Project #3 Develop Your Own Game Engine (Report)

2020105651 이호준

Goal

Pygame을 기초로 하여 게임 엔진을 제작한다. 완벽한 게임 엔진을 만드는 것이 아닌 관심있고 구현해 보고 싶던 기능들을 이해한 뒤 개발, 프로그래밍 해보는 경험에 초점을 둔다. 각 기능들의 작동을 확인할 수 있는 테스트 코드와 함께 작성한다. 라이브러리와 패키지들을 사용하는 것에 제한은 없으나 이미 존재하는 기능들을 불러와 실행만 하는 것은 지양한다. 가능한 처음부터 기능들을 직접 구현하거나 기존 기능들을 응용하여 추가적인 기능들을 개발하는 경험 중점으로 프로젝트를 진행한다.

Game Engine Design & Structure

게임 엔진을 제작하며 처음의 생각으로는 한 가지 제작하고 싶은 게임을 떠올린 뒤 필요한 기능을 제작해 보는 것이었다. 하지만 첫째로 프로젝트 시작 당시에 측정 장르의 게임이 떠오르지 않았으며 고민을 하여도 각 게임들의 기능이 각자 개발될 것인지 어떤 기능들은 게임 엔진으로 구현해야 할지에 대한 기준이 없었다. 그리하여 우선적으로 프로젝트 공지에 포함된 구현 기능들의 예시를 보고 하나씩 구현해보고자 하였다.

우선 프로젝트 폴더 안에 각 기능들을 분류하기 위해 폴더를 생성하였다. 소리 관련 기능의 Audio, 제작 관련 기능의 Design, 그래픽 관련 기능의 Graphics, 입력과 출력 관련 기능의 IO, 수학적 라이브러리를 대체할 Math, 물리 관련 기능의 Physics 폴더로 분류하였다. 추가로 게임 엔진이라면 범용적으로 활용하기 좋은 소리 샘플들이 필요하다고 생각하여 Audio_Samples 폴더를 생성하여 로열티 프리 효과음들을 저장해 두었다.

해당 프로젝트의 게임 엔진은 2D 게임 엔진으로 모든 기능이 2차원을 상정하고 작성되었다.

Audio 폴더에는 2차원 패닝을 적용하는 Spatial_Audio.py, 소리의 발생 위치에 따라 스테레오를 활용하여 정보를 주기 위한 X_axis_Audio.py가 있다.

Design 폴더에는 간단히 GUI적으로 타일들을 배치하고 해당 정보를 텍스트 형식으로 내보내는 Make-tile.py가 있다.

Graphics 폴더에는 메쉬들의 크기와 위치를 변경하는 Mesh_Editing_Tool.py, 오브젝트 파일을 읽어오는 obj_Reader.py 파일이 있다.

IO 폴더에는 컨트롤러 입력을 처리하는 DualSense.py와 XBOX_input.py, 모니터 화면의 정보를 클래스 형식으로 담는 MonitorInfo.py가 있다.

Math 폴더에는 미분을 담당하는 Numerical_derivative.py, 정적분을 담당하는 Numerical_integral.py, 삼각함수 관련 계산을 담당하는 Trigonometric.py, 벡터의 내적과 외적을 담당하는 VectorOperations.py가 있다.

가장 중요한 Physics 폴더에는 물체의 충돌을 계산하는 Collision_Detection.py, 게임에서 활용가능한 파티클을 생성해주는 Particle_System.py, 리지드바디 관련 기능을 제공하는 Rigid_Body_Dynamics.py가 있다.

최상위 디렉토리에는 간단한 플랫포머 게임의 씬인 main.py가 있다. 또한 각각의 코드들마다 디버깅 또는 활용 방법의 이해를 돕기 위한 테스트 코드가 포함되어 있다.

Feature Description & Code Description

코드 설명은 가능한 주석으로 대체한다.

1. Audio

1-1. Spatial_Audio.py

Spatial_Audio.py는 오디오에 2차원 패닝을 적용한다. 좌우 음향 채널의 음량비율을 조정하여 입체음향의 효과를 내는 것이다. apply_2d_panning를 각도와 함께 실행시키면 코사인 함수를 통해 감쇠 계수를 계산, 좌우 채널에 적용한 뒤 각 채널의 값을 반환한다.

```
import numpy as np
import librosa

class SpatialAudio:
    def __init__(self, num_channels):
        # 초기화 메서드
        self.num_channels = num_channels
        self.audio_data = None

    def load_audio_data(self, audio_file):
        # 오디오 데이터를 불러오는 메서드
        self.audio_data, _ = librosa.load(audio_file, sr=None, mono=True)

    def apply_2d_panning(self, azimuth_angle):
        # 2 차원 패닝을 적용하는 메서드
        # 입력 오디오 데이터의 길이를 확인
        num_samples = len(self.audio_data)
```

```
        # 좌우 채널을 저장할 배열을 생성
        left_channel = np.zeros(num_samples)
        right_channel = np.zeros(num_samples)

        # 각도를 라디안으로 변환
        theta = np.radians(azimuth_angle)

        # 각도에 따른 감쇠 계수를 계산
        attenuation = np.cos(theta)

        # 좌우 채널에 각각 감쇠 계수를 적용
        left_channel = self.audio_data * attenuation
        right_channel = self.audio_data * -attenuation

        return left_channel, right_channel

# 테스트 코드
if __name__ == "__main__":
    # SpatialAudio 클래스 객체 생성
    spatial_audio = SpatialAudio(num_channels=2)

    # 오디오 데이터 로딩
    spatial_audio.load_audio_data(audio_file="Audio_Samples\start-race-8-
bit.mp3")

    # 2 차원 패닝 적용
    left_channel, right_channel =
spatial_audio.apply_2d_panning(azimuth_angle=30)
    # 패닝된 결과를 출력
    print("left_channel:", left_channel)
    print("right_channel:", right_channel)

    # 2 차원 패닝 적용
    left_channel, right_channel =
spatial_audio.apply_2d_panning(azimuth_angle=60)
    # 패닝된 결과를 출력
    print("left_channel:", left_channel)
    print("right_channel:", right_channel)
```

