Project #2 Design & Develop Your Own Game

컴퓨터공학과 2019102212 이정호

1. Goal

이번에 제작할 게임의 모티브가 되는 게임은 '스크램블, 뱅가드' 등의 횡 스크롤 슈팅 게임이다. '갤러그, 스페이스 인베이더' 등의 종 스크롤 슈팅 게임은 게임 이용자가 조종하는 맛이 너무 없을 것 같고, 다른 전방위 슈팅 게임 등은 만들기에 시간이 부족할 것 같아서 횡 스크롤 슈팅 게임을 선택했다. 횡 스크롤 슈팅 게임에 대해 간단히 얘기하자면, 비행기는 상하좌우 어느 곳으로든 움직일 수 있고 왼쪽으로 진행하는 배경에 의한 착시 현상으로 전반적으로 오른쪽으로 이동하는 듯한 느낌이 든다. 예외도 있지만 일반적으로 사격은 알아서 하되 오른쪽으로만 사격하며 적도 오른쪽에서만 등장한다.

그림 1 스크램블 - 횡 스크롤

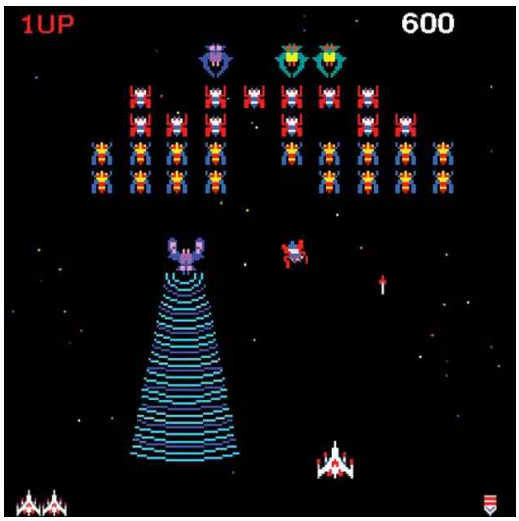


그림 2 갤러그 - 종 스크롤

2. Game Design & Structure

이 문단에서는 이번에 제작할 게임의 전반적인 구현 목표와 목표를 구현하기 위해서 어떤 식으로 코드를 구성할 것인지에 대해서 간단한 설명을 하게 될 것이다. 먼저 전반적인 구현 목표에 대해서 살펴보자. 크게 배경과 객체로 나눠지게 된다. 먼저 배경의 구현에 대한 간단히 살펴보자면, 대부분의 횡 스크롤 게임은 화면에 게임 이용자 객체는 가만히 두고, 배경을 진행 방향의 반대 방향으로 움직이게끔 해서 착시 현상을 이용하여 객체가 진행 방향으로 간다고 느끼도록 구현한다. 게임 이용자 객체의 고정 진행 방향이 오른쪽이므로 배경이 왼쪽으로 계속해서 이동하도록 만들면

의도한 착시 현상을 일으킬 수 있다.

