3D게임프로그래밍1 과제2 설명문서

2025.06.09

게임공학과

2022184015

김해님

1. 접근방식

첫 번째 과제로 작성했던 코드와 따라하기 14(인스턴싱)를 기반으로 코드를 작성하였다.

따라하기 예제에서는 오브젝트 쉐이더를 이용해 하나의 쉐이더가 오브젝트들을 들고 있는 방식으로 구현되어 있었지만, 개인적으로 오브젝트 쉐이더에 대해 이해하기 힘들었다. 특히 오브젝트 쉐이더는 그동안 짜오던 프레임워크와 잘 맞지 않는 형식이라고 생각하였다(항상 씬 안에 게임오브젝트의 std::vector를 가지고 오브젝트를 관리했기 때문에 씬 안에서 오브젝트의 동작 로직을 구현했다. 하지만 오브젝트 쉐이더의 경우 씬이 오브젝트 쉐이더를 가지고 있고, 오브젝트 쉐이더가 게임오브젝트를 들고 있기 때문에 씬에서 게임오브젝트들에 대한 직접적인 접근이 어렵다고 판단했다. 직접적인 접근이 가능하도록 구현할 수도 있겠지만 좋은 구조와 방식이 아니라고 생각했다. – 잘못된 이해일 수도 있지만, 이 이해 상태를 바탕으로 코드를 작성했다.). 폭발 애니메이션(저번 과제의 CExplosiveObject)을 제외하면 그렇게 많은 오브젝트가 한 번에 그려지진 않는다고 판단하여 각 오브젝트가 쉐이더를 들고 있는 형태로 구현할 예정이다(CDiffusedShader를 사용한다). 쉐이더는 BuildObject시에 씬 당 하나만 생성해서 게임오브젝트가 포인터로 들고 있도록 할 것이다.

위에 적은 것처럼 이전 과제에서 프레임 드랍의 문제가 되었던 폭발 애니메이션에 대해 인스턴싱 쉐이더를 이용해서 프레임 드랍을 줄일 예정이다. CGameobject 클래스를 상속받은 CExplosiveObject 클래스를 만들어서 이 오브젝트의 메쉬를 폭발 전에 그릴 메쉬로 설정하고, 폭발이 시작되면 원래 메쉬 대신 인스턴싱 쉐이더의 Render 함수를 호출하도록 할 것이다.

2. 변경점

첫 번째 과제의 프레임워크처럼 따라하기 14의 프레임워크를 변경하였다.

- GameFramework 클래스

GameFramework 클래스는 DirectX와 관련된 설정 및 씬 관리를 담당한다. 기존 따라하기 코드에서는 플레이어(m_pPlayer)와 카메라 관리를 GameFramework 클래스에서 하도록 짜여 있는데, 여러 개의 씬을 만들어야 하기 때문에 플레이어와 카메라를 각 씬에서 관리하도록 이동하였다.

- CScene 클래스

CScene 클래스는 GameFramework 위에서 동작하는 클래스이다. CScene 클래스는 각 씬의 부모클래스이며, 이 클래스로부터 상속받아 TitleScene, MenuScene 등 각 씬을 생성

한다.

- .obj파일으로부터 메쉬 생성

기존에 사용했던 .obj파일을 그대로 이용할 예정이다. 저번에 만들었던 파일 로드 함수와 바운딩박스 생성 함수를 하나로 합쳐서 ObjMesh 클래스의 생성자에서 동작하도록 수정하였다(코드 가독성은 떨어지지만 이 함수를 다른 클래스에서 부를 일이 없어 이렇게 수정했다).