1-2. X_axis_Audio.py

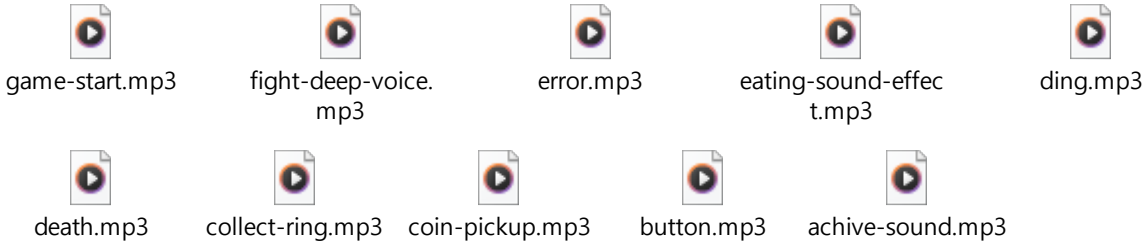
X_axis_Audio.py 또한 입체 음향을 표현하기 위한 기능이다. play_audio_with_stereo 함수를 소리 파일, x 좌표, 화면 너비 총 세 파라미터로 호출하면 입체 음향을 재생한다. 입력된 x값이 0에 가까울수록 좌측 채널의 소리가, 화면 너비에 가까울수록 우측 채널의 소리가 증가한다. 따라서 화면 속 소리의 위치에 따라 사용자는 편향된 소리를 듣게 된다.

```
import pygame
```

```python
def play_audio_with_stereo(audio_file_path, x_axis_value=960,
screen_width=1920):
    # 초기화
    pygame.init()
    pygame.mixer.init()

    # 오디오 파일 로드
    audio = pygame.mixer.Sound(audio_file_path)

    # 스테레오 오디오 계산
    pan = x_axis_value / screen_width

    channel = pygame.mixer.Channel(0)
    channel.set_volume(1 - pan, pan)

    # 오디오 재생
    channel.play(audio)

if __name__ == "__main__":
    play_audio_with_stereo(audio_file_path="Audio_Samples\ding.mp3",
x_axis_value=100)

    # 오디오 재생이 끝날 때까지 대기
    while pygame.mixer.get_busy():
        pass

    play_audio_with_stereo(audio_file_path="Audio_Samples\ding.mp3",
x_axis_value=1500)

    # 오디오 재생이 끝날 때까지 대기
    while pygame.mixer.get_busy():
        pass

    pygame.quit()
```

2. Audio_Samples

준비된 오디오 파일의 목록은 다음과 같다.



3. Design

3-1. Make-tile.py

Make-tile.py는 라이브러리나 함수가 아닌 독립적인 프로그램이다. 실행 시 기존에 설정한 창의 너비, 높이와 타일의 크기에 따라 새로운 창이 열리며 해당 창에서 마우스 클릭을 통해 해당 타일을 검은색(1)으로 채우거나 흰색(0)으로 비울 수 있으며 스페이스바로 종료하며 생성한 맵을 텍스트 파일로 저장한다. Maps 폴더안의 해당 파일의 형식은 아래와 같다.

00000000000000000000000000000000000000
01111111111111111111111111111111111100
01000000000000000000000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000011000000000000000100
01000000000000000000000000000000000100

```python
import pygame

# 초기화
pygame.init()

# 화면 크기 및 제목 설정
SCREEN_WIDTH, SCREEN_HEIGHT = pygame.display.Info().current_w,
pygame.display.Info().current_h
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("맵 편집기")

# 색깔 정의
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# 타일 크기 및 색상
TILE_SIZE = 50

# 맵 데이터
map_data = []
for i in range((SCREEN_HEIGHT // TILE_SIZE) + 1):
    map_data.append([0] * ((SCREEN_WIDTH // TILE_SIZE) + 1))

# 메인 루프
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            # 마우스 클릭 위치를 타일 좌표로 변환
            mouse_x, mouse_y = pygame.mouse.get_pos()
            tile_x = mouse_x // TILE_SIZE
            tile_y = mouse_y // TILE_SIZE

            # 왼쪽 버튼 클릭시 벽 생성, 오른쪽 버튼 클릭시 빈 공간 생성
            if event.button == 1:
                map_data[tile_y][tile_x] = 1
            elif event.button == 3:
                map_data[tile_y][tile_x] = 0
        elif event.type == pygame.KEYDOWN:
            # 스페이스바를 누르면 종료
            if event.key == pygame.K_SPACE:
                running = False

    # 화면 초기화
    screen.fill(WHITE)
```

```
    # 맵 그리기
    for y, row in enumerate(map_data):
        for x, tile in enumerate(row):
            if tile == 1:
                pygame.draw.rect(screen, BLACK, (x * TILE_SIZE, y * TILE_SIZE,
TILE_SIZE, TILE_SIZE))

    # 화면 업데이트
    pygame.display.flip()

# 맵 데이터 저장
with open("Design\Maps\map_data.txt", "w") as f:
    for row in map_data:
        for tile in row:
            f.write(str(tile))
        f.write("\n")

# 게임 종료
pygame.quit()
```

