보고서

---

1. 전체적인 코드 흐름

1.1 메인 실행 흐름 (main.py)

- 게임 루프 실행:
    - main.py 는 게임의 진입점으로, 무한 루프를 통해 게임의 전체 실행을 관리합니다.
- 게임 준비 상태 전환:
    - GameReady 객체를 생성하고 호출하여 메뉴 선택과 초기 게임 설정을 처리합니다.

1.2 게임 준비 (game_ready.py)

- GameReady 클래스 초기화:
    - 화면 크기 설정: SCREEN_WIDTH, SCREEN_HEIGHT 를 기준으로 화면 레이아웃을 초기화합니다.
    - 배경 이미지: BACKGROUND_INFO 에서 배경 정보를 가져와 게임 준비 화면에 표시합니다.
    - 폰트 및 텍스트 초기화: ImageFont 를 이용하여 텍스트 스타일과 위치를 정의합니다.
    - 레벨 및 배경 선택:
        - 사용자 입력에 따라 게임 레벨 및 배경을 변경할 수 있도록 준비합니다.

1.3 게임 실행 (game_starter.py)

- 초기화:
    - 플레이어 및 적 초기화:
        - Player 객체는 플레이어의 위치와 속성 정보를 기반으로 생성, Enemy 객체는 현재 단계(step)와 난이도(level)에 따라 생성됩니다.
- 게임 루프:
    - 플레이어 입력 처리:
        - 각 버튼 입력에 따라 플레이어 위치(L, R, U, D)와 행동(A 버튼)이 제어됩니다.
    - 적 생성 및 패턴 관리:
        - 일정 시간이 지나면 새로운 적이 스폰되며, 난이도에 따라 적의 수와 속도가 조정됩니다.
    - 충돌 관리:
        - 적과 플레이어 미사일의 충돌, 플레이어와 적 미사일의 충돌 등 다양한 충돌 이벤트를 처리합니다.
    - 게임 종료 조건:
        - 플레이어의 체력이 0 이 되거나 모든 적(특히 보스)이 처치되면 게임이 종료됩니다.

1.4 게임 종료 및 결과 처리

- 결과 표시 (game_starter.py):
    - 게임이 종료되면 승리 여부(Player Win)에 따라 결과 메시지("You Win!" 또는 "Game Over")를 표시합니다.
- 다시 시작 또는 종료 선택:
    - A 버튼 입력: 게임을 다시 시작합니다.
    - B 버튼 입력: 프로그램을 종료합니다.

---

2. 사용된 기술

2.1 사용된 주요 라이브러리와 모듈

프로젝트 전반에서 사용된 주요 라이브러리와 모듈들은 다음과 같습니다:

1) Pillow (PIL)

- 기능:
    - 이미지 로드, 크기 조정, 텍스트 렌더링, 이미지 합성 등을 처리합니다.
    - 배경 스크롤, UI 텍스트 표시, 객체 렌더링 등에 사용.
- 적용된 클래스/파일:
    - background.py: 배경 이미지 스크롤 및 텍스트 표시.
    - game_objects.py: 객체의 이미지 크기 조정 및 렌더링.
    - game_ready.py: 준비 화면의 텍스트와 이미지를 설정.

2) Adafruit RGB Display

- 기능:
    - ST7789 디스플레이와의 SPI 통신을 통해 게임 화면을 출력.
    - 하드웨어 디스플레이 초기화와 화면 렌더링에 사용.
- 적용된 파일:
    - settings.py: 디스플레이 핀 설정 및 초기화.
    - game_ready.py, background.py: 화면 출력에 사용.

3) DigitalIO

- 기능:
    - GPIO 핀을 통해 물리적 버튼 입력을 처리.
    - 각 버튼의 입력 상태(누름/떼기)를 읽어 게임 동작 제어.
- 적용된 파일:
    - settings.py: 버튼 핀 설정.
    - button.py: 각 버튼 상태를 읽어 속성으로 제공.

4) Colorsys

- 기능:
  - HSV(Hue, Saturation, Value)를 RGB 색상으로 변환하여 UI 텍스트 색상을 랜덤화.
- 적용된 파일:
  - background.py: UI 텍스트 표시 시 다양한 색상을 동적으로 설정.