- 추가한 라이브러리

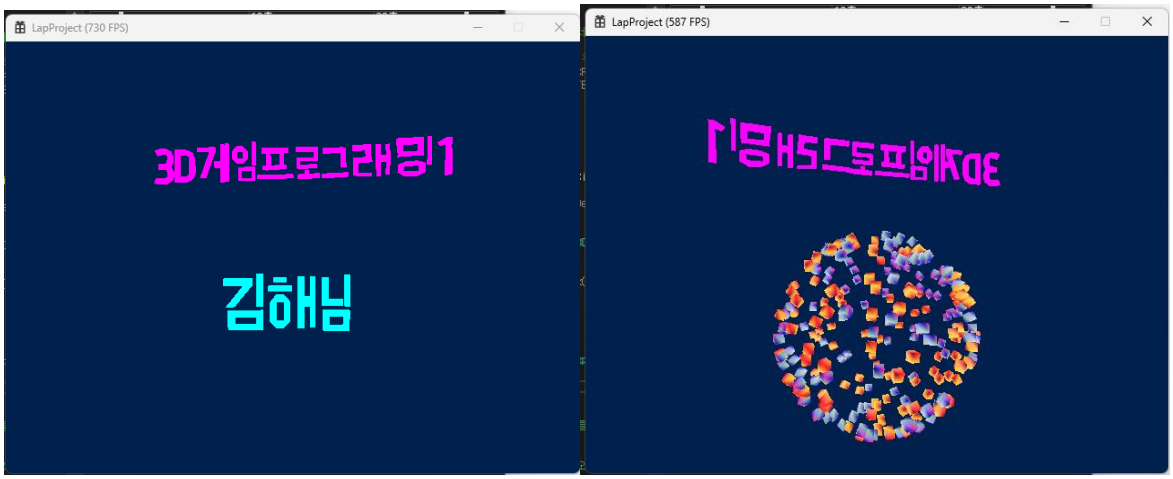
C++의 여러 라 이브러리를 추가하였다.

```
// cpp 헤더 파일입니다.
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <tuple>
#include <algorithm>
#include <array>
#include <unordered_map>
#include <unordered_set>
```

Fstream과 sstream은 .obj 파일을 읽기 위해, array와 unordered_map, unordered_set은 롤러코스터 트랙을 읽기 위해 추가했다. Vector는 메인 컨테이너로 사용하고 있고, algorithm 헤더는 clamp()함수를 사용하기 위해 추가했다.

3. 구현

1) 타이틀 씬



➔ 기본 실행화면과 이름 클릭 후 실행화면이다.

첫 번째 과제에서 구현했던 방식처럼 .obj 파일을 읽어 메쉬를 생성한 뒤, 이를 게임 오

브젝트로 구성하여 렌더링하기로 계획하였다. 타이틀 씬에서는 오브젝트의 개수가 적고 별도의 복잡한 처리가 필요 없기 때문에, 각각 개별적으로 변수로 선언하는 대신 std::vector<CGameObject*> 컨테이너를 사용하여 일괄적으로 관리하였다.

TitleScene 클래스는 타이틀 화면을 담당하는 씬으로, 타이틀 텍스트 및 오브젝트 렌더링, 사용자 입력 처리, 폭발 애니메이션 후 씬 전환을 제어한다. CScene을 상속하며, 공통 파이프라인과 셰이더를 활용해 간단한 상호작용과 애니메이션을 구현하고 있다.

1-2) 함수 설명

BuildObjects 함수는 플레이어와 카메라를 생성하고 세팅, obj 파일로부터 데이터를 읽어서 메쉬를 생성하고 초기 위치를 지정하는 동작을 한다. 3D게임프로그래밍 1 오브젝트는 CRotatingObject 클래스로 선언하여 회전하도록 만들었고, 이름 오브젝트는 CExplosiveObject 클래스로 만들어 피킹시 폭발 애니메이션이 동작하도록 구현하였다. 두 오브젝트를 std::vector<CGameObject*> objects; 벡터에 담아 관리하였다.

OnProcessingMouseMessage 함수에서는 마우스 좌클릭이 발생했을 때, 마지막 오브젝트가 마우스 커서에 의해 피킹 되었는지 확인하고, 해당 오브젝트를 폭발시키는 StartExplosion 함수를 호출한다. 여기서 dynamic_cast<>를 이용하여 CGameObject*에 저장되어 있던 오브젝트를 CExplosiveObject*로 다운캐스팅하여 함수를 호출했다.