4. Graphics

4-1. Mesh_Editing_Tool.py

Mesh_Editing_Tool.py는 메시를 이동, 스칼라배, 회전, 변형시키는 함수들을 제공한다. 덧셈을 이용한 이동, 곱셈을 이용한 스칼래배, 삼각함수를 이용한 회전과 함께 사용자 지정 함수가 제공되면 그를 사용하여 메시를 변형시킨다.

```
import math

class MeshEditingTool:
    def __init__(self, mesh):
        self.mesh = mesh

    def translate(self, translation_vector):
        # 주어진 벡터만큼 메시의 각 꼭짓점을 이동
        for i in range(len(self.mesh.vertices)):
            for j in range(2):
                self.mesh.vertices[i][j] += translation_vector[j]

    def scale(self, scale_factor):
        # 주어진 스케일 비율로 메시의 각 꼭짓점을 조정
        for i in range(len(self.mesh.vertices)):
            for j in range(2):
                self.mesh.vertices[i][j] *= scale_factor

    def rotate(self, rotation_angle):
```

```
        # 주어진 회전 각도로 메시의 각 꼭짓점을 회전
        # 간단한 2D 회전 행렬 사용
        for i in range(len(self.mesh.vertices)):
            x = self.mesh.vertices[i][0]
            y = self.mesh.vertices[i][1]
            new_x = x * math.cos(rotation_angle) - y *
math.sin(rotation_angle)
            new_y = x * math.sin(rotation_angle) + y *
math.cos(rotation_angle)
            self.mesh.vertices[i] = [new_x, new_y]

    def apply_deformation(self, deformation_function):
        # 사용자 정의 변형 함수를 메시에 적용
        self.mesh.vertices = deformation_function(self.mesh.vertices)

# 테스트 코드
class Mesh:
    def __init__(self, vertices):
        self.vertices = vertices

# MeshEditingTool 클래스와 간단한 Mesh 클래스 정의 후에 테스트 코드 작성
if __name__ == "__main__":
    # 테스트할 Mesh 객체 생성
    initial_vertices = [[1, 1], [2, 2], [3, 3]]
    test_mesh = Mesh(initial_vertices)

    # MeshEditingTool 객체 생성
    mesh_editor = MeshEditingTool(test_mesh)
    print("Mesh vertices:", test_mesh.vertices)

    # 이동 테스트
    translation_vector = [1, 2]
    mesh_editor.translate(translation_vector)
    print("이동 후 Mesh vertices:", test_mesh.vertices)

    # 스케일 테스트
    scale_factor = 2
    mesh_editor.scale(scale_factor)
    print("스케일 후 Mesh vertices:", test_mesh.vertices)

    # 회전 테스트
    rotation_angle = math.radians(45)  # 45 도를 라디안으로 변환
    mesh_editor.rotate(rotation_angle)
    print("회전 후 Mesh vertices:", test_mesh.vertices)
    mesh_editor.rotate(rotation_angle)
    print("2 번째 회전 후 Mesh vertices:", test_mesh.vertices)

    # 변형 함수 적용 테스트 (임의의 변형 함수 예시)
```

```
    def deformation_function(vertices):
        for i in range(len(vertices)):
            for j in range(2):
                vertices[i][j] = vertices[i][j] ** 2  # 각 좌표값을 제곱
        return vertices

    mesh_editor.apply_deformation(deformation_function)
    print("변형 함수 적용 후 Mesh vertices:", test_mesh.vertices)
```

4-2. obj_Reader.py

obj_Reader.py는 obj파일을 받아 정점 데이터와 면 데이터를 추출하는 기능을 제공한다. 텍스트로 이루어져 있는 obj를 적절하게 끊어 데이터들을 추출한다. 테스트에 활용된 예시 파일은 아래와 같다.

v 0.0 0.0

v 1.0 0.0

v 1.0 1.0

v 0.0 1.0

f 1 2 3

f 1 3 4

```
class ObjReader:
    def __init__(self, file_path):
        self.file_path = file_path
        self.vertices = []
        self.faces = []

    def load_obj(self):
        with open(self.file_path, 'r') as file:
            for line in file:
                if line.startswith('v '):
                    # 정점 데이터를 추출합니다.
                    vertex = list(map(float, line[2:].split()[:2]))  # 2D
객체이므로 x, y 좌표만 고려
                    self.vertices.append(vertex)
                elif line.startswith('f '):
                    # 면 데이터를 추출합니다.
                    face = [int(vertex.split('/')[0]) for vertex in
line[2:].split()]
                    self.faces.append(face)

    def get_vertices(self):
        return self.vertices
```

```
    def get_faces(self):
        return self.faces

# 테스트 코드
if __name__ == "__main__":
    obj_reader = ObjReader("Graphics\ex.obj")
    obj_reader.load_obj()

    vertices = obj_reader.get_vertices()
    faces = obj_reader.get_faces()

    print("2D 정점 좌표:")
    for vertex in vertices:
        print(vertex)

    print("\n면 데이터:")
    for face in faces:
        print(face)
```

5. IO

5-1, 2. DualSense.py, XBOX_input.py

DualSense.py와 XBOX_input.py는 같은 코드이다. 직접 개발, 작성한 코드는 아니며 inputs 라이브러리를 활용하여 곧대로 실행한다. 프로젝트에서 지양하고자 하는 방법이나 무릇 게임이라면 컨트롤러를 지원해야 한다는 개인적 견해에 검색을 해보니 마침 관련 라이브러리가 존재하여 포함하였다.

```
from inputs import get_gamepad

def get_dualsense_controller_input():
    while True:
        events = get_gamepad()
        for event in events:
            if event.ev_type == "Key":
                print(f"Button {event.ev_code} {'pressed' if event.ev_value ==
1 else 'released'}")
            elif event.ev_type == "Absolute":
                print(f"Axis {event.ev_code} value: {event.ev_value}")

if __name__ == "__main__":
    get_dualsense_controller_input()
```

