TROPICAL ISLAND

2팀

팀장 : 신영민

김민성 김현수 신지혜 조준한

CONTENTS

Class

- Player
- Enemy
- Score Board
- Special Move

Python Code

- List, Dictionary
- Conditional Statements
- Loops
- Module
- Function
- Class, Object

CLASS

CLASS - Player

플레이어 클래스
Class : Player
Method : __init__(), update(), shoot(), find_nearest_enemy()

* **__init__()** : 캐릭터 기본 상태 설정
캐릭터 성별 선택에 따라 다른 이미지가 보여지도록 설정하고 키보드 방향키에 따라 보여지는 이미지가 다르게 설정했다. 기본 캐릭터 사이즈를 설정해 각 모션 별 이미지 사이즈를 통일했고 스크린 폭의 넓이와 깊이를 활용해 기본 위치를 설정했다. 또한 이동 속도와 체력, 공격력과 공격 속도 등 캐릭터의 기본 속성을 설정했다.

* **update()** :캐릭터 이동 설정
입력된 키에 따라 상하좌우 위치 이동량을 주었고 화면의 경계를 설정해 화면 밖으로는 이동하지 못하게 설정했다. 이동 방향에 따라 다른 이미지가 나오게 설정하고 이동중이 아닐 때는 마지막 방향에 맞춰 정지 이미지를 로드하게 했다.

* **shoot()** : 캐릭터 공격 설정
공격 쿨타임 차면 가까운 적을 찾아 타겟으로 설정 후 공격하게 했다. 또한 다음 스테이지로 넘어갈 때 사선 투사체 아이템을 획득한 경우에 투사체의 방향 설정을 바꿔 공격할 수 있게 설정했다. 이 또한 화면의 경계를 설정해 화면 안에 있도록 좌표를 조정했다.

* **find_nearest_enemy()** : 가장 가까운 적 찾기
보스 몬스터의 방어할 때를 제외하고 가장 가까운 적을 찾아 타켓으로 설정하게 했다. 모든 적에 대해 플레이어와 적 간의 거리를 계산해 가장 가까운 거리에 있는 적을 반환한다.

```python
# 플레이어 클래스
class Player(pygame.sprite.Sprite):
    def __init__(self, game, depth=0):
        super().__init__()
        self.game = game  # Game 객체 저장
        self.select_character = 'male'
```

```python
    def update(self):
        # 이동 처리
        keys = pygame.key.get_pressed() # 키 입력 상태 확인
        self.is_moving = False # 이동 상태 초기화

        dx = 0 # x축 이동량
```

```python
    def shoot(self, current_time, all_sprites, projectiles, enemies, on_reflect):
        if current_time - self.last_shot > 500 / self.attack_speed:  # 공격 쿨타임이 지났으면
            target = self.find_nearest_enemy(enemies, 600)  # 가장 가까운 적 찾기, 사정거리
            if target:  # 적이 있으면
                # 발사할 투사체 리스트 초기화
                projectile_list = []
```

```python
    def find_nearest_enemy(self, enemies, max_distance):
        nearest_enemy = None  # 가장 가까운 적
        min_distance = float('inf')  # 최소 거리 (무한대로 초기화)

        for enemy in enemies:  # 모든 적에 대해

            if monster.Enemy.is_shield and enemy.enemy_type == 'boss': # 보스가 방어시 타게팅에 안잡히게 설정
                continue

            distance = math.sqrt(  # 플레이어와 적 간의 거리 계산
```

CLASS – Enemy(1)

적 클래스

Class : Enemy
Method : __init__(), update(), load_images(), update_animation(), melee_ behavior(), ranged_behavior(), boss_behavior(), check_boss_state()

* **__init__()** : 적 이미지 및 위치 세팅, 난이도 별 적의 최대 hp와 공격 데미지 설정

* **update()** : 근거리 공격과 원거리 공격 그리고 보스 몬스터의 애니메이션 설정. 화면 안에서 움직이게 설정했다.