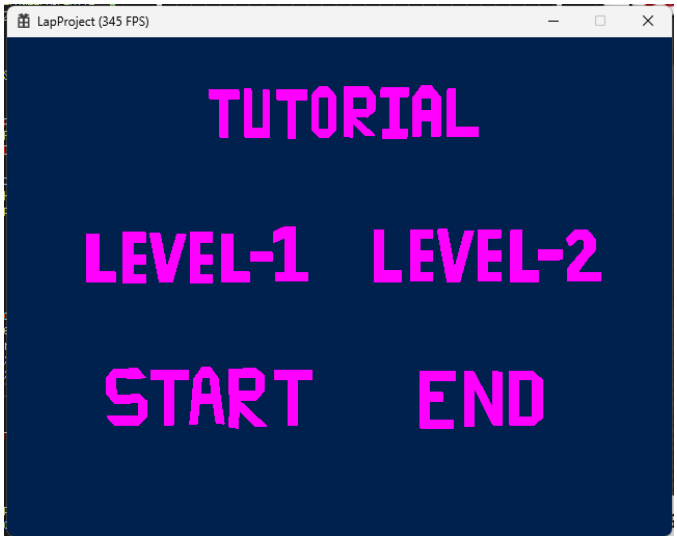
Reset 함수에서는 오브젝트들의 상태를 초기화한다. 이름 오브젝트의 폭발 상태를 초기화한다. 처음에는 오브젝트를 모두 삭제하고 buildObject 함수를 다시 호출하는 방법으로 구현하려고 했으나, 오브젝트를 삭제하려면 업로드 힙에 다시 리소스를 올리는 작업을 해야 하기 때문에 위치와 상태만 초기화하는 방법을 사용했다.

PickObjectPointedByCursor 함수에서는 마우스 클릭 좌표를 기준으로 카메라 뷰 및 프로젝션 행렬을 이용하여 Ray를 생성한 뒤, 오브젝트의 바운딩 박스와의 충돌 여부를 검사하여 피킹 기능을 구현한다. 가장 가까운 오브젝트를 반환하여 상호작용에 사용할 수 있도록 한다.

1-3) 조작 방법

이름 오브젝트를 마우스 좌클릭하면 이름 오브젝트가 폭발한다. 폭발이 끝나면 메뉴 씬으로 전환된다.

2) 메뉴 화면



➔ 메뉴 씬의 실행 화면이다.

플레이어가 게임의 각 레벨로 진입하기 전 메뉴를 선택할 수 있는 장면을 구성하는 씬이다. 사용자는 마우스로 오브젝트들을 클릭하여 각기 다른 씬으로 진입할 수 있으며, 이러한 선택은 Ray 기반의 피킹 방식으로 처리된다.

MenuScene 클래스는 CScene 클래스를 상속받아 피킹과 씬 전환 처리 동작을 한다.

2-2) 함수 설명

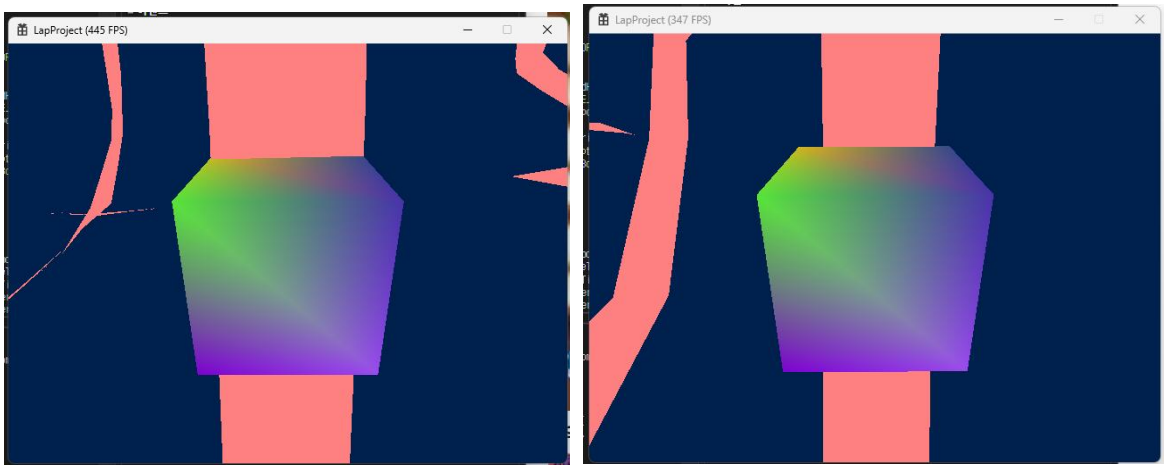
BuildObject 함수는 메뉴 씬의 핵심 오브젝트들을 생성하는 함수이다. CPlayer 객체를 생성하고 카메라를 세팅한 뒤, 각 메뉴 항목(Tutorial.obj, Lv1.obj, Lv2.obj, Start.obj, End.obj)을 CObjMeshDiffused로 로드하여 CGameObject로 생성 및 위치, 회전 설정을 수행한다. 오브젝트들은 모두 std::vector<CGameObject*> objects 컨테이너에 저장되어 이후 렌더링 및 피킹 처리에 활용된다.