5-3. MonitorInfo.py

MonitorInfo.py는 현재 화면의 정보를 담고 있는 클래스를 제공한다. Pygame의 display.Info()를 활용한 단순하며 존재 의미가 크지 않은 코드이며 실 사용시에는 직접

display.Info()를 부를 것이라고 예상되나 실용적, 완벽한 게임 엔진을 만드는 것이 아닌 관심 있는 분야의 게임 엔진을 개발하는 것이 프로젝트의 목표이기에 부담 없이 추가하였다.

```
# 화면 정보를 담고 있는 클래스
import pygame

class MonitorInfo:
    def __init__(self):
        self.width = pygame.display.Info().current_w
        self.height = pygame.display.Info().current_h
        self.center = (self.width/2, self.height/2)
        self.top = (self.width/2, 0)
        self.bottom = (self.width/2, self.height)
        self.left = (0, self.height/2)
        self.right = (self.width, self.height/2)
        self.topLeft = (0, 0)
        self.topRight = (self.width, 0)
        self.bottomLeft = (0, self.height)
        self.bottomRight = (self.width, self.height)
        self.centerTop = (self.width/2, 0)
        self.centerBottom = (self.width/2, self.height)
        self.centerLeft = (0, self.height/2)
        self.centerRight = (self.width, self.height/2)

    def getCenter(self):
        return self.center

    def getTop(self):
        return self.top

    def getBottom(self):
        return self.bottom

    def getLeft(self):
        return self.left

    def getRight(self):
        return self.right

    def getTopLeft(self):
        return self.topLeft

    def getTopRight(self):
        return self.topRight

    def getBottomLeft(self):
```

```
        return self.bottomLeft

    def getBottomRight(self):
        return self.bottomRight

    def getCenterTop(self):
        return self.centerTop

    def getCenterBottom(self):
        return self.centerBottom

    def getCenterLeft(self):
        return self.centerLeft

    def getCenterRight(self):
        return self.centerRight

    def getWidth(self):
        return self.width

    def getHeight(self):
        return self.height
```

6. Math

6-1. Numerical_derivative.py

Numerical_derivative.py는 수치 미분을 활용하여 미분의 근사값을 얻는다. 해당 코드에서는 중앙 차분을 사용하였으며 단변수, 이변수, 다변수 함수를 위해 각기 다른 세 함수를 제공한다.

```
# 수치미분 함수 (단변수 함수)
def numerical_derivative(f, x, h=1e-5):
    return (f(x+h) - f(x-h)) / (2*h)

# 수치미분 함수 (이변수 함수)
def numerical_derivative_2d(f, x, y, h=1e-5):
    return (f(x+h, y+h) - f(x-h, y-h)) / (2*h)

# 수치미분 함수 (다변수 함수)
def numerical_derivative_nd(f, x, h=1e-5):
    grad = [0.0] * len(x)

    for i in range(len(x)):
        tmp_val = x[i]
        x[i] = tmp_val + h
        fx1 = f(x)
```

```
        x[i] = tmp_val - h
        fx2 = f(x)

        grad[i] = (fx1 - fx2) / (2 * h)

        x[i] = tmp_val

    return grad

# 테스트 코드
if __name__ == "__main__":
    # f(x) = x^2 함수의 도함수를 구하는 예제
    f = lambda x: x**2
    df = numerical_derivative(f, 3.0)
    print(df)

    # f(x, y) = 2x + 3xy + y^3 함수의 도함수를 구하는 예제
    f = lambda x, y: 2*x + 3*x*y + y**3
    df_x = numerical_derivative_2d(f, 1.0, 2.0)
    df_y = numerical_derivative_2d(f, 1.0, 2.0)
    print(df_x, df_y)

    # f(x, y, z) = 2x + 3xy + y^3 + 2xz + z^2 함수의 도함수를 구하는 예제
    f = lambda x: 2*x[0] + 3*x[0]*x[1] + x[1]**3 + 2*x[0]*x[2] + x[2]**2
    df = numerical_derivative_nd(f, [1.0, 2.0, 3.0])
    print(df)
```

6-2. Numerical_integral.py

Numerical_integral.py는 사다리꼴 규칙을 이용하여 적분의 근사값을 얻는다. 함수의 적분값이 해당 함수의 그래프를 그렸을 때의 넓이를 나타낸다는 것을 이용한다. 미분과 같이 변수의 수 별로 다른 함수들을 제공한다.

```
# 수치미분 함수 (단변수 함수)
def numerical_derivative(f, x, h=1e-5):
    return (f(x+h) - f(x-h)) / (2*h)

# 수치미분 함수 (이변수 함수)
def numerical_derivative_2d(f, x, y, h=1e-5):
    return (f(x+h, y+h) - f(x-h, y-h)) / (2*h)

# 수치미분 함수 (다변수 함수)
def numerical_derivative_nd(f, x, h=1e-5):
    grad = [0.0] * len(x)

    for i in range(len(x)):
        tmp_val = x[i]
        x[i] = tmp_val + h
```

```
        fx1 = f(x)

        x[i] = tmp_val - h
        fx2 = f(x)

        grad[i] = (fx1 - fx2) / (2 * h)

        x[i] = tmp_val

    return grad

# 테스트 코드
if __name__ == "__main__":
    # f(x) = x^2 함수의 도함수를 구하는 예제
    f = lambda x: x**2
    df = numerical_derivative(f, 3.0)
    print(df)

    # f(x, y) = 2x + 3xy + y^3 함수의 도함수를 구하는 예제
    f = lambda x, y: 2*x + 3*x*y + y**3
    df_x = numerical_derivative_2d(f, 1.0, 2.0)
    df_y = numerical_derivative_2d(f, 1.0, 2.0)
    print(df_x, df_y)

    # f(x, y, z) = 2x + 3xy + y^3 + 2xz + z^2 함수의 도함수를 구하는 예제
    f = lambda x: 2*x[0] + 3*x[0]*x[1] + x[1]**3 + 2*x[0]*x[2] + x[2]**2
    df = numerical_derivative_nd(f, [1.0, 2.0, 3.0])
    print(df)
```

