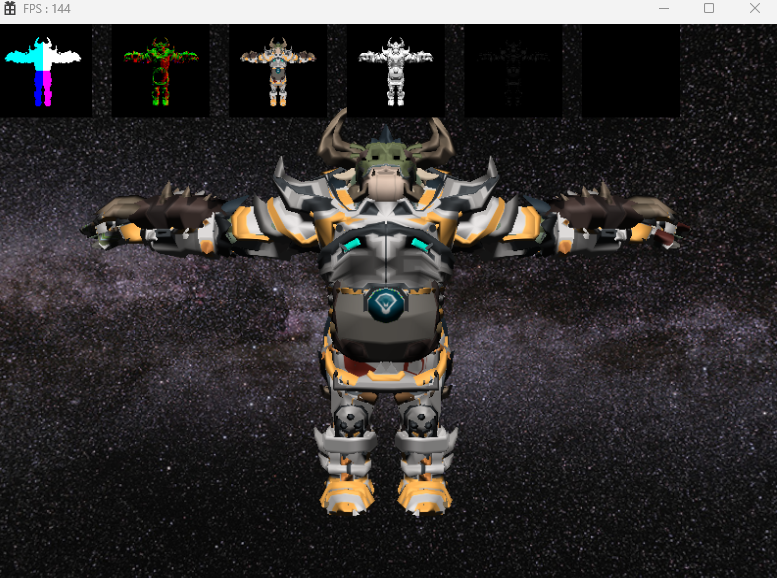
| **비단길(Silk Road)** | |
| --- | --- |
| **2주차** | **2023.12.31 (일) ~ 2023.01.06 (토)** |

**■ 주간 회의록**

**▶ FBX 파일을 통한 3D 모델 불러오기**

* fbx 파일을 통해 모델을 화면에 랜더링하는 것까지 구현 완료
* fbx 파일을 바이너리 형태로 만들고 불러오는 코드 구현 필요

unity상의 prefab 모델 -> C# 스크립트를 통해 바이너리 파일로 변환 (우리 게임에 적용시 필요한 정보만 추출) -> 바이너리 파일을 통해 계층모델 읽어오기

**▶ Git repo 생성**

**▶ 네트워크 모델 설계**

**1초당 60회 통신 가정**

* 플레이어 정보
  + 입력 정보를 서버(호스트)로 전송 (Ex: 전진 키 다운, 전진 키 업)
  + 플레이어의 현재 시점 회전값
* 몹 정보
  + 몹 목표 지점 좌표
  + 몹 어그로
  + 몹 공격
    - 근접: xz 좌표를 계산해 범위 안에 플레이어가 있다면 해당 방향으로 공격 모션 시작 -> 피격 판정 시간에 들어온 플레이어가 있다면 해당 플레이어에게 패킷 전송
    - 투사체: 각 원거리 몹 클래스에 가지고 있는 최대 투사체를 기준, 공격 가능 여부를 판단해 공격. 원거리를 몹을 랜더링 할 때와 피격 계산을 할 때, 투사체가 존재한다면 함께 랜더링 및 계산
  + 플레이어가 보내주는 피격 여부를 호스트에서 확인 후 체력을 계산해서 모든 클라이언트에 전송하여 체력바 표시
* 스테이지 정보 (난이도 미터기, 아이템, …)

**▶ 코딩스타일**

* **귀찮다고 변수명 막 짓지 않기**
* **파일이름** 
  + CommandQueue.cpp
  + CommandQueue.h
* **클래스**
  + class ConstantBuffer {

public:

ConstantBuffer();

~ConstantBuffer();

void Init(CBV\_REGISTER reg, uint32 size, uint32 count);

void Clear();

private:

ComPtr<ID3D12Resource> m\_cbvBuffer;

uint32 m\_elemetSize = 0;

}

* **구조체**
  + struct RenderTarget {

shared\_ptr<Texture> target;

float clearColor[4];

}

* **enum, enum class**
  + enum class CONSTANT\_BUFFER\_TYPE : uint8 {

GLOBAL,

TRANSFORM,

MATERIAL,

END

};

* enum {

CONSTANT\_BUFFER\_COUNT = static\_cast<uint8>(CONSTANT\_BUFFER\_TYPE::END)

};

* **매크로**
  + #define MAX\_LOADSTRING 100
* **쉐이더 파일**
  + struct VS\_IN

{

float3 pos : POSITION;

float2 uv : TEXCOORD;

float3 normal : NORMAL;

float3 tangent : TANGENT;

};