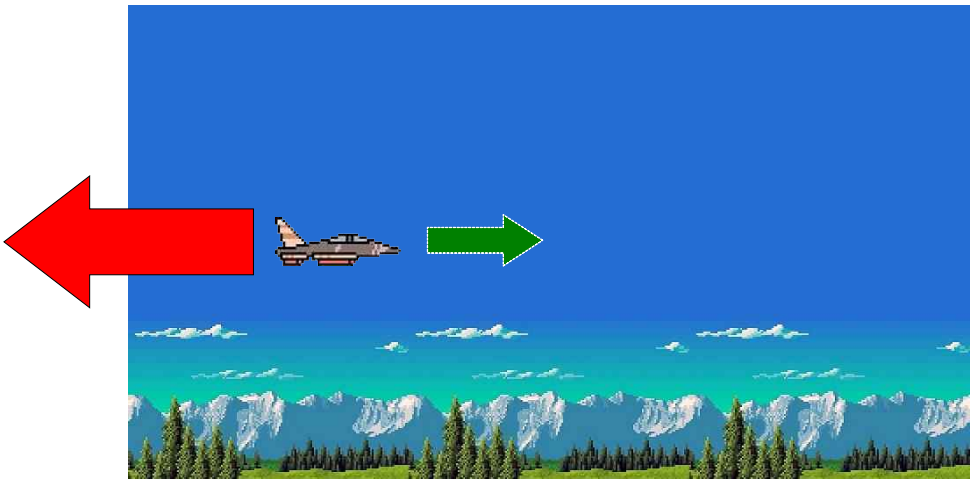


그림 3 배경을 진행 방향의 반대쪽으로 이동시켜 착시 유도

다음은 객체, 스프라이트들에 대한 설명이다. 스프라이트들은 크게 게임 이용자 스프라이트 (aircraft), 총알 스프라이트 (bullet), 적 스프라이트 (helicopter), 3가지로 나눌 수 있다. aircraft 같은 경우에는 게임 이용자가 제어하는 스프라이트이기 때문에, 키보드 입력을 받아서 입력을 처리하여 aircraft 움직임 벡터를 조정하는 방식으로 이동을 제어한다. bullet 같은 경우에는 게임 이용자가 필요할 때 수동적으로 발사하여 적을 처치할 수 있게끔 한다. 마지막으로 helicopter 같은 경우에는 사전에 정해둔 코드에 의해 자동으로 움직이는 스프라이트로 aircraft, bullet 간의 충돌을 판정하고 상황에 맞게 충돌 결과를 처리해서 게임 화면에 구현한다.

다음으로 인터페이스에 관한 내용이다. 인터페이스에 표현될 내용은 크게 점수, 난이도, 아이템 레벨, 연속 처치 등이 될 것이며, 해당 내용은 화면의 왼쪽 위에 표시되어 게임 이용자가 볼 수 있도록 구현할 것이다.

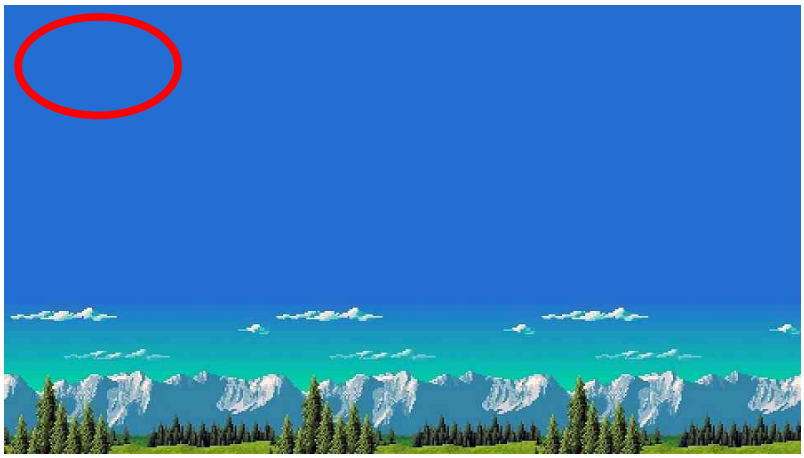
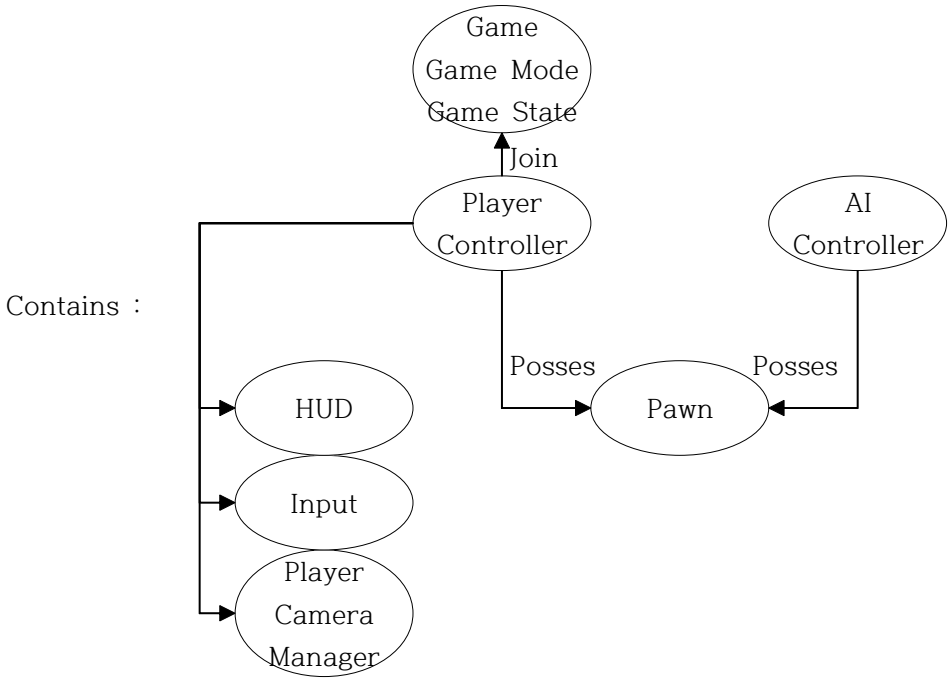


그림 4 화면 왼쪽 위에 현재 게임 이용자의 상태가 표시된다.