6-3. Trigonometric.py

Trigonometric.py는 사인, 코사인, 탄젠트 함수를 제공한다. import numpy를 사용할 수도 있지만 나만의 삼각함수라는 점에서 각 함수의 이름 앞에 my_를 추가했다. 각 삼각함수들은 테일러 급수를 이용하여 나타낸 뒤 역으로 계산하여 실수로 표현한다. 코드의 수를 변형해 몇 번째 수까지 사용할지 정할 수 있으며 기본값은 10이다.

```
# 테일러 급수를 이용한 사인, 코사인, 탄젠트 함수 구현
PI = 3.141592653589793

# 각을 라디안으로 변환하는 함수
def degrees_to_radians(degrees):
    return degrees * (PI / 180.0)

# 사인 함수 구현
def my_sin(x):
    x = degrees_to_radians(x)  # 각을 라디안으로 변환
    result = 0.0
```

```
    term = x
    n = 1

    for i in range(10):
        result += term
        term *= -(x**2) / ((2 * n) * (2 * n + 1))
        n += 1

    return result

# 코사인 함수 구현
def my_cos(x):
    x = degrees_to_radians(x)  # 각을 라디안으로 변환
    result = 1.0
    term = 1.0
    n = 1

    for i in range(10):
        term *= -(x**2) / ((2 * n - 1) * (2 * n))
        result += term
        n += 1

    return result

# 탄젠트 함수 구현
def my_tan(x):
    return my_sin(x) / my_cos(x)

# 테스트 코드
if __name__ == "__main__":
    print(my_sin(30))
    print(my_cos(30))
    print(my_tan(30))

    print(my_tan(45))
    print(my_sin(110))
```

6-4. VectorOperations.py

VectorOperations.py는 다양한 벡터의 계산을 담당한다. 첫 벡터의 스칼라 곱과 나눗셈, 두 벡터의 덧셈, 뺄셈, 내적, 외적과 평행, 수직, 동일방향 여부를 담당하는 함수들로 이루어져 있다. 이중 벡터의 외적의 경우 2차원 벡터의 외적이기에 반환 값이 일반적인 외적과 같이 벡터가 아닌 스칼라이다.

```
class VectorOperations:
    def __init__(self, vector1, vector2):
        self.vector1 = vector1
```

```
        self.vector2 = vector2

    # 벡터 덧셈
    def add(self):
        return [self.vector1[0] + self.vector2[0], self.vector1[1] +
self.vector2[1]]

    # 벡터 뺄셈
    def subtract(self):
        return [self.vector1[0] - self.vector2[0], self.vector1[1] -
self.vector2[1]]

    # 벡터 스칼라 곱셈
    def multiply(self, scalar):
        return [self.vector1[0] * scalar, self.vector1[1] * scalar]

    # 벡터 스칼라 나눗셈
    def divide(self, scalar):
        return [self.vector1[0] / scalar, self.vector1[1] / scalar]

    # 벡터의 내적
    def dot_product(self):
        return self.vector1[0] * self.vector2[0] + self.vector1[1] *
self.vector2[1]

    # 벡터의 외적 (2 차원 벡터이기에 z 값이 반환된다)
    def cross_product(self):
        return self.vector1[0] * self.vector2[1] - self.vector1[1] *
self.vector2[0]

    def is_parallel(self):
        return self.cross_product() == 0

    def is_orthogonal(self):
        return self.dot_product() == 0

    def is_same_direction(self):
        return self.dot_product() > 0

    def is_opposite_direction(self):
        return self.dot_product() < 0

# 테스트 코드
if __name__ == "__main__":
    v1 = [1, 2]
    v2 = [3, 4]
    vector_operations = VectorOperations(v1, v2)
```

```
    print("벡터 덧셈:", vector_operations.add())
    print("벡터 뺄셈:", vector_operations.subtract())
    print("벡터 스칼라 곱셈:", vector_operations.multiply(3))
    print("벡터 스칼라 나눗셈:", vector_operations.divide(3))
    print("벡터의 내적:", vector_operations.dot_product())
    print("벡터의 외적:", vector_operations.cross_product())
    print("벡터의 평행 여부:", vector_operations.is_parallel())
    print("벡터의 직교 여부:", vector_operations.is_orthogonal())
    print("벡터의 같은 방향 여부:", vector_operations.is_same_direction())
    print("벡터의 반대 방향 여부:", vector_operations.is_opposite_direction())
```