* **load_images()** : 이미지 로드

* **update_animation()** : 보스 몬스터와 일반 몬스터의 애니메이션 처리. 보스 몬스터의 경우 방어, 공격, 이동, 대기 모션이 있고 일반 몬스터는 공격, 이동, 대기 모션이 있다.

```python
class Enemy(pygame.sprite.Sprite):
    is_shield = False

    def __init__(self, game, x, y, stage, select_difficulty, enemy_type="melee", depth=20):
        super().__init__()

        self.game = game
        self.enemy_type = enemy_type # 적 타입 (melee: 근접, ranged: 원거리)
        self.attack_images = None
        self.shield_images = None
        self.stage = stage
```

```python
    def update(self, player, current_time, all_sprites, enemy_projectiles):
        if self.enemy_type == "melee": # 근접 적
            self.melee_behavior(player)
        elif self.enemy_type == "ranged": # 원거리 적
            self.ranged_behavior(player, current_time, all_sprites, enemy_projectiles)
        elif self.enemy_type == "boss": # 보스
```

```python
    def load_images(self, sprite_sheet, frame_width, frame_height, frame_count, rows=0):
        images = []
        if rows != 0:
            rows -= 1

        for i in range(frame_count):
            frame_image = pygame.Surface((frame_width, frame_height), pygame.SRCALPHA).convert_alpha() # Surface 생성 시 투명도 설정 추가
            frame_rect = pygame.Rect(i * frame_width, rows * frame_height, frame_width, frame_height) # Rect 객체 사용
```

```python
    def update_animation(self):
        # 보스 타입일 경우의 애니메이션 처리
        if self.enemy_type == "boss":
            if Enemy.is_shield and self.shield_images:  # 방어 모션
                self.shield_frame_counter += 1
                if self.shield_frame_counter >= self.animation_speed:
                    self.shield_frame_counter = 0
                    self.shield_frame = (self.shield_frame + 1) % len(self.shield_images)

                    if self.facing_direction == "right":
                        self.image = self.shield_images_right[self.shield_frame]
                    else:
                        self.image = self.shield_images[self.shield_frame]
```

CLASS – Enemy(2)

* **melee_behavior()** : 몬스터의 근거리 공격 행동 설정. 감지 범위 안에 있을 때 플레이어 방향에 맞게 몬스터 방향을 설정해 충돌했을 시 공격을 하게 설정하고 감지 범위 밖에 있을 때 공격 범위 안에 있으면 멈추고 공격을 하게 설정했다.

* **ranged_behavior()** : 몬스터 원거리 공격 행동 설정. 공격 범위 안에 플레이어가 있으면 이동을 멈추고 발사체를 던지게 설정했고 공격 범위 밖에 있으면 공격을 멈추고 플레이어 방향에 맞춰 몬스터가 이동하게 설정했다.

* **boss_behavior()** : 보스 몬스터 행동 설정. 보스 몬스터의 이동과 공격 그리고 휴식 상태 구현. 휴식 중일 땐 공격을 생략한다. 공격 로직은 쿨타임을 설정해 두고 6번 공격 후에 휴식상태로 전환 되게 설정했다.