5) Random

- 기능:
  - 무작위 수 생성에 사용.
  - 적의 위치, 아이템 생성 확률, 객체 ID 발급에 사용.
- 적용된 파일:
  - object_controller.py: 객체 ID 발급.
  - game_starter.py: 적의 위치 및 아이템 생성.
  - background.py: 텍스트 색상 랜덤화.

6) Time

- 기능:
  - 시간 기반 작업 처리(예: 적 스폰 타이밍, 공격 주기, 게임 루프).
- 적용된 파일:
  - game_starter.py: 적 생성 주기, 일시 정지 처리.
  - game_objects.py: 적의 공격 주기, 플레이어/적의 발사 간격 제어.

7) OS

- 기능:
  - 파일 경로 및 디렉터리 관리.
- 적용된 파일:
  - game_objects.py, background.py: 이미지 파일 경로 관리.

2.2 게임 설계의 특징

1. 모듈화:
   - 게임 준비, 실행, 객체 관리, 배경 처리 등 각 역할별로 파일과 클래스를 분리하여 관리.
   - 유지보수와 확장이 용이.
2. 확장 가능성:
   - 새로운 적, 무기, 효과 등을 추가할 때 OBJECT_INFO 에 데이터를 추가하고, 클래스 상속을 활용하면 쉽게 구현 가능.
3. 하드웨어 연동:
   - ST7789 디스플레이 및 GPIO 버튼을 통해 물리적 인터페이스와 소프트웨어를 통합.

---

3. 문제점과 해결책

3.1 배경 표시 문제

- 문제:
  - 배경 화면이 정적일 경우, 게임 진행이 단조로워 보이는 문제가 발생합니다.
  - 배경 이미지와 게임 객체(플레이어, 적, 미사일 등) 간의 자연스러운 시각적 통합이 부족합니다.
- 해결책:
  - 스크롤 배경 구현: background.py 에서 배경 이미지를 점진적으로 이동시키는 메서드를 추가합니다.
  - 배경과 객체 통합 렌더링: ObjectController 를 사용하여 배경 위에 모든 객체를 정확한 좌표에 렌더링.

3.2 적과 아군의 구분

- 문제:
  - 플레이어와 적 객체, 미사일 간의 충돌 및 상호작용이 구분되지 않거나 잘못된 객체가 처리되는 문제가 발생합니다.
- 해결책:
  - 객체 생성 시 team 속성을 명확히 지정합니다.('player', 'enemy', 'none')
  - ObjectController 에서 team 과 role 속성을 활용해 충돌 판단과 객체 제거를 체계적으로 관리합니다.

3.3 탄환 충돌 설정

- 문제:
  - 미사일과 전투기의 충돌 감지가 부정확하거나, 적 미사일끼리 충돌이 발생하는 문제가 관찰됩니다.
- 해결책:
  - ObjectController.__colision() 메서드를 사용하여 정확한 충돌 감지를 구현합니다.
  - 충돌 발생 시, 해당 미사일과 전투기를 삭제하거나 HP 를 감소시키는 로직을 추가합니다.

3.4 게임 상태 전환의 정확성

- 문제:
  - 게임 종료 후, 준비 상태로 정확히 전환되지 않거나 상태가 혼재되는 경우가 있습니다.
- 해결책:
  - GameStatus 클래스를 통해 명확한 상태 관리합니다.

3.5 하드웨어 입력 처리 문제

- 문제:
    - 버튼 입력 중복(버튼을 한 번 누른 후 여러 번 입력으로 처리되는 현상이 발생합니다.
- 해결책:
    - 버튼 입력 후 짧은 대기 시간을 두어 디바운싱 처리를 추가합니다.

3.6 적의 난이도 조정

- 문제:
    - 모든 적이 동일한 속도와 패턴을 사용하면 게임이 단조롭습니다.
- 해결책:
    - __set_enemy() 메서드를 통해 적의 종류, 이동 패턴(left, right, middle), 공격 주기를 조정합니다.
    - 보스 적(boss) 추가로 난이도를 점진적으로 증가시킵니다.