7. Physics

7-1. Collision_Detection.py

Collision_Detection.py는 개인적으로 게임 엔진에서 가장 중요하다고 생각하는 물체 간의 충돌을 담당한다. AABB, OBB, 다음 예상 위치를 계산에 포함한 충돌 계산, 점과 사각형의 충돌 계산, 점과 원의 충돌 계산, 두 원의 충돌 계산, 선분과 원의 충돌 계산, 사각형과 원의 충돌 계산으로 구성되어 있다.

```
import math
import numpy as np

def check_AABB_collision(rect1, rect2):
    """
    Axis-Aligned Bounding Box (AABB)를 사용하여 두 사각형 간의 충돌을 감지하는
함수

    Parameters:
    - rect1: (x1, y1, width1, height1) 형태의 튜플로 나타낸 첫 번째 AABB 정보
    - rect2: (x2, y2, width2, height2) 형태의 튜플로 나타낸 두 번째 AABB 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False 를 반환
    """
    return (
        rect1[0] <= rect2[0] + rect2[2] and
        rect1[0] + rect1[2] >= rect2[0] and
        rect1[1] <= rect2[1] + rect2[3] and
        rect1[1] + rect1[3] >= rect2[1]
    )

def check_obb_collision(rect1, rect2):
    """
    Oriented Bounding Box (OBB)를 사용하여 두 사각형 간의 충돌을 감지하는 함수

    Parameters:
```

```
    - rect1: (x1, y1, width1, height1, angle1) 형태의 튜플로 나타낸 첫 번째 OBB 정보
    - rect2: (x2, y2, width2, height2, angle2) 형태의 튜플로 나타낸 두 번째 OBB 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False를 반환
    """
    # 두 사각형의 중심 좌표
    center1 = (rect1[0] + rect1[2] / 2, rect1[1] + rect1[3] / 2)
    center2 = (rect2[0] + rect2[2] / 2, rect2[1] + rect2[3] / 2)

    # 두 사각형의 변 방향 벡터
    axis1 = np.array([math.cos(rect1[4]), math.sin(rect1[4])])
    axis2 = np.array([math.cos(rect2[4]), math.sin(rect2[4])])

    # 두 사각형의 변 벡터
    edges1 = [np.array([math.cos(rect1[4] + math.pi / 2), math.sin(rect1[4] + math.pi / 2)]),
              np.array([math.cos(rect1[4] - math.pi / 2), math.sin(rect1[4] - math.pi / 2)])]

    edges2 = [np.array([math.cos(rect2[4] + math.pi / 2), math.sin(rect2[4] + math.pi / 2)]),
              np.array([math.cos(rect2[4] - math.pi / 2), math.sin(rect2[4] - math.pi / 2)])]

    # 두 사각형의 중심 간의 벡터
    delta = np.array(center1) - np.array(center2)

    # 축으로 투영하여 겹침 확인
    for axis in [axis1, axis2] + edges1 + edges2:
        projection1 = [np.dot(np.array([rect1[i] - center1[0], rect1[i + 1] - center1[1]]), axis) for i in range(0, 8, 2)]
        projection2 = [np.dot(np.array([rect2[i] - center2[0], rect2[i + 1] - center2[1]]), axis) for i in range(0, 8, 2)]

        overlap = (
            max(projection1) >= min(projection2) and
            max(projection2) >= min(projection1)
        )

        if not overlap:
            return False

def check_moving_objects_collision(rect1, velocity1, rect2, velocity2):
    """
    두 개의 이동 중인 사각형 간의 충돌을 감지하는 함수
```

```
    Parameters:
    - rect1: (x1, y1, width1, height1) 형태의 튜플로 나타낸 첫 번째 사각형 정보
    - velocity1: (vx1, vy1) 형태의 튜플로 나타낸 첫 번째 사각형의 속도 정보
    - rect2: (x2, y2, width2, height2) 형태의 튜플로 나타낸 두 번째 사각형 정보
    - velocity2: (vx2, vy2) 형태의 튜플로 나타낸 두 번째 사각형의 속도 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False 를 반환
    """
    # 두 객체 간의 상대적인 속도 계산
    relative_velocity = np.array([velocity1[0] - velocity2[0], velocity1[1] -
velocity2[1]])

    # 두 객체의 예측 위치 계산
    future_rect1 = (rect1[0] + velocity1[0], rect1[1] + velocity1[1],
rect1[2], rect1[3])
    future_rect2 = (rect2[0] + velocity2[0], rect2[1] + velocity2[1],
rect2[2], rect2[3])

    # 예측 위치에서의 충돌 여부 확인
    return check_AABB_collision(future_rect1, future_rect2)

def check_point_in_rect(point, rect):
    """
    주어진 점이 주어진 사각형 내부에 있는지 확인하는 함수

    Parameters:
    - point: (x, y) 형태의 튜플로 나타낸 점의 위치
    - rect: (x, y, width, height) 형태의 튜플로 나타낸 사각형 정보

    Returns:
    - 점이 사각형 내부에 있으면 True, 그렇지 않으면 False 를 반환
    """
    return (
        rect[0] <= point[0] <= rect[0] + rect[2] and
        rect[1] <= point[1] <= rect[1] + rect[3]
    )

def check_point_in_circle(point, circle):
    """
    주어진 점이 주어진 원 내부에 있는지 확인하는 함수

    Parameters:
    - point: (x, y) 형태의 튜플로 나타낸 점의 위치
    - circle: (x, y, radius) 형태의 튜플로 나타낸 원의 정보

    Returns:
```

```
    - 점이 원 내부에 있으면 True, 그렇지 않으면 False를 반환
    """
    return np.linalg.norm(np.array(point) - np.array(circle[:2])) <= circle[2]

def check_circle_collision(circle1, circle2):
    """
    두 개의 원 간의 충돌을 감지하는 함수

    Parameters:
    - circle1: (x1, y1, radius1) 형태의 튜플로 나타낸 첫 번째 원의 정보
    - circle2: (x2, y2, radius2) 형태의 튜플로 나타낸 두 번째 원의 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False를 반환
    """
    return np.linalg.norm(np.array(circle1[:2]) - np.array(circle2[:2])) <=
circle1[2] + circle2[2]

def check_segment_circle_collision(segment, circle):
    """
    선분과 원 간의 충돌을 감지하는 함수

    Parameters:
    - segment: (x1, y1, x2, y2) 형태의 튜플로 나타낸 선분의 정보
    - circle: (x, y, radius) 형태의 튜플로 나타낸 원의 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False를 반환
    """
    # 선분의 양 끝점과 원의 중심 간의 거리를 계산
    distance1 = np.linalg.norm(np.array(segment[:2]) - np.array(circle[:2]))
    distance2 = np.linalg.norm(np.array(segment[2:]) - np.array(circle[:2]))

    # 선분의 양 끝점이 원 내부에 있는 경우
    if distance1 <= circle[2] or distance2 <= circle[2]:
        return True

    # 선분의 양 끝점이 원의 반지름을 기준으로 반대쪽에 있는 경우
    if distance1 >= circle[2] and distance2 >= circle[2]:
        # 선분의 양 끝점과 원의 중심을 지나는 직선의 방정식 계수 계산
        a = segment[3] - segment[1]
        b = segment[0] - segment[2]
        c = segment[2] * segment[1] - segment[0] * segment[3]

        # 선분의 양 끝점과 원의 중심을 지나는 직선과 원의 중심 간의 거리 계산
        distance = abs(a * circle[0] + b * circle[1] + c) / math.sqrt(a * a +
b * b)
```

```
        # 선분의 양 끝점과 원의 중심 간의 거리가 원의 반지름보다 작은 경우
        if distance <= circle[2]:
            return True

def check_rect_circle_collision(rect, circle):
    """
    사각형과 원 간의 충돌을 감지하는 함수

    Parameters:
    - rect: (x, y, width, height) 형태의 튜플로 나타낸 사각형 정보
    - circle: (x, y, radius) 형태의 튜플로 나타낸 원의 정보

    Returns:
    - 충돌이 발생하면 True, 그렇지 않으면 False 를 반환
    """

    # 사각형의 각 꼭지점을 원과의 충돌 검사
    for point in [(rect[0], rect[1]), (rect[0] + rect[2], rect[1]), (rect[0],
rect[1] + rect[3]), (rect[0] + rect[2], rect[1] + rect[3])]:
        if check_point_in_circle(point, circle):
            return True

    # 원의 중심을 사각형과의 충돌 검사
    if check_point_in_rect(circle[:2], rect):
        return True

    # 사각형의 각 변을 원과의 충돌 검사
    for edge in [(rect[0], rect[1], rect[0] + rect[2], rect[1]), (rect[0] +
rect[2], rect[1], rect[0] + rect[2], rect[1] + rect[3]), (rect[0], rect[1] +
rect[3], rect[0] + rect[2], rect[1] + rect[3]), (rect[0], rect[1], rect[0],
rect[1] + rect[3])]:
        if check_segment_circle_collision(edge, circle):
            return True

    return False
```