* **check_boss_state()** : 보스의 상태를 체크하고 업데이트한다. Hp에 따라 분노모드 진입 및 해제 그리고 다른 몬스터를 소환하게 설정했다. Hp가 80, 60, 40, 30에 돌입했을 때 각 기능을 수행한다.

```
class Enemy(pygame.sprite.Sprite):
    def melee_behavior(self, player):


        else: # 감지 범위 밖에 있으면
            self.is_moving = False
```

```
    # 원거리 행동
    def ranged_behavior(self, player, current_time, all_sprites, enemy_projectiles):
        dx = player.rect.centerx - self.rect.centerx # 플레이어와 x좌표 차이
        dy = player.rect.centery - self.rect.centery # 플레이어와 y좌표 차이
        distance = math.sqrt(dx ** 2 + dy ** 2) # 플레이어와 거리

        if distance < self.attack_range: # 공격 범위 안에 있으면
            self.is_moving = False # 이동 멈춤
            self.under_attack = True # 공격
```

```
    # 보스 행동
    def boss_behavior(self, player, current_time, all_sprites, enemy_projectiles):
        dx = player.rect.centerx - self.rect.centerx
        dy = player.rect.centery - self.rect.centery
        distance = math.sqrt(dx ** 2 + dy ** 2)

        # 보스 체력 확인 및 소환 로직을 맨 앞으로 이동
        self.check_boss_state()

        if Enemy.is_shield:
```

```
    def check_boss_state(self):
        """보스의 상태를 체크하고 업데이트하는 메서드"""
        for enemy in self.game.enemies:
            if enemy.enemy_type == 'boss':
                hp_percent = int((enemy.hp / enemy.max_hp) * 100)

                # 첫 번째 소환 및 방어 페이즈 (80%)
                if hp_percent <= 80 and not self.has_spawned and not hasattr(self, 'first_spawn_done'):
                    self.game.spawn_stage_enemies()
                    self.has_spawned = True
                    self.first_spawn_done = True  # 첫 번째 소환 완료 표시
                    Enemy.is_shield = True
                    self.under_attack = False
```

CLASS - Score Board

점수판 기능

* 최종 점수는 처리한 적의 수에 난이도별 포인트, 완료한 스테이지 수, 남은 시간을 더해 곱해 계산한다.

```
class Game:

    def start_next_stage(self, current_time):
        # 점수 갱신
        elapsed_time = (current_time - self.start_time) / 1000 # 경과 시간 계산
        self.time_left += max(0, self.stage_time - elapsed_time) # 남은 시간 계산

        self.score = self.enemy_kill_count * (self.select_difficulty + self.stage + self.time_left)
```

* 게임 오버 화면과 스테이지 클리어 화면에 각각 계산한 최종 점수를 텍스트로 띄운다

```
score_text = self.button_font.render(f"Score: {int(self.score)}", True, main.WHITE) # 최종 점수
score_rect = score_text.get_rect(center=(main.SCREEN_WIDTH//2, main.SCREEN_HEIGHT*0.52)) # 텍스트 Rect
self.screen.blit(score_text, score_rect) # 텍스트 그리기
```

Class - Ultimate

필살기 클래스

Class : Ultimate
Method : __init__(), use_ultimate()

* **use_ultimate()** : 필살기 활성화와 필살기 사용 시간 설정, 필살기에 활용되는 모션 불러오기.

```
class Ultimate():
    def __init__(self, game):
        self.game = game
        pass

    def use_ultimate(self, all_enemies): # 필살기 사용시 화면에 모든 적 제거(김현수)
        self.is_using_ultimate = True # 필살기 사용 활성화
        self.ultimate_start_time = pygame.time.get_ticks()  # 🔥 필살기 시작 시간 저장

        pygame.time.set_timer(pygame.USEREVENT + 1, 1000) # 1초후 필살기 사용 종료

        fireball = Fireball(self.game.player.rect.centerx, self.game.player.rect.top, "up", self.game)
        self.game.all_sprites.add(fireball)
        self.game.player_projectiles.add(fireball)
```

CLASS - Explosion

필살기 클래스

Class : Explosion
Method : __init__(), update()

* **__init__()** : 폭발 이미지 로드 및 크기 조정, 프레임 설정

* **update()** : 폭발 애니메이션 프레임 설정, 애니메이션이 끝나면 화면에서 이미지 삭제

```python
class Explosion(pygame.sprite.Sprite): # 폭발 애니메이션 추가(김현수)
    def __init__(self, x, y, game):
        super().__init__()
        self.game = game

        # 폭발 이미지
        explosion_frames = [os.path.join(main.resources_path, f"effect_explosion_0{i}.png") for i in range(1,9)]

        # 이미지 로드 및 크기 조정
        self.frames = [pygame.image.load(img).convert_alpha() for img in explosion_frames]
        self.frames = [pygame.transform.scale(frame, (300, 300)) for frame in self.frames]

        self.image = self.frames[0]  # 첫 번째 이미지
        self.rect = self.image.get_rect(center=(x, y))

        self.current_frame = 0  # 현재 애니메이션 프레임
        self.last_update = pygame.time.get_ticks()  # 마지막 업데이트 시간
        self.frame_speed = 50  # 프레임 속도 (50ms마다 다음 프레임)

    def update(self):
        """💥 애니메이션 업데이트"""
        now = pygame.time.get_ticks()
        if now - self.last_update > self.frame_speed:
            self.last_update = now
            self.current_frame += 1

            if self.current_frame < len(self.frames):
                self.image = self.frames[self.current_frame]  # 다음 프레임
            else:
                self.kill()  # 애니메이션 끝나면 삭제
```