---

4. 코드 설명

4.1 main.py

- 역할:
    - 게임의 시작점으로, GameReady 와 GameStarter 클래스 간의 흐름을 관리합니다.
- 핵심 코드:

```
while True:
    if GameStatus.getGameReady():
        GameStatus.setGameReady(False)
        game_ready = GameReady()
        game_ready()
```

    - GameStatus.getGameReady(): 현재 게임 상태를 확인합니다.
    - GameReady 클래스 호출을 통해 준비 화면을 실행합니다.

4.2 settings.py

- 역할:
    - 하드웨어와 디스플레이 설정, 전역 상수를 정의합니다.
- 핵심 코드:

```
DISPLAY = st7789.ST7789(SPI, height=240, y_offset=80, rotation=180, ...)

BUTTON_A = DigitalInOut(board.D5)
```

BACKLIGHT.value = True

- ST7789 디스플레이와 버튼 핀을 초기화합니다.
- 화면 크기(SCREEN_WIDTH, SCREEN_HEIGHT)와 시작 좌표(START_POINT)를 설정합니다.

4.3 object_controller.py

- 역할:
  - 게임 내 객체(플레이어, 적, 미사일 등)의 생성, 관리, 삭제, 충돌 감지 등을 담당합니다.
- 핵심 코드:

```
@classmethod
def __colision(cls, object1, object2):
    if max(0, min(object1[2], object2[2]) - max(object1[0], object2[0])) * ...
```

  - 두 객체의 좌표가 겹치는지를 판단해 충돌 여부를 반환합니다.

```
@classmethod
def renew(cls):
    cls.__remove_objects(cls.__player_missile_ids,
cls.__player_missile_objects)
    cls.__remove_objects(cls.__enemy_ids, cls.__enemy_objects)
    ...
```

  - 화면 밖으로 나간 객체나 충돌한 객체를 삭제하고, 게임 루프 내 객체 상태를 갱신합니다.

4.4 game_status.py

- 역할:
  - 게임의 현재 상태(준비, 실행, 종료)를 관리합니다.
- 핵심 코드:

```
@classmethod
def start(cls):
    cls.__game_play = True
    cls.__game_text = ''
```

  - start()와 end() 메서드를 통해 게임 상태와 텍스트를 초기화하거나 설정합니다.

4.5 game_ready.py

- 역할:
    - 게임 준비 화면을 표시하고 사용자 입력을 대기합니다.
- 핵심 코드:

```
def __set_menu(self, up):
    self.__draw.rectangle((10, 10, self.__width - 10, ...))
    self.__set_text((20, self.__height // 2), 'Game Start')
```

    - 메뉴 UI 를 구성하고 텍스트를 렌더링합니다.

4.6 game_starter.py

- 역할:
    - 게임 실행의 메인 로직(적 생성, 충돌 처리, 게임 종료 조건 등)을 담당합니다.
- 핵심 코드:

```
def __call__(self):
    while True:
        if button.left and player.obj_coord[0] > 5:
            player.move('L')
        ...
        ObjectController.renew()
```

    - 사용자 입력에 따른 플레이어 동작을 제어합니다.
    - ObjectController.renew()를 호출하여 객체 상태를 업데이트합니다.

4.7 game_objects.py

- 역할:
    - 플레이어, 적, 미사일, 아이템 등 게임 객체를 정의합니다.
- 핵심 코드:

```
class Player(GameObject):
    def shoot(self):
        missile_coord = (self.obj_coord[0], self.obj_coord[1] - self.height // 2 -
5)
        Missile(missile_coord, ...)
```

    - 플레이어의 발사 동작을 구현합니다.
    - 객체별 행동(이동, 공격 등)을 클래스별로 정의합니다.

4.8 background.py

- 역할:
    - 배경 이미지의 스크롤 처리 및 게임 객체 렌더링을 담당합니다.
- 핵심 코드:

```
def __get_image(self):
    cropped_image1 = image.crop((0, self.__crop_point, self.width,
self.height))
    cropped_image2 = image.crop((0, 0, self.width, self.__crop_point))
    ...
```

    - 배경 스크롤 효과를 구현합니다.
    - 배경 이미지 위에 객체를 합성하여 표시합니다.

4.9 button.py

- 역할:
    - 물리적 버튼의 입력 상태를 관리합니다.
- 핵심 코드:

```
@property
def left(self):
    return False if self.__left.value else True
```

    - 버튼 입력을 속성 형태로 제공하여 사용자 입력을 확인합니다.