OnProcessingMouseMessage 함수는 마우스 클릭 이벤트가 발생했을 때 호출된다. 좌클릭이 발생하면 커서 위치 기준으로 Ray를 쏘아 충돌한 오브젝트를 확인하고, 특정 메뉴 오브젝트가 클릭된 경우 change 플래그를 true로 설정하고 idx 값으로 씬 전환 목적지를 지정한다.

2-3) 조작 방법

Tutorial 메쉬를 좌클릭하면 타이틀 씬으로, Lv1 메쉬 또는 Start 메쉬를 좌클릭하면 롤러코스터 씬(씬1)으로, Lv2 메쉬를 좌클릭하면 탱크게임 씬(씬2)으로, End 메쉬를 좌클릭하면 Exit(0)을 호출해 프로그램을 종료한다.

3) 씬1(롤러코스터 씬)



➔ 롤러코스터 씬의 실행 화면이다.

Scene1은 트랙(.obj 파일)을 따라 자동으로 움직이는 카트 형태의 플레이어 오브젝트를 구현한 롤러코스터 씬이다.

씬 내의 주된 구성 요소는 트랙 메쉬, 카트(플레이어), 경로 추적을 위한 선형 경로이다.

플레이어는 사전 정의된 경로를 따라 이동하며, 이동 방향에 따라 회전 보간 및 경사도 기반 가속도가 적용된다.

저번 과제에서 만들었던 것처럼 플레이어의 회전을 구현하려 했는데, 여러 문제가 있어서 플레이어의 회전을 제외하고 구현하였다. 지금은 rotate 함수를 이용해서 yaw방향으로만 회전하게 구현되어 있다.

3-2) 함수 설명

BuildObjects 함수는 .obj 파일로부터 트랙의 라인 정보를 로드하고 경로를 구성한다.

LoadLineToMesh 함수는 .obj 파일을 파싱하여 정점 및 선 데이터를 추출한다. 전체 정점과 선의 개수를 처음부터 알지 못하기 때문에 std::vector를 이용하여 가변길이 배열을 사용한다.

BuildOrderedLinePath 함수는 .obj 파일로부터 불러온 정점과 선 정보를 기반으로, 하나의 연결된 경로를 순서대로 정렬하는 기능을 수행한다. .obj 파일의 데이터는 선분으로 연결된 정점의 쌍을 제공하기 때문에, 게임 내에서 플레이어가 따라갈 수 있도록 정점들을 선형으로 정렬된 경로 형태로 재구성하는 과정이 필요하다.

이 함수는 먼저 선 정보를 이용하여 각 정점의 인접 정점을 기록한 인접 리스트를 구성한다. 인접 노드가 하나뿐인 정점을 시작점으로 설정하여, 방문하지 않은 이웃 정점을 따라가며 전체 경로를 순차적으로 탐색한다. 이러한 방식은 그래프의 깊이 우선 탐색(DFS)과 유사한 방식으로, 정점 간 연결 관계를 따라가며 outPath 벡터에 정점 좌표를 차례대

로 저장하게 된다. 탐색 도중 이미 방문한 정점을 제외하고 다음 정점을 선택함으로써 중복 방문을 방지하며, 종점에 도달하면 경로 구성이 완료된다.

이렇게 완성된 outPath는 이후 애니메이션 처리 함수에서 카트가 따라가는 경로로 사용되며, 실질적인 카트 이동 및 회전에 중요한 역할을 한다.