* **디렉터리 이름**
  + Object
  + Collider

**■ 다음 주 세부 계획**

**▶ [공통]**

* 유니티 모델 프리팹 export, Directx12 환경에서 import
* 애니메이션 샘플 적용

**▶ [권순원]**

* 방학 때 제작한 코드에서 2주차 회의에서 결정한 코드스타일과 다른 부분 수정
* 유니티 프리팹 파일과 파티클 시스템 공부해서 정리(파티클 시스템 큰 틀 구상)

**▶ [박준영]**

* 개별 브랜치 파서 direct2d 적용한 ui 모듈 개발시작 -> Scene 내에 포함시켜 각 씬마다 필요한 ui 요소(HUD, 버튼, 텍스트, 게이지) 저장 가능하게
* 우선적으로 문자열, 수치 텍스트 출력 구현

**▶ [엄장헌]**

* 물리엔진을 위한 클래스를 만들고, 게임 객체들을 담기 위한 자료구조를 만든다. 이후 옥트리를 통해 공간분할
* 충돌검사를 위한 collider 컴포넌트를 제작해 게임 객체에서 사용할 수 있게 한다.
* 간단한 완전 탄성 충돌 코드 작성후 테스트

**■ 다음 주 회의 안건**

* 서버 관련 추가적인 의견 및 결정

**■ 작업 일지**

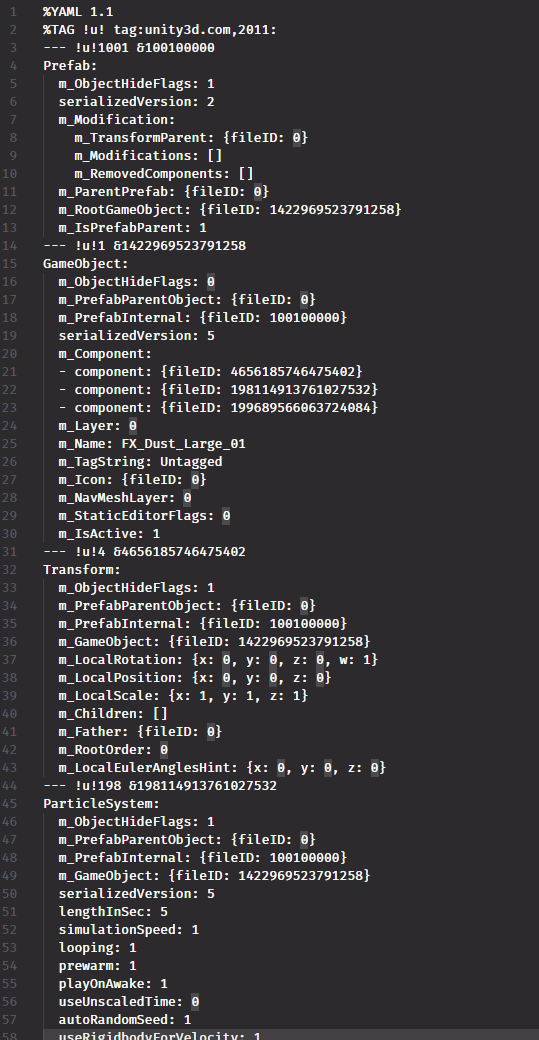
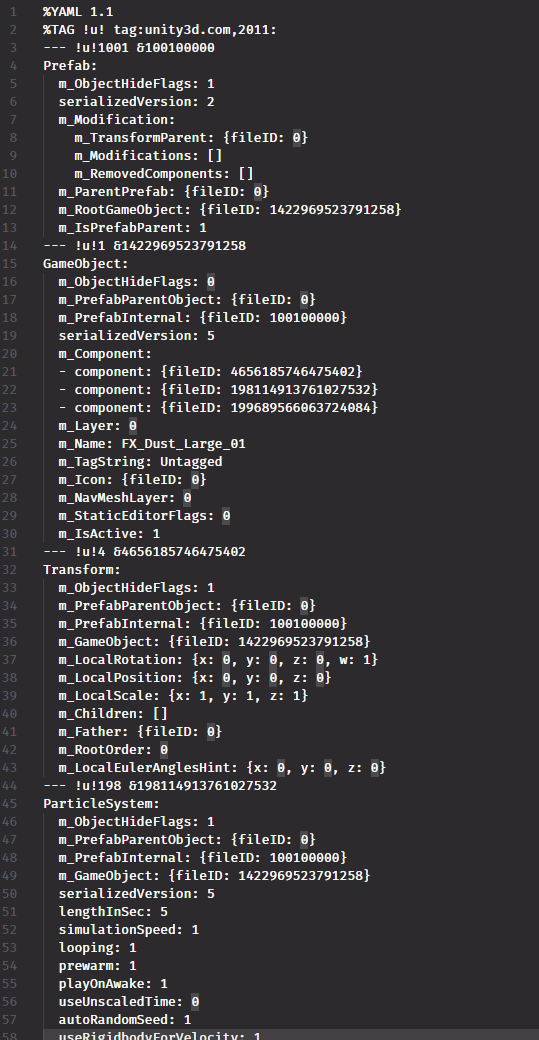
**▶ [권순원]**