게임의 전반적인 로직을 다이어그램으로 표현하면 다음과 같다.



먼저 Game, Game Mode, Game State 부분에서는 게임의 승리 (패배) 규칙과 현재 상태에 대해서 제어한다. 게임의 승리 (패배) 규칙은 간단하다. 일단 무한하게 점수를 쌓고 최고 점수를 비교하는 스코어링 게임의 특성상 이 게임은 승리 조건이 존재하지 않는다. 대신 패배 규칙이 존재하여 점수 쌓기에 방해를 가하는 방식이다. 이 게임의 패배 규칙은 aircraft가 helicopter, missile 등에 충돌하면 패배한다. Player Controller는 게임 이용자가 제어하게 될 객체 그 자체로 aircraft라고 생각하면 된다. AI Controller는 사전에 코드로 작성된 대로 컴퓨터에 의해 자동으로 제어하게 될 객체로 helicopter, missile 등을 나타낸다. Pawn은 Input을 받아 aircraft, helicopter, missile의 움직임을 총괄하는 객체로 객체들의 움직임을 처리한다. HUD는 인터페이스로 현재 상태를 수치화하여 텍스트로 게임 이용자에게 보여주게 된다. Player Camera Manager 같은 경우 착시 현상을 이용하여 사실상 aircraft는 계속해서 한 화면에 머물러 있어 화면 자체를 구현하는 것 외에 추가적인 구현이 필요하지는 않다.

4. Code Description

```python
import pygame as P

def runGame():
    global screen, clock

    crashed = False

    while not crashed:
        for event in P.event.get():
            if event.type == P.QUIT:
                crashed = True

        screen.fill((255, 255, 255))
        P.display.update()
        clock.tick(60)

    P.quit()

def initGame():
    global screen, clock

    P.init()
    P.display.set_caption('2019102212_Game')
    screen = P.display.set_mode((1920, 1080))

    clock = P.time.Clock()
    runGame()

initGame()
```

가장 기본이 될 화면과 창을 구현하는 코드이다. 60의 프레임 제한을 걸어 무한 루프를 돌린다. 창은 X 표시를 눌렀을 때만 종료되게 된다.

```python
background_image_1_x -= 5
background_image_2_x -= 5

if background_image_1_x == -background_width:
    background_image_1_x = background_width

if background_image_2_x == -background_width:
    background_image_2_x = background_width
```

진행 방향의 반대로 이동하여 착시 현상을 일으키는 코드이다. 이미지를 2장 이용해서 연속적으로 둔다.

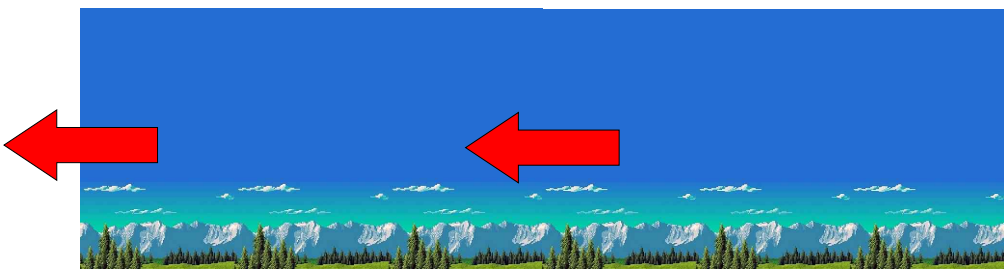


그림 8 이미지 2장을 이용하여 구현한다.