7-2. Particle_System.py

Particle_System.py는 파티클 이펙트를 제공한다. 위치, 크기, 수명, 속도, 가속도, 색상과 투명도를 파라미터로 입력 받아 작은 원들을 만들어 "꾸물"거리게 한다. 그래픽에 관련된 코드라고 분류할 수도 있지만 각 파티클들의 물리적인 처리가 주를 이루기에 Physics에 담았다.

```
import random
import math

class Particle:
```

```
    def __init__(self, x, y, min_size, max_size, min_life, max_life,
initial_speed, acceleration, color, alpha=255):
        self.x = x
        self.y = y
        self.size = random.uniform(min_size, max_size)
        self.life = random.uniform(min_life, max_life)
        self.initial_life = self.life
        self.initial_speed = initial_speed
        self.acceleration = acceleration
        self.color = color
        self.alpha = alpha

    def move(self):
        self.life -= 1
        if self.life > 0:
            self.x += self.initial_speed * math.cos(random.uniform(0, 2 *
3.14159))
            self.y += self.initial_speed * math.sin(random.uniform(0, 2 *
3.14159))
            self.initial_speed += self.acceleration
            self.alpha = int(255 * self.life / self.initial_life)

# 테스트 코드
if __name__ == "__main__":
    import pygame
    pygame.init()
    screen = pygame.display.set_mode((400, 400))
    clock = pygame.time.Clock()

    particles = []
    for _ in range(100):
        particles.append(Particle(200, 200, 1, 5, 100, 200, 0.1, 0.01, (255,
0, 0)))

    while True:
        screen.fill((0, 0, 0))

        for particle in particles:
            particle.move()
            pygame.draw.circle(screen, particle.color + (particle.alpha,),
(int(particle.x), int(particle.y)), int(particle.size))

        pygame.display.flip()
        clock.tick(60)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit()
```

7-3. Rigid_Body_Dynamics.py

Rigid_Body_Dynamics.py는 이름 그대로 리지드 바디를 담당한다. 다른 게임 엔진을 사용해본 경험에서 가장 뇌리에 남은 둘이 Collision과 Rigid Body였는데 이렇게 직접 개발하게 되니 감회가 새로웠다. 대부분 매우 기초적이며 당연한 내용이지만 개인적으로 직접 가속도를 제어하는 것이 아닌 힘의 크기를 입력 받고 물체의 질량에 따라 다른 가속도를 얻는 부분과 friction를 통해 마찰력을 적용하는 부분이 흥미롭고 재미있었다. main.py의 데모씬에서 자세히 볼 기회가 주어지기에 테스트 코드가 단순하다.

```
class RigidBody:
    def __init__(self, x, y, width, height, velocity_x=0, velocity_y=0,
mass=1.0, friction=0.1, angle=0):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.velocity_x = velocity_x
        self.velocity_y = velocity_y
        self.mass = mass
        self.friction = friction
        self.angle = angle

    # 힘을 가하는 함수
    def apply_force(self, force_x, force_y):
        # 뉴턴의 2 번째 법칙: F = ma
        acceleration_x = force_x / self.mass
        acceleration_y = force_y / self.mass

        self.velocity_x += acceleration_x
        self.velocity_y += acceleration_y

    # 각속도를 위한 힘을 가하는 함수
    def apply_impulse(self, torque):
        # 각 가속도: alpha = torque / I
        angular_acceleration = torque / self.mass
        self.angle += angular_acceleration

    # 리지드 바디의 상태를 업데이트하는 함수
    def update(self, is_friction):
        self.x += self.velocity_x
        self.y += self.velocity_y

        # 마찰력 모델
        if self.velocity_x != 0 and is_friction:
            friction_force_x = -1 * self.friction * self.velocity_x
```

```
            self.apply_force(friction_force_x, 0)

        if self.velocity_y != 0 and is_friction:
            friction_force_y = -1 * self.friction * self.velocity_y
            self.apply_force(0, friction_force_y)

        # 각속도 모델
        self.angle %= 360
        self.angle += self.velocity_x

# 테스트 코드
if __name__ == "__main__":
    rigid_body = RigidBody(0, 0, 10, 10, 2, 2, 1, 0.1)
    print("질량이 1 인 RigidBody 객체 생성:", rigid_body.__dict__)

    rigid_body.apply_force(10, 10)
    print("10, 10 크기의 힘을 가함:", rigid_body.__dict__)

    rigid_body.update(1)
    print("1 초 후:", rigid_body.__dict__)
```

8. Demo: main.py

main.py는 단순한 데모씬을 담는다. 플랫포머와 비슷한 프로그램으로 빨간 플레이어와 흰 플랫폼들로 이루어져 있다. 사용된 코드는 Collision_Detection, Rigid_Body_Dynamics, Particle_System, X_axis_Audio이다.