CLASS - Fireball

필살기 클래스

Class : Fireball
Method : __init__(). update(), explode()

* **__init__()** : 불꽃 이미지 로드, 속도, 거리 등 조정, 방향에 따라 이미지 이동 방향 설정

* **update()** : 불꽃 이동과 거리를 체크하고 일정 거리가 지나면 이미지가 화면에서 사라지게 설정.
한 번 폭발하면 중복돼서 한 번 더 폭발하지 않는다.

* **explode()** : 불꽃이 폭발하게 되면 적을 삭제하고 폭발 애니메이션을 실행한다.
적의 HP를 HP의 최대치만큼 감소시켜 HP가 0 이하일 때 삭제되는 방식 사용.

```python
class Fireball(pygame.sprite.Sprite): # 파이어볼 클래스 (김현수)
    def __init__(self, x, y, direction, game):
        super().__init__()

        # 불꽃 발사체 이미지 로드
        original_image = pygame.image.load(os.path.join(main.resources_path, "fireball.png")).convert_alpha()
        image_size = (25, 80)
        self.image = pygame.transform.scale(original_image, image_size)
        self.rect = self.image.get_rect(center=(x, y))

        self.speed = 3  # 불꽃 발사체 속도
        self.distance_traveled = 0  # 이동 거리 추적
        self.max_distance = 300  # 불꽃 발사체가 이동할 최대 거리
        self.game = game  # 게임 객체 저장
        self.damage = 0

        self.exploded = False  # 폭발이 이미 발생했는지 확인하는 변수 추가

        # dx, dy 초기화 추가
        self.dx, self.dy = 0, 0
```

```python
def update(self):
    """불꽃 발사체 이동 및 거리 체크"""
    self.rect.x += self.dx
    self.rect.y += self.dy
    self.distance_traveled += self.speed

    # 🔥 적과 충돌 확인 (디버깅용 로그 추가)
    for enemy in self.game.enemies:
        if self.rect.colliderect(enemy.rect) or self.distance_traveled >= self.max_distance and not self.exploded:
            self.exploded = True  # 중복 폭발 방지
            self.explode()
            self.kill()
            return  # 한 번 폭발하면 더 이상 진행하지 않음

    # # 최대 거리에 도달하면 폭발하고 모든 적을 제거
    # if self.distance_traveled >= self.max_distance and not self.exploded:
    #     self.exploded = True  # 중복 폭발 방지
    #     self.explode()  # 적 제거 + 폭발 애니메이션 생성
    #     self.kill()  # 파이어볼 제거

def explode(self):
    """🔥 파이어볼이 폭발하면 적 제거 + 폭발 애니메이션 실행"""
    enemies_to_remove = list(self.game.enemies)  # 모든 적 복사
    for enemy in enemies_to_remove:
        explosion = Explosion(enemy.rect.centerx, enemy.rect.centery, self.game)  # 적 위치에서 폭발
        self.game.all_sprites.add(explosion)  # 폭발 애니메이션 추가
        enemy.hp-=50 # HP -50
        if enemy.hp <= 0:
            enemy.kill()
            self.game.enemy_kill_count += 1
        #enemy.kill()  # 적 제거
        #self.game.enemies.remove(enemy)  # 리스트에서도 제거
```

Python Code

Python Code

List 리스트

: 여러 개의 데이터를 담을 수 있는 자료형.