뒤의 이미지가 화면의 왼쪽 끝에 도달하면 앞의 이미지를 뒤의 이미지 뒤로 옮겨 무한하게 반복되도록 구현한다.

```
if event.type == P.KEYDOWN:
    if event.key == P.K_LEFT:
        aircraft_x_acceleration = -.5
    elif event.key == P.K_RIGHT:
        aircraft_x_acceleration = .5
    elif event.key == P.K_UP:
        aircraft_y_acceleration = -.5
    elif event.key == P.K_DOWN:
        aircraft_y_acceleration = .5
    elif event.key == P.K_LCTRL:
        bullet_x = aircraft_x + 100
        bullet_y = aircraft_y + 20
        bullet_list.append([bullet_x, bullet_y, bullet_remove])

if event.type == P.KEYUP:
    if event.key == P.K_LEFT or event.key == P.K_RIGHT:
        aircraft_x_acceleration = 0
    elif event.key == P.K_UP or event.key == P.K_DOWN:
        aircraft_y_acceleration = 0
```

aircraft의 제어는 방향키와 왼쪽 컨트롤 키를 이용해서 한다. aircraft는 기본적으로 velocity와 acceleration을 통해서 위치가 제어되기 때문에 방향키로는 acceleration만 제어하게 된다. 방향키를 누르면 방향키의 방향으로 0.5만큼의 acceleration을 준다. 방향키를 떼면 방향키의 축에 맞는 acceleration을 다시 0으로 바꾼다.

```
aircraft_x_move += aircraft_x_acceleration
aircraft_y_move += aircraft_y_acceleration

if abs(aircraft_x_move) >= 7:
    aircraft_x_move = aircraft_x_move/abs(aircraft_x_move) * 7
if abs(aircraft_y_move) >= 7:
    aircraft_y_move = aircraft_y_move/abs(aircraft_y_move) * 7

if aircraft_x_acceleration == 0:
    aircraft_x_move *= 0.9
if aircraft_y_acceleration == 0:
    aircraft_y_move *= 0.9

if aircraft_x < 0:
    aircraft_x = 0
elif aircraft_x > background_width * 0.9:
    aircraft_x = background_width * 0.9

if aircraft_y < 0:
    aircraft_y = 0
elif aircraft_y > background_height * 0.7:
    aircraft_y = background_height * 0.7

aircraft_x += aircraft_x_move
aircraft_y += aircraft_y_move
```

aircraft의 최고 속도는 7을 넘을 수 없도록 제한한다. 만일 방향키 입력이 없다면 velocity에 0.9를 계속해서 곱해 최종적으로는 멈추도록 한다. aircraft가 지정된 범위 바깥으로 나갈 수 없도록 위치를 제한한다.

```
helicopter_1_move += 0.1
helicopter_1_x -= helicopter_1_move

if helicopter_1_x <= -50:
    helicopter_1_x = background_width
    helicopter_1_y = R.uniform(0, background_height * 0.7)
    helicopter_1_move = 3
    combo = 0

if score_count >= 10000:
    level_count = 2
    helicopter_2_move += 0.2
    helicopter_2_x -= helicopter_2_move

    if helicopter_2_x <= -50:
        helicopter_2_x = background_width
        helicopter_2_y = R.uniform(0, background_height * 0.7)
        helicopter_2_move = 5
        combo = 0
```

게임에 등장하는 적은 총 3가지이다. helicopter 1, helicopter 2, missile이다. helicopter 1은 가장 처음부터 등장하는 level 1의 적이고, 시작 velocity는 3, acceleration은 0.1로 정했다. helicopter 2는 score 10,000점을 달성한 이후로 등장하는 level 2의 적이고, 시작 velocity는 5, acceleration은 0.2로 정했다. 이 두 적은 왼쪽 끝에 도달하게 되면 다시 오른쪽 끝으로 옮겨지고 velocity, combo가 초기화된다. 그리고 y축으로는 지정된 랜덤한 범위 내로 옮겨진다.

```
if score_count >= 30000:
    level_count = 3
    missile_move += 0.5
    missile_x -= missile_move

if score_count >= 50000:
    item_count = 3

    if missile_x <= -5000:
        missile_x = background_width
        missile_y = R.uniform(0, background_height * 0.7)
        missile_move = 3
```