플레이어 캐릭터는 좌우 방향키로 가속도를 받으며 스페이스바로는 위쪽 방향으로 힘을 받는다. 중력 또한 힘으로 적용 받는다. 플레이어 캐릭터는 플랫폼과 닿기 전에는 중력의 영향을 받으며 충돌하는 동안 Y좌표가 고정되며 Y축의 가속도가 0으로 설정된다. 또한 충돌중인 경우 마찰력의 영향을 받는다. 또한 충돌중인 동안만 점프가 가능하다. 스페이스바를 눌러 점프 시 플레이어의 X축 위치에 따라 입체음향이 재생되며 바닥에는 파티클 이펙트가 재생된다. 작동을 확인하고 코드들을 테스트 하기 위한 씬 이기에 승리 조건이나 재시작 기능은 존재하지 않는다.

```
import pygame
import sys
import random
from Physics import Collision_Detection
from Physics import Rigid_Body_Dynamics
from Physics import Particle_System as Particle
from Audio import X_axis_Audio

# pygame 초기화
pygame.init()
```

```
# 화면 설정
screen_width = 1800
screen_height = 1000
screen = pygame.display.set_mode((screen_width, screen_height))

# 프로그램 이름
pygame.display.set_caption("Platformer")

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

# 색깔
white = (255, 255, 255)
black = (0, 0, 0)
red = (255, 0 , 0)

# 플레이어 리지드 바디
player_rigid_body = Rigid_Body_Dynamics.RigidBody(100, 100, 50, 50, 0, 0, 1,
0.02)
player_friction_x = False

# 플레이어 점프
player_jump = False
player_jump_power = 50

# 지형 리지드 바디 리스트
terrain_rigid_body_list = [
    Rigid_Body_Dynamics.RigidBody(0, 900, 1100, 10),
    Rigid_Body_Dynamics.RigidBody(1200, 800, 600, 10),
    Rigid_Body_Dynamics.RigidBody(600, 600, 500, 10),
    Rigid_Body_Dynamics.RigidBody(0, 500, 500, 10),
    Rigid_Body_Dynamics.RigidBody(1200, 400, 400, 10),
    Rigid_Body_Dynamics.RigidBody(0, 200, 400, 10),
    Rigid_Body_Dynamics.RigidBody(1700, 200, 300, 10)
]

# 파티클 생성 함수
def create_particles(x, y):
    particles = []
    for _ in range(20):  # 파티클 개수 조절 가능
        particle_color = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))
        particles.append(Particle.Particle(x, y, 1, 7, 100, 200, 1, 0.01,
particle_color))
    return particles
```

```
particles = []

# 메인 루프
while True:
    # 이벤트 처리
    for event in pygame.event.get():
        # 종료 이벤트
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        # 키보드 이벤트
        if event.type == pygame.KEYDOWN:
            # ESC 키
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()

            if event.key == pygame.K_SPACE:
                player_jump = True

            if event.key == pygame.K_LEFT:
                player_rigid_body.velocity_x = -10

            if event.key == pygame.K_RIGHT:
                player_rigid_body.velocity_x = 10

    # 중력 적용
    player_rigid_body.apply_force(0, 9.8 * player_rigid_body.mass)

    # 플레이어 리지드 바디와 지형 리지드 바디 충돌 감지
    for terrain_rigid_body in terrain_rigid_body_list:
        player_rect_list = [player_rigid_body.x, player_rigid_body.y,
player_rigid_body.width, player_rigid_body.height]
        terrain_rect_list = [terrain_rigid_body.x, terrain_rigid_body.y,
terrain_rigid_body.width, terrain_rigid_body.height]
        if Collision_Detection.check_AABB_collision(player_rect_list,
terrain_rect_list):
            # 플레이어 리지드 바디와 지형 리지드 바디 충돌 시 플레이어 리지드 바디의
위치를 조정
            player_rigid_body.y = terrain_rigid_body.y -
player_rigid_body.height
            player_rigid_body.velocity_y = 0
            player_friction_x = True
            if player_jump:
                particles = create_particles(player_rigid_body.x +
player_rigid_body.width / 2, player_rigid_body.y + player_rigid_body.height)
                player_rigid_body.apply_force(0, -player_jump_power)
```

```
                X_axis_Audio.play_audio_with_stereo("Audio_Samples\jump.mp3",
player_rigid_body.x, screen_width)
                player_rigid_body.update()
                player_jump = False

    # 플레이어 리지드 바디 업데이트
    player_rigid_body.update(player_friction_x)

    player_friction_x = False

    # 파티클 업데이트
    for particle in particles:
        particle.move()

    # 화면 지우기
    screen.fill(black)

    # 지형 그리기
    for terrain_rigid_body in terrain_rigid_body_list:
        terrain_rect_list = [terrain_rigid_body.x, terrain_rigid_body.y,
terrain_rigid_body.width, terrain_rigid_body.height]
        pygame.draw.rect(screen, white, terrain_rect_list)

    # 플레이어 그리기
    pygame.draw.rect(screen, red, player_rect_list)

    # 파티클 그리기
    for particle in particles:
        pygame.draw.circle(screen, particle.color + (particle.alpha,),
(int(particle.x), int(particle.y)), int(particle.size))

    # 화면 업데이트
    pygame.display.update()
    clock.tick(fps)

# pygame 종료
pygame.quit()
```

Execution Environment

해당 프로젝트는 이전의 게임 개발 프로젝트와 달리 .exe 파일로 실행되는 main.py가 부실한 데모 씬에 불구하다. 따라서 이 게임 엔진을 사용고자 한다면 사용하고자 하는 기능의 코드가 존재하는 위치를 파악한 뒤 임포트하는 과정을 거쳐야 한다. 이후 해당 기능의 구성 방식에 따라 함수를 호출하거나 클래스를 선언해서 사용하면 된다. 추가로 해당 게임 엔진에는 스무 개의 효과음 샘플이 있다. 전부 로열티 프리 음원들이므로 경로를 입력하여 활용하면 된다.