 몬스터를 만들 때, 몇 번째 이미지인지(monster_type)도 함께 전달!
  # (원래 코드에서는 Monster 생성 시 monster_type에 항상 1을 넣었는데, 실제로는 이미지에 맞게 넣어줘야겠죠
  # 여기서는 임의로 monster_type_index를 사용해서 0, 1, 2, 3을 부여한다고 가정해볼게요.
  my_monster = Monster(random.randint(0, WINDOW_WIDTH - 64), random.randint(101, WINDOW_HEIGHT - 165), im
  my_monster_group.add(my_monster)
  monster_type_index += 1 # 다음 몬스터는 다음 타입으로!


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

  # --- 게임 세상의 모든 것을 업데이트하고 그리는 순서! ---
```

```
    window_surface.fill((0, 0, 0)) # 1. 배경은 매번 새로!

    # 2. 플레이어 팀 업데이트하고 그리기!
    my_player_group.update()
    my_player_group.draw(window_surface)

    # 3. 몬스터 군단 업데이트하고 그리기!
    my_monster_group.update()
    my_monster_group.draw(window_surface)

    # 4. 게임 지휘관(Game 객체)도 자기 할 일 하기! (정보 그리고, 시간 업데이트!)
    my_game.draw()   # Game 객체가 가진 정보들을 화면에 그려줘!
    my_game.update() # Game 객체의 시간 등을 업데이트 해줘!

    pygame.display.update() # 5. 최종적으로 완성된 한 장면을 화면에 딱!
    clock.tick(FPS)         # 6. 시간아, 약속된 속도로 흘러가렴!

pygame.quit()
```

지혜: `my_game = Game(my_player)` 이렇게 Game 객체를 만들 때 `my_player`를 넣어주니까, Game 클래스 안에서 `self.player.lives` 처럼 플레이어의 정보를 쉽게 가져다 쓸 수 있겠네요!

코드쌤: 맞아! 이렇게 객체들이 서로 정보를 주고받으면서 협력하는 것이 객체 지향 프로그래밍의 중요한 특징 중 하나야. 그리고 몬스터를 만들 때 `Monster(..., img, monster_type_index)` 이렇게 `monster_type` 정보도 같이 넘겨주면, 나중에 "플레이어가 잡은 몬스터가 수배 몬스터랑 같은 종류인가?"를 확인할 때 이 `monster_type`을 사용할 수 있겠지? (지금 코드에서는 몬스터 생성 시 `monster_type`에 항상 1을 넣고 있는데, 실제로는 각 이미지에 맞는 타입을 넣어주는 로직이 필요할 거야. 여기서는 설명을 위해 `monster_type_index`라는 가상의 변수를 사용해서 각기 다른 타입이 들어간다고 가정했어.)

슬기: 게임 루프 안에서 `my_game.draw()`랑 `my_game.update()`가 추가됐네요! 그럼 매 프레임마다 Game 클래스가 화면에 점수판이랑 수배 몬스터 정보 그려주고, 시간도 계속 업데이트하는 거군요! 진짜 지휘관처럼 게임 전체를 돌보고 있어요!

코드쌤: 정확해! Player와 Monster는 각자 자기 할 일(움직이기, 튕기기)을 하고, Game 클래스는 게임 전체의 규칙을 적용하고 정보를 보여주는 역할을 하는 거지. 이렇게 각 클래스가 자기 역할에만 집중하면, 나중에 게임이 더 복잡해져도 코드를 이해하고 수정하기가 훨씬 쉬워진단다.

지혜: 오늘 Game 클래스라는 새로운 지휘관을 만나서 게임이 훨씬 더 체계적으로 돌아가는 것 같아요! 수배 몬스터를 잡는 미션이 생기니까 이제 뭘 해야 할지 목표도 분명해졌고요!

코드쌤: 훌륭해! 오늘 우리는 Game 클래스를 통해 게임 전체의 상태를 관리하고, 게임의 흐름을 제어하는 방법을 배웠어. 그리고 '수배 몬스터'나 '안전지대' 같은 새로운 게임 규칙들이 어떻게 코드로 구현되는지도 살펴봤지. 이제 이 게임에 플레이어가 수배 몬스터를 잡았을 때 점수를 더 많이 주거나, 다른 몬스터와 부딪혔을 때 목숨이

깎이는 '충돌 처리' 로직만 추가하면 더욱 완벽한 게임이 될 수 있을 거야! 그건 우리의 다음 모험이 되겠지?