ProcessInput 함수는 마우스 우클릭 드래그로 플레이어와 카메라의 yaw 회전 동작을 처리하는 함수이다.

Reset 함수는 속도와 현재 위치, 타이머를 초기화하여 트랙을 처음부터 실행할 수 있도록 지정하는 함수이다.

OnProcessingKeyboardMessage 함수는 n키를 눌렀을 때 다음 씬으로 전환, esc 키를 눌렀을때 메뉴 씬으로 전환되게 구현하였다.

AnimateObjects 함수는 Scene1 클래스에서 플레이어(카트)가 트랙 경로를 따라 자연스럽게 이동하도록 처리하는 함수이다. 이 함수는 프레임마다 호출되어 트랙에 따른 플레이어 이동 및 회전을 처리하며, 트랙의 경사에 따른 가속도 조절 및 방향 벡터 보간 등을 통해 현실적인 움직임을 구현한다.

먼저 현재 경로 인덱스(m_iCurrentPathIndex)가 전체 경로점 수(m_vTrackPoints.size())보다 크거나 같은지 확인해 경로의 끝에 도달했는지를 판단한다. 만약 마지막 지점에 도달한 경우, 타이머를 통해 일정 시간(2초) 대기 후 탱크게임 씬으로 전환한다.

플레이어의 현재 위치와 목표 위치를 비교하여 이동 방향 벡터와 거리를 계산한다. 만약 두 점의 거리가 매우 가까운 경우, 목표 위치에 도달한 것으로 간주하고 다음 경로 인덱스로 이동한다.

트랙의 높이 차에 따라 경사 방향을 판단하고 이에 따라 가속도를 적용한다. 내리막길일 경우 빠르게 가속하며, 오르막길일 경우 느리게 감속하도록 설계되어 있다. 결과적으로 현재 속도(m_fCurrentSpeed)는 일정 범위 내에서 보정되어 뒤로 가거나 하는 일을 방지한다.

현재 속도를 기반으로 이동 거리를 계산하고, 해당 프레임에서 이동 가능한 거리가 목표 지점과의 거리보다 클 경우 즉시 목표 지점으로 이동시킨다. 그렇지 않으면 보간된 방향 벡터에 따라 위치를 업데이트한다.

트랙 메시에 정의된 법선 벡터(normal)를 기준으로 플레이어의 Look, Right, Up 벡터를 다시 계산한다. 이 벡터들은 현재 상태와 선형 보간을 통해 부드럽게 변경된다. 이후 새 Look 벡터와 이전 Look 벡터 간의 yaw 각도를 계산하여 플레이어의 회전을 적용한다.

이동과 회전 처리가 완료된 후, 플레이어 및 트랙 오브젝트의 애니메이션과 내부 상태를

업데이트한다. 플레이어는 매 프레임 자연스럽게 트랙을 따라 이동하고, 회전하며, 트랙의 지형에 부합하는 움직임을 보이게 된다.

3-3) 조작 방법

n키를 누르면 다음 씬(탱크 게임 씬)으로 전환된다. Esc를 누르면 메뉴 씬으로 전환된다. 우클릭 드래그하면 플레이어 오브젝트를 회전할 수 있다.

4) 씬2(탱크 게임 씬)

➔ 탱크게임 씬의 실행 화면이다.

Scene2는 CScene을 상속받는 게임 장면 클래스이다. 이 클래스는 플레이어 객체, 적 객체(탱크), 장애물 객체, 맵, 승리 메시지 오브젝트 등을 관리하며, 해당 객체들의 초기화, 애니메이션, 입력 처리, 충돌 처리, 렌더링을 수행한다.

장애물은 CObstacles 클래스를 이용하여 구현했는데, 이 클래스는 CGameObject에서 상속받아 정해진 방향으로 정해진 속도만큼 계속 이동하도록 동작한다. 장애물은 총알을 막고, 플레이어와 적 오브젝트가 지나가지 못하도록 막는 역할을 한다.