예시 : 타이틀 글자, 몬스터 리스트

타이틀 글자의 경우 각 글자를 데이터로 각 줄의 리스트에 담았다.

몬스터 리스트의 경우 각 몬스터를 리스트에 담았고 몬스터의 세부 정보는 리스트 내 딕셔너리로 처리했다.

```python
def draw_title(self): # 각 글자 이미지를 화면 중앙 상단에 배치
    first_line = [ 'T', 'R', 'O', 'P', 'I', 'C', 'A', 'L' ]
    second_line = [ 'I', 'S', 'L', 'A', 'N', 'D' ]
```

```python
monster_list = [
    {
        'name': 'angry_pig', # 이름
        'max_hp': 80, # 최대 체력
        'speed': 3, # 이동 속도
        'attack_damage': 5, # 공격력 / 근거리 적용 안됨
        'detection_range': 800, # 플레이어 인식 범위
        'attack_range': 50, # 공격 범위(실제 공격하는 범위가 아님)
        'attack_cooldown': 1000, # 공격 쿨타임 (ms) / 근거리 아직 적용 안됨
        'type': 'melee', # 타입
        'images': [
            # 몬스터의 경우 왼쪽 모션만 사용
            # 대기 모션
            {'type': 'idle_image', 'file_name': 'angry_pig_idle_36x30.png', 'info': (36, 30, 9), 'size': 120},

            # 이동 모션
            {'type': 'move_image', 'file_name': 'angry_pig_move_36x30.png', 'info': (36, 30, 16), 'size': 120},

            # 공격 모션
            {'type': 'attack_image', 'file_name': 'angry_pig_attack_36x30.png', 'info': (36, 30, 3), 'size': 120},

            # 발사 무기
            {'type': 'projectile', 'file_name': 'flash_bomb.png', 'size': 15},

            # 타격 이펙트
            {'type': 'effect', 'file_name': 'effect_01.png'}

        ] # 이미지

    },
    {
        'name': 'trunk', # 이름
        'max_hp': 80, # 최대 체력
        'speed': 3, # 이동 속도
        'attack_damage': 5, # 공격력 / 근거리 적용 안됨
        'detection_range': 800, # 플레이어 인식 범위
        'attack_range': 50, # 공격 범위(실제 공격하는 범위가 아님)
        'attack_cooldown': 1000, # 공격 쿨타임 (ms) / 근거리 아직 적용 안됨
        'type': 'melee', # 타입
        'images': [
            # 몬스터의 경우 왼쪽 모션만 사용
            # 대기 모션
            {'type': 'idle_image', 'file_name': 'trunk_idle_64x32.png', 'info': (64, 32, 18), 'size': 120},

            # 이동 모션
            {'type': 'move_image', 'file_name': 'trunk_move_64x32.png', 'info': (64, 32, 14), 'size': 120},

            # 공격 모션
```

Python Code

Dic 딕셔너리

: key와 value가 상호 간 매핑 되어 쌍으로 구성된 자료구조

예시 : 몬스터(angry pig)

몬스터의 이름, 체력, 이동 속도, 공격력 등의 기본 정보를 각각 항목과 값으로 지정해 저장했다.

```
{
    'name': 'angry_pig', # 이름
    'max_hp': 80, # 최대 체력
    'speed': 3, # 이동 속도
    'attack_damage': 5, # 공격력 / 근거리 적용 안됨
    'detection_range': 800, # 플레이어 인식 범위
    'attack_range': 50, # 공격 범위(실제 공격하는 범위가 아님)
    'attack_cooldown': 1000, # 공격 쿨타임 (ms) / 근거리 아직 적용 안됨
    'type': 'melee', # 타입
    'images': [
        # 몬스터의 경우 왼쪽 모션만 사용
        # 대기 모션
        {'type': 'idle_image', 'file_name': 'angry_pig_idle_36x30.png', 'info': (36, 30, 9), 'size': 120},

        # 이동 모션
        {'type': 'move_image', 'file_name': 'angry_pig_move_36x30.png', 'info': (36, 30, 16), 'size': 120},

        # 공격 모션
        {'type': 'attack_image', 'file_name': 'angry_pig_attack_36x30.png', 'info': (36, 30, 3), 'size': 120},

        # 발사 무기
        {'type': 'projectile', 'file_name': 'flash_bomb.png', 'size': 15},

        # 타격 이펙트
        {'type': 'effect', 'file_name': 'effect_01.png'}

    ] # 이미지

},
```