c++ 멀티스레드만 집중적으로 강의하는 강좌는 없어서 이미 알고 있던 것과 레퍼런스 문서 등을 참고해 지금까지 해온 프로젝트에 스레드를 적용하면서 공부하는 것을 목표로 바꾸었다.

다른 파티클 효과들과 구매한 에셋의 파티클을 확인해보았을 때 유니티 상의 파티클은 prefab(프리팹)의 형태로 저장되는 것을 확인할 수 있었다.

유니티 공식 문서에서 프리팹 관련 문서를 확인할 수 있었는데, 유니티 상에서의 사용법 위주로 나와있고 내부적인 구조는 따로 설명되어 있는 문서가 없었다.

대신 프리팹 파일을 텍스트 에디터로 불러와 구조를 볼 수 있었다.

프리팹 자체는 다양한 형태의 데이터를 저장할 수 있도록 구성이 되어 있는 것처럼 보였고, 내부의 파티클 시스템 부분을 공부해 바이너리 형태로 추출할 수 있는 도구를 만들고 이를 불러 온다면 파티클 시스템에 필요한 데이터를 불러오는 부분은 해결 할 수 있을 것이라는 생각을 했다.

**주간 소감:** 가족여행과 새해이고 방학이라는 생각이 많이 들어 집중해서 많은 시간을 투자하지 못한 것 같다. 다음 주부터는 약속도 없고 다시 기숙사에 들어가서 생활하게 되므로 열심히 해야겠다는 생각을 했다.

**▶ [박준영]**

12/31부터 1/3까지 장염으로 인해 사회복무 병가내고 요양

이후 UI 개발 준비 -> 다음주까지 실제 눈에 보이는 결과 나올 수 있게

우리 게임에 필요하게 될 UI를 세부적으로 생각해볼 필요가 있었음

UI, UX 디자인을 뛰어나게 할 필요성까진 없지만 적어도 구조적/퍼포먼스 적으로 UI가 전체 게임 플레이 및 개발에 있어 방해가 되지 않게

Directx 12 기반 샘플이 그리 많지는 않지만 공식문서 및 github에서 찾은 UI 샘플 을 참조

[**https://learn.microsoft.com/en-us/windows/uwp/gaming/tutorial--adding-a-user-interface#the-user-interface-overlay**](https://learn.microsoft.com/en-us/windows/uwp/gaming/tutorial--adding-a-user-interface#the-user-interface-overlay)

DX12의 경우 UWP기반 공식문서가 대부분이나 실제 적용하는 부분은 win32api과 큰 차이가 없어 충분히 참고가능

XAML을 이용한 UI제작도 가능하다고 함, 신기하긴 하나 UWP기반에서 사용 가능한 관계로 도형, 이미지를 사용하는 부분은 Direct2D, 텍스트 출력은 DirectWrite를 적용하여 할 예정

**▶ [엄장헌]**

물리엔진의 파이프라인중 광역탐지 단계에서 충돌 가능성을 가진 쌍을 추리기 위해 옥트리를 공부 -> 다음주에 해당 클래스를 만들예정

지역 탐지 단계를 위한 교차 단계로 객체가 가진 여러개의 콜라이더를 체크하기로 결정

현대의 물리엔진이 제약기반 물리로 접근하고, 실시간 적분법으로는 반-암시적 오일러 적분법을 이용한다고 하는데 간단하게 처리할지, 이 적분법을 적용해서 할지 다음주에 공부후 생각…

에셋을 사용하기 위해 구조를 분석해봤는데 주로 프리펩을 이용해야될 것 같았다.

하지만 구조가 생각보다 복잡해서 익스포터를 찾던중, 유니티에서 제공하는 익스포터 발견-> fbx 파일로 비교적 간단한 형태로 변경되었다.  
이전에 만들어 둔 fbx로더를 이용해 불러오기에 성공

이후 바이너리 파일로 만들어 필요한 정보만 가져올 예정