적은 CTankObject 클래스를 이용하여 구현했다. 이 클래스는 CExplosiveObject 클래스로부터 상속받았고, Body(탱크의 몸체, CGameObject 클래스로 구현되어 있다)와 이동방향, 속도를 가지고 있다. 일정 속도로 맵에서 왔다갔다하는 동작을 한다. CExplosiveObject 클래스는 CInstancingShader를 가지고 있고, 폭발 애니메이션은 인스턴싱 쉐이더를 통해 구현했다. 폭발 애니메이션이 동작하는 동안에는 body의 active를 false로 변경하여 폭발 애니메이션만 보이도록 구현했다.

플레이어는 CPlayer타입의 플레이어 객체(탱크의 머리 부분)와 CGameObject타입의 몸통으로 구현되어 있다. 플레이어는 yaw 회전을 할 수 있으며 몸통은 플레이어의 위치를 따라간다.

충돌은 맵과 장애물, 플레이어, 적 충돌, 장애물과 플레이어, 적, 총알 충돌, 플레이어와 적 충돌, 적과 총알, 적 충돌을 구현하였다.

4-2) 함수 설명

OnProcessingKeyboardMessage함수는 키보드 입력을 처리하는 함수이다. s키를 누르면 쉴드 활성화(쉴드는 3초 활성화된다), a키는 자동 타겟팅 토글, w키는 모든 적을 폭발시키며 게임을 종료시킨다. Ctrl은 자동 타겟팅이 켜져있을 때 피킹으로 선택된 적에게 총알을 발사하고, 자동 타겟팅이 꺼져있으면 look 방향으로 총알을 발사한다, f1키를 누르면 1인칭, f3키를 누르면 3인칭으로 시점이 변환된다. Esc를 누르면 메뉴 씬으로 돌아간다. 플레이어를 CPlayer로 선언했기 때문에 dynamic_cast를 이용해서 CTankPlayer로 변환하여 ctrl 키 동작을 수행하도록 구현했다. 총알은 0.25초에 한발씩 발사할 수 있도록 구현했다.

OnProcessingMouseMessage 함수는 마우스 우클릭으로 피킹을 처리하는 함수이다. 자동 타겟팅이 켜져있을 때만 동작한다.

BuildObjects 함수는 게임에 필요한 모든 게임 오브젝트를 생성하고 초기화하는 함수이다. 플레이어는 기본적으로 'head.obj' 파일을 메쉬로 사용하며, 3인칭 카메라가 적용되어 있다. 플레이어와 별도로 몸체 오브젝트인 pBody도 함께 생성되며, 큐브 형태의 메쉬를 사용하고 플레이어의 머리보다 약간 아래에 배치해 탱크의 머리와 몸통을 표현했다. 일정 시간 동안 무적 상태를 나타내는 쉴드 오브젝트도 생성하는데, 크기가 더 큰 큐브 메쉬로 만들어져 있다. 이 쉴드 오브젝트는 쉴드가 켜져 있을 때만 플레이어의 위치를 따라가고, 그려진다.

게임 내 장애물은 총 5개가 생성되며, 각 장애물은 X축 방향으로 이동할 수 있도록 초기 방향 벡터와 속도를 랜덤하게 설정한다. 장애물은 일정 간격으로 Z축 상에 배치되어 플레이어와 적, 총알의 경로를 방해하게 된다.

적 오브젝트는 총 10개가 생성되며, 폭발 애니메이션을 갖춘 적 탱크 객체로 구현된다. 적의 위치는 다른 적이나 장애물과 겹치지 않도록 거리를 통해 검사하고 생성된다.

이 외에도 게임 전체 맵을 구성하는 오브젝트가 'map.obj' 메쉬를 기반으로 생성되며, 플레이어 시점에 따라 위치가 조정될 수 있도록 설정된다. 모든 적이 제거되었을 때 출력되는 승리 메시지(you win) 오브젝트도 함께 생성되며, 초기에는 보이지 않지만 승리 조건 충족 시 화면에 그려지도록 설계되어 있다.

Reset 함수는 오브젝트들의 상태를 초기 상태로 되돌리는 함수이다. 플레이어와 몸통의 위치를 원점으로 되돌리고, 적들의 위치를 새로 지정하고 상태 변수를 초기화한다.

Render 함수에서는 카메라와 상태변수에 따라 오브젝트들을 렌더링한다.