Python Code

if, elif, else 조건문

: 조건에 따라 코드를 실행

예시 : 스테이지 전환, hp바 이미지

스테이지가 7이 되면 보스 스테이지를 실행하고 8이상이 됐을 땐 게임 클리어 화면으로 전환한다.

플레이어의 체력에 따라 현재 hp의 상태를 보여주는 hp바 이미지를 다르게 설정한다.

```python
if self.stage == 7: # 보스 스테이지
    self.boss = True

elif self.stage == 8: # 게임 클리어
    self.game_state = GameState.CLEAR
```

```python
# 현재 체력 이미지 생성
if self.player.hp == self.player.max_hp:  # 최대 HP일 때
    hp_image = self.hp_image01
elif self.player.hp > self.player.max_hp - 15:
    hp_image = self.hp_image02
elif self.player.hp > self.player.max_hp - 30:
    hp_image = self.hp_image03
elif self.player.hp > self.player.max_hp - 45:
    hp_image = self.hp_image04
elif self.player.hp > self.player.max_hp - 60:
    hp_image = self.hp_image05
elif self.player.hp > self.player.max_hp - 75:
    hp_image = self.hp_image06
else:
    hp_image = self.hp_image07
```

Python Code

for, while 반복문

: 코드를 반복 실행하는 역할

예시 : 적 공격, 게임 실행

근거리, 원거리 공격 적의 공격을 받았을 때 공격을 받는 동안의 플레이어 체력이 0이하가 되면 게임 오버로 상태 변경하게 for문과 if문을 사용해 설정했다.

게임을 실행하는 동안 화면의 상태를 업데이트하고 각 이벤트를 실행할 수 있게 while문을 사용해 설정했다.

```
# 적 투사체와 플레이어 충돌
hits = pygame.sprite.spritecollide(self.player, self.enemy_projectiles, True) # 플레이어와 충돌한 적 투사체 확인 (투사체는 사라짐)
for hit in hits: # 충돌한 모든 투사체에 대해
    self.player.hp -= hit.damage # 플레이어 체력 감소
    if self.player.hp <= 0: # 플레이어 체력이 0 이하이면
        self.game_state = GameState.GAME_OVER # 게임 오버 상태로 변경

# 근접 적과 플레이어 충돌
hits = pygame.sprite.spritecollide(self.player, self.enemies, False) # 플레이어와 충돌한 적 확인
for enemy in hits: # 충돌한 모든 적에 대해
    if enemy.enemy_type == "melee": # 근접 적일 경우
        self.player.hp -= enemy.attack_damage * 0.1  # 초당 데미지 (0.1초마다 데미지 적용)
        if self.player.hp <= 0: # 플레이어 체력이 0 이하이면
            self.game_state = GameState.GAME_OVER # 게임 오버 상태로 변경
```

```
running = True # 게임 실행 상태
self.current_module.update()
while running: # 게임 루프
    current_time = pygame.time.get_ticks() # 현재 시간

    self.draw(current_time) # 화면 그리기
    pygame.display.flip() # 화면 업데이트
    self.clock.tick(FPS) # 프레임 속도 제한
```

Python Code

Module 모듈

: 함수, 변수, 클래스를 갖고 있는 파일

예시 : main.py

Main함수에서 게임 제작을 위해 pygame, os, random, math, enum을 사용했고 게임 플레이를 위해 분리해둔 몬스터, 플레이어 등의 모듈을 사용했다.

```
import pygame
import os
import random
import math
from enum import Enum

from modules import monster
from modules import player
from modules import portal
from modules import gameover
from modules import menu
from modules import difficulty
from modules import clear
from modules import ultimate
from modules import music
from modules.object import Object
```