missile은 score 30,000점을 달성한 이후로 등장하는 level 3의 적이고, 시작 velocity는 3, acceleration은 0.5로 정했다. missile은 helicopter 들에 비해 훨씬 빠른 속도로 날아오는 대신 x 위치 초기화 조건을 -5,000으로 길게 잡았기 때문에 출현 빈도가 낮다.

```
if len(bullet_list) != 0:
    for i, bullet in enumerate(bullet_list):
        bullet[0] += 30
        bullet_list[i][0] = bullet[0]

        if bullet[0] > helicopter_1_x:
            if item_count == 1:
                if bullet[1] > helicopter_1_y and bullet[1] < helicopter_1_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_1 = True
                    combo += 1

            elif item_count == 2:
                if bullet[1] > helicopter_1_y and bullet[1] < helicopter_1_y + helicopter_height or\
                   bullet[1] + 30 > helicopter_1_y and bullet[1] + 30 < helicopter_1_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_1 = True
                    combo += 1

            elif item_count == 3:
                if bullet[1] > helicopter_1_y and bullet[1] < helicopter_1_y + helicopter_height or\
                   bullet[1] + 30 > helicopter_1_y and bullet[1] + 30 < helicopter_1_y + helicopter_height or\
                   bullet[1] - 30 > helicopter_1_y and bullet[1] - 30 < helicopter_1_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_1 = True
                    combo += 1

        if bullet[0] > helicopter_2_x:
            if item_count == 1:
                if bullet[1] > helicopter_2_y and bullet[1] < helicopter_2_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_2 = True
                    combo += 1

            elif item_count == 2:
                if bullet[1] > helicopter_2_y and bullet[1] < helicopter_2_y + helicopter_height or\
                   bullet[1] + 30 > helicopter_2_y and bullet[1] + 30 < helicopter_2_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_2 = True
                    combo += 1

            elif item_count == 3:
                if bullet[1] > helicopter_2_y and bullet[1] < helicopter_2_y + helicopter_height or\
                   bullet[1] + 30 > helicopter_2_y and bullet[1] + 30 < helicopter_2_y + helicopter_height or\
                   bullet[1] - 30 > helicopter_2_y and bullet[1] - 30 < helicopter_2_y + helicopter_height:
                    score_count += 500 * (combo + 1)
                    bullet_list[i][2] = True
                    shoot_helicopter_2 = True
                    combo += 1

        if bullet[0] >= background_width:
            try:
                bullet_list[i][2] = True
            except:
                pass

    for bullet in bullet_list:
        if bullet[2]:
            bullet_list.remove(bullet)
```

다음으로 bullet 객체이다. bullet 객체는 기본적으로 게임 이용자가 왼쪽 컨트롤 키를 누른 시점에 aircraft 앞 적절한 위치에 생성되어 좌표를 bullet_list에 저장한다. score가 올라감에 따라 item_level도 올라가도록 설정했는데, 처음에는 item_level이 1이었다가 20,000점을 넘기면 item_level을 2, 50,000점을 넘기면 item_level이 3이 되도록 설정했다. item_level에 따라서 한 번 bullet을 발사했을 때 날아가는 총알의 개수가 달라진다. item_level이 1이면 1발, 2면 2발, 3이면 3발이 발사된다. 이에 따

라서 객체를 새로 생성하는 것이 아닌, 시각적으로 2발, 3발로 보이도록 구현했다. bullet의 발사를 이런 식으로 구현했기 때문에 item_level에 따라 bullet과 적의 충돌 판정이 달라지도록 구현했다. item_level이 올라갈수록 bullet의 개수는 y축 방향으로 많아지기 때문에, bullet 사이를 촘촘하게 해서 가장 위의 bullet과 가장 아래의 bullet 사이에는 충돌 판정이 세는 부분이 없도록 설정했다. bullet 객체는 helicopter 들과 충돌하거나 화면 오른쪽 끝에 도달하면 삭제한다. missile 객체는 기본적으로 bullet과 충돌한다고 삭제되지 않는다.