AnimateObjects 함수는 게임 루프에서 매 프레임 호출되는 함수로, 장면 내의 모든 오브젝트에 대해 애니메이션을 적용하고 게임 상태를 갱신하는 역할을 수행한다.

먼저, 남아있는 적의 수가 0 이하일 경우 승리 조건이 충족되었다고 판단하여, 화면에 출력할 승리 메시지 오브젝트의 위치를 플레이어 바로 앞쪽으로 이동시킨다. change 플래그를 true로 설정하고 다음 장면으로 전환할 수 있도록 한다. 플레이어의 체력이 0 이하로 떨어졌을 경우에는 게임을 종료하고 메뉴화면으로 돌아간다. 발사 딜레이(delay)와 쉴드 유지 시간을 경과 시간만큼 증가시키며, 쉴드가 3초 이상 유지된 경우 자동으로 해제되도록 처리한다.

플레이어 오브젝트는 CTankPlayer로 다운캐스팅한 후, 해당 객체의 Animate 및 Update 함수를 호출하여 위치, 회전, 카메라 등을 시간에 따라 갱신한다. 이어서, 적 오브젝트들과 장애물 리스트를 순회하며 각각의 Animate 함수를 호출하여 움직임이나 애니메이션 효과를 갱신한다.

마지막으로, 게임의 진행에 있어 중요한 충돌 처리를 수행한다. 총알과 적의 충돌, 총알과 장애물 충돌, 적과 벽 또는 장애물과의 충돌, 적들끼리의 충돌, 적과 플레이어의 충돌 등 다양한 조건에 따라 상황을 판별하고 그에 따른 반응을 적용한다.

ProcessInput 함수에서는 이동 키 입력과 화면 회전을 처리한다. 이동을 수행하기 전에는 플레이어가 이동할 예정인 위치에 대한 충돌 여부를 검사한다. 플레이어 몸체의 바운딩 박스를 다음 위치로 옮긴 뒤, 장애물들과의 충돌 여부를 확인하여 충돌 검사한다. 만약 이동 예정 위치가 어떤 장애물과도 겹치지 않는다면 실제 이동이 수행된다. 마우스 좌클릭을 누르고 드래그하여 화면을 회전할 수 있다.

CheckEnemyByBulletCollisions 함수는 적과 총알의 충돌을 검사한다. 플레이어로부터 총알 배열을 가져온 후, 각 총알이 활성화되어 있는지를 확인하고, 적의 바운딩 박스와 총알의 바운딩 박스가 충돌했는지를 판별한다. 충돌이 발생하면 적은 폭발 상태로 전환되며, 총알은 초기화되고 남은 적의 수가 하나 감소한다.

CheckObstacleByBulletCollisions 함수는 총알이 장애물과 충돌하는지를 검사한다. 장애물과의 충돌이 감지된 경우 총알이 제거된다.

CheckEnemyByWallCollisions 함수는 적 오브젝트가 게임 맵의 경계(–7.0f ~ 7.0f)를 벗어나는지를 확인한다. 경계를 넘는 경우, 해당 적의 이동 방향 벡터에서 X축 성분을 반전시켜 반대 방향으로 이동하게 한다. 적 오브젝트가 벽을 뚫고 나가지 못하도록 막는 함수이다.

CheckEnemyByEnemyCollisions 함수는 적들 간의 충돌을 검사하며, 두 적의 몸체 바운딩 박스가 겹치는 경우를 판단한다. 충돌이 발생하면 서로의 이동 방향 벡터를 교환하여 반대 방향으로 이동하게 구현하였다.

ClampPlayerBodyPosition 함수는 플레이어 몸통이 맵의 경계(–7.0f ~ 7.0f)를 넘지 않도록

제한하는 역할을 한다. 경계 제한 후에는 플레이어의 머리 위치도 자동으로 보정된다. 맵 배경 오브젝트의 Z위치를 플레이어 위치에 동기화시켜 맵이 무한하게 이어지는 것처럼 구현한다.

CheckEnemyByObstacleCollision 함수는 적 오브젝트가 장애물과 충돌했는지를 판단한다. 충돌이 발생한 경우, 충돌 지점의 상대 거리(X축 또는 Z축)를 기준으로 더 큰 방향으로 이동 방향을 반전시킨다. 즉, 수평 충돌이면 X축, 수직 충돌이면 Z축 방향이 반전된다.