Python Code

function 함수

: 호출될 때 특정 기능을 실행하는 재사용 가능한 코드 블록

예시 : 가까운 적 찾는 함수

대표적으로 플레이어와 적 간의 거리를 계산해서 비교한 후 가장 가까운 거리에 있는 적을 반환하는 함수가 있다.

```python
def find_nearest_enemy(self, enemies, max_distance):
    nearest_enemy = None  # 가장 가까운 적
    min_distance = float('inf')  # 최소 거리 (무한대로 초기화)

    for enemy in enemies:  # 모든 적에 대해

        if monster.Enemy.is_shield and enemy.enemy_type == 'boss': # 보스가 방어시 타게팅에 안잡히게 설정
            continue

        distance = math.sqrt(  # 플레이어와 적 간의 거리 계산
            (enemy.rect.centerx - self.rect.centerx) ** 2 +
            (enemy.rect.centery - self.rect.centery) ** 2
        )

        if distance < min_distance and distance <= max_distance:  # 현재까지의 최소 거리보다 더 가깝고, 최대 거리 이내에 있으면
            min_distance = distance  # 최소 거리 갱신
            nearest_enemy = enemy  # 가장 가까운 적 갱신

    return nearest_enemy # 가장 가까운 적 반환
```

Python Code

Class, object 클래스와 객체

: 클래스는 객체를 생성하기 위한 설계도 역할을 한다. 속성과 메서드를 정의해 여러 개의 객체를 만들 수 있다.

: 객체는 클래스를 통해 생성된 실체이며 클래스에서 정의한 속성과 메서드를 가진 독립적인 개체이다.

예시 : 필살기 - Explosion

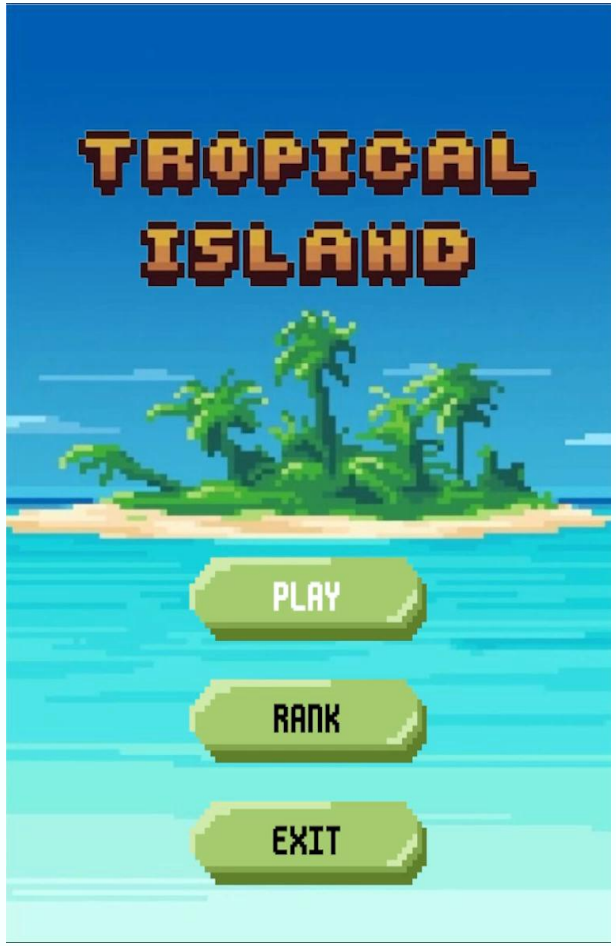
필살기 사용 시 나타나는 폭발을 클래스로 선언해 내부에 애니메이션을 메서드로 표현했다. 이 클래스를 다른 필살기 기능에서 explosion이라는 객체를 생성해 사용했다.

```python
class Explosion(pygame.sprite.Sprite): # 폭발 애니메이션 추가(김현수)
    def __init__(self, x, y, game):
        super().__init__()
        self.game = game

        # 폭발 이미지
        explosion_frames = [os.path.join(main.resources_path, f"effect_explosion_0{i}.png") for i in range(1,9)]

        # 이미지 로드 및 크기 조정
        self.frames = [pygame.image.load(img).convert_alpha() for img in explosion_frames]
        self.frames = [pygame.transform.scale(frame, (300, 300)) for frame in self.frames]

        self.image = self.frames[0]  # 첫 번째 이미지
        self.rect = self.image.get_rect(center=(x, y))

        self.current_frame = 0  # 현재 애니메이션 프레임
        self.last_update = pygame.time.get_ticks()  # 마지막 업데이트 시간
        self.frame_speed = 50  # 프레임 속도 (50ms마다 다음 프레임)

    def update(self):
        """💥 애니메이션 업데이트"""
        now = pygame.time.get_ticks()
        if now - self.last_update > self.frame_speed:
            self.last_update = now
            self.current_frame += 1

            if self.current_frame < len(self.frames):
                self.image = self.frames[self.current_frame]  # 다음 프레임
            else:
                self.kill()  # 애니메이션 끝나면 삭제
```

```python
explosion = Explosion(enemy.rect.centerx, enemy.rect.centery, self.game)  # 적 위치에서 폭발
self.game.all_sprites.add(explosion)  # 폭발 애니메이션 추가
enemy.hp-=50 # HP -50
if enemy.hp <= 0:
    enemy.kill()
    self.game.enemy_kill_count += 1
```