기본적으로 충돌했다고 판정된 객체들은 삭제해야 한다. 하지만 충돌 판정이 되었을 때 객체를 바로 삭제하면 out of range 오류가 발생할 것이다. 예시를 들어보면, 한 tick에서 만일 bullet 1개가 helicopter 2개와 동시에 충돌 판정이 나게 되면, bullet 객체를 2개 삭제해야 한다. 하지만 bullet 객체는 1개밖에 없었으므로, 1번 삭제하고 나면, 2번째 삭제 명령은 -1 index를 가리키게 되어 out of range 오류가 발생할 것이다. 따라서 bullet 객체에 bullet_remove라는 새로운 attribute를 추가하여 false로 초기화한다. 충돌이 발생 해당 bullet의 bullet_remove attribute를 true로 바꾸고, 이후에 bullet_remove가 true인 bullet 들만 일괄적으로 삭제하면 out of range 오류가 더 이상 발생하지 않게 된다.

```
if not shoot_helicopter_1:
    screen.blit(helicopter_image_1, (helicopter_1_x, helicopter_1_y))
else:
    screen.blit(shoot_image, (helicopter_1_x, helicopter_1_y))
    shoot_helicopter_1_count += 1
    if shoot_helicopter_1_count > 5:
        shoot_helicopter_1_count = 0
        helicopter_1_x = background_width
        helicopter_1_y = R.uniform(0, background_height * 0.3)
        shoot_helicopter_1 = False

if not shoot_helicopter_2:
    screen.blit(helicopter_image_2, (helicopter_2_x, helicopter_2_y))
else:
    screen.blit(shoot_image, (helicopter_2_x, helicopter_2_y))
    shoot_helicopter_2_count += 1
    if shoot_helicopter_2_count > 5:
        shoot_helicopter_2_count = 0
        helicopter_2_x = background_width
        helicopter_2_y = R.uniform(0, background_height * 0.3)
        shoot_helicopter_2 = False
```

bullet 객체와 helicopter 객체가 충돌하게 되면, helicopter 객체의 이미지를 폭발하는 이미지로 바꿔 충돌했다는 사실을 시각적으로 게임 이용자에게 알려줘야 한다. helicopter 객체는 일정 시간 동안 폭발하는 이미지로 바뀌어 비행하다가 다시 위치를 초기화하여 화면 오른쪽 끝에서 등장한다. 하지만 combo 시스템을 활용하기 위해 velocity는 초기화되지 않고, 빠른 속도로 aircraft를 향해 돌진한다.

```
if aircraft_x + aircraft_width > helicopter_1_x and aircraft_x < helicopter_1_x + helicopter_width:
    if (aircraft_y > helicopter_1_y and aircraft_y < helicopter_1_y + helicopter_height) or\
       (aircraft_y + aircraft_height > helicopter_1_y and aircraft_y + aircraft_height < helicopter_1_y + helicopter_height):
        gameover('SCORE : ' + str(score_count), 480, gameover_image)
        gameover('GAME START!', 200, background_image_1)
        score_count = 0
        runGame()

if aircraft_x + aircraft_width > helicopter_2_x and aircraft_x < helicopter_2_x + helicopter_width:
    if (aircraft_y > helicopter_2_y and aircraft_y < helicopter_2_y + helicopter_height) or\
       (aircraft_y + aircraft_height > helicopter_2_y and aircraft_y + aircraft_height < helicopter_2_y + helicopter_height):
        gameover('SCORE : ' + str(score_count), 480, gameover_image)
        gameover('GAME START!', 200, background_image_1)
        score_count = 0
        runGame()

if aircraft_x + aircraft_width > missile_x and aircraft_x < missile_x + missile_width:
    if (aircraft_y > missile_y and aircraft_y < missile_y + missile_height) or\
       (aircraft_y + aircraft_height > missile_y and aircraft_y + aircraft_height < missile_y + missile_height):
        gameover('SCORE : ' + str(score_count), 480, gameover_image)
        gameover('GAME START!', 200, background_image_1)
        score_count = 0
        runGame()
```

마지막으로 게임 종료 화면이다. aircraft가 helicopter, missile과 충돌하게 되면 Game State가 Game Over로 전환되며, 2초간 score를 보여준 뒤 Game State를 다시 전환하여 게임 이용자가 게임을 끄기 전까지 게임이 계속해서 반복되도록 설정했다. 킬링타임용 게임을 하면서 항상 느꼈던 점이 게임의 재시작에 시간이 오래 걸리면 불편함이 많이 느껴졌었다. 따라서 한 게임 한 게임 간의 간격을 최대한 줄임으로써 흐름이 끊기지 않고 계속 게임을 즐길 수 있도록 유도했다.

5. Execution Environment

버전은 2.5.2이다. 파이 게임을 제외한 모듈은 기본 모듈들밖에 없어서 부가적으로 설치해야 하거나 하는 사항은 없다. 단순히 실행 파일을 실행하기만 하면 된다.