CheckEnemyByPlayerCollision 함수는 적이 플레이어와 충돌했는지를 검사한다. 충돌이 감지되면 해당 적은 폭발하고 제거되며, 플레이어의 체력이 1 감소한다. 쉴드가 활성화되어 있으면 충돌을 무시한다.

PickObjectPointedByCursor 함수는 피킹을 처리하는 함수이다. 오브젝트 자체가 몸통과 머리 두개의 오브젝트가 합쳐진 형태이기 때문에 PickObjectByRayIntersection함수를 재정의하였다. 이 함수는 머리와 몸통 두 오브젝트의 바운딩박스 모두에 Ray 충돌 검사를 해서 충돌한 숫자를 다 더해서 반환한다.

4-3) 조작 방법

키보드 방향키로 움직이고, 좌클릭 드래그로 화면 회전한다. F1을 누르면 1인칭 시점, F3을 누르면 3인칭 시점으로 전환된다. 피킹은 우클릭으로 한다. S키를 누르면 쉴드가 활성화되고, A키는 자동 타겟팅 토글이다. W키를 누르면 모든 적 오브젝트를 폭발시키고 게임이 끝난다. CTRL키로 총을 발사하고, 총은 선택된 오브젝트가 있으면 자동 조준되어 따라간다.

4. 느낀 점

교수님께서 말씀하셨던 대로 '화면에 띄우는 것'조차 오랜 시간이 걸렸다. 아무래도 DirectX의 렌더링 과정과 초기화 순서를 제대로 이해하진 못한 것 같다. 그래도 과제를 통해서 이론으로만 배웠던 내용들을 실습하며 수많은 버그와 오동작들을 만나는 과정에서 조금이나마 이해하게 된 것 같다.

가졌던 가장 큰 의문점은 메뉴 씬을 구현할 때, End라고 적힌 오브젝트가 제대로 렌더링되지 않았던 점이 의문이었다. 저번 과제의 obj 파일을 그대로 사용했는데 어떤 이유에서인지 프러스텀 컬링에 걸려서 제대로 렌더링이 되지 않았다. 도저히 이해가 안가서 위치도 변경해보고, face orientation도 확인해봤는데 아무런 문제가 없었다. 다른 위치에 옮겨서 그려도 그려지지 않았다. 같은 위치에 다른 .obj 파일을 이용해서 그렸을 때에는 또 제대로 그려졌다. .obj 파일이 깨졌다고 생각해서 블렌더에서 .obj파일을 z축방향으로 크기를 조금 늘린 후 다시 export해서 렌더링 했더니 제대로 그려졌다. 정말 이유를 알 수 없는 문제였다. 해결돼서 다행이다. 두께에 따라서 뭔가 오동작하는 경우가 있는지 궁금

하다.

Reset을 추가하는 과정에서 모든 오브젝트를 삭제하고 buildObject를 다시 호출하도록 만들었다가 아무것도 그려지지 않는 일이 발생했다. 처음에는 뭐 때문에 이렇게 된 건지 몰랐는데, 찬찬히 살펴보니 리소스가 버퍼에 올라가지 않은 것 같았다. 오브젝트를 생성할 때마다 리소스를 버퍼에 올려줘야 한다는 사실을 잊고 버퍼를 다시 생성해 올리고 삭제하는 코드를 넣지 않은 것이었다. 하지만 똑같은 리소스 데이터를 두번이나 올리는 것은 비효율적이라고 생각해서 위치와 상태값만 초기화하도록 구현했다.

인스턴싱 쉐이더를 이용해서 폭발 애니메이션을 구현했을 때 가장 뿌듯했다. 저번 과제에서도 폭발 애니메이션이 프레임을 너무 많이 잡아먹어서 불편했었는데, 이제 방어가 잘 되는 것 같아서 다행이라고 생각한다. 모든 오브젝트가 쉐이더를 들고있는 구조에 대해서는 개선할 점이 있겠지만, 지금은 기본적인 프레임 방어는 가능하니까 어느정도 만족하고 있다. 더 다양한 기능을 추가해보고 싶다는 생각이 들었다.