THANK
YOU

Demo



역할

신영민

기획서, 보고서 작성
게임 UI (메뉴, 게임 오버
캐릭터 HP, 필살기 게이지,
아이템 선택, 캐릭터 모션)
구현
이미지 수집 및 가공
발표

김민성

기본 UI (메뉴, 캐릭터 선
택) 기능 구현
랭킹 기능 구현
난이도 기능 구현
피격 판정 구현
이미지 수집 및 가공

김현수

캐릭터 HP 구현
필살기 게이지 구현
아이템 선택 구현
필살기 기능 구현
이미지 수집 및 가공

신지혜

게임정지 기능 구현
캐릭터 선택 기능 구현
캐릭터 HP 기능 구현
아이템 선택 기능 구현
캐릭터 모션 기능 구현
이미지 수집 및 가공

조준한

게임오버 기능 구현
랭킹 기능 구현
아이템 선택 기능 구현
피격 판정 구현
발사체, 몬스터 로직 구현
코드 통합 및 정리
이미지 수집 및 가공

소감

신영민

파이썬을 이용해서 본격적인 게임을 만드는 것은 처음이라 새로운 경험이었습니다. 게임을 만드는 데 필요한 것들이 생각보다 많다는 걸 알게 되었고 아이디어를 모아 게임이 완성되는 걸 보는게 즐거웠습니다.

김민성

이번 프로젝트를 통해 팀원들과 서로 의견을 주고받으며 협동하는 과정을 거치면서,어떻게 진행되는지 알게 됐고,결과물을 보며 성취감도 느낄 수 있었던 소중한 경험이었습니다.

김현수

파이썬으로 파이 게임을 처음 만들어보며 처음엔 어려웠지만 옆에 팀원들이 도와주고 제가 만든 코드가 게임에 적용되는 화면을 보며 나름 보람차고 컴퓨터 언어에 대한 재미를 붙일 수 있게 된 거 같습니다.

신지혜

서로 협동하고 이해하는 것이 생각했던 것보다 더 중요하다는 것을 체감하였습니다. 다음 팀 프로젝트에서는 더 팀원들과 좋은 시너지를 낼 수 있을 것 같다는 자신감이 생겼고 부족한 부분을 더 공부해서 완성도 높은 결과물을 내고 싶습니다.

조준한

프로젝트를 하며 팀원들과 역할을 나누어 협업하며 서로의 의견을 조율하는 과정은 뜻깊은 경험이었고, 함께 하나의 목표를 이루어가는 과정에서 많은 것을 배울 수 있었습니다.