| **비단길(Silk Road)** | |
| --- | --- |
| **4주차** | **2024.01.13 (일) ~ 2023.01.20 (토)** |

**■ 주간 회의록ㅣ**

**▶ 네트워크 개발 과정에서의 문제 해결 (실제 루프의 구성)**

* 지난주차 회의결과로 서버 기반을 담당하게된 권순원 학생, 개발을 하는 중에 구조적인 문제에 봉착하게 되어 추후 작업의 편의를 회의 안건으로 요청

-> 기존 제작해본 서버와 다르게 호스트의 클라이언트에서 쓰레드로 분리되어 동작하기 때문에 호스트 클라이언트와 서버 쓰레드 간의 데이터의 이동이 너무 까다로웠음

-> 이로 인해 호스트와 게스트 플레이어의 코드 구조가 많이 달라질 수 있겠다고 생각을 하였음

* 엄장헌 학생 제안  
  게임의 업데이트문은 동일하게 진행할 수 있다.  
  처음 recv와 나중의 send부분만 호스트인지로 차이를 두어 분기점을 만든다.  
  데이터 자체는 큐를 중간에 둠으로써 교환을한다.
* 이 구조는 마치 쇼핑 중독자의 집과 같다.  
  이 사람은 엄청난 쇼핑광으로 쿠팡을 1초에 수십개를 시킨다.  
  이 사람이 택배를 개봉하다가 쿠팡맨이 와서 배달을 할때마다 하던걸 놓고 현관에 나가서 박스를 들고오는 것 보단, 까던 박스들을 다 까면 현관에 쌓인 박스들을 한번에 가져오자.  
  여기에서 이 택배 상자들을 보관해두는 현관이 우리 게임 루프에서의 큐와 같고 우리는 뮤텍스를 이용하여 Queue를 아주 손쉽게 쓰레드간 충돌없이 이용 할 수 있는 Lock기반 Queue 구조를 사용하려고 한다. (오늘 회의의 핵심)

**▶ 네트워크 세부적으로 어떻게 구현 할 것인가? (서버는 무엇을 하는가)**

* 게임 루프는 호스트나 게스트나 거진 동일하게 돌아감. 왜냐면 서버가 1초에 60번의 연산을 한다고 하더라도 이 연산에 의존하여 클라이언트가 돌아가게 되면 엄청난 문제가 생김. 1억원 짜리 컴퓨터를 써도 60프레임밖에 안나온다.
* 그래서 물리적인연산 (속도에 따른 물체의 위치변환, 가속도에 따른 속도변화) 이런건 호스트/게스트 둘다 각자 클라이언트에서 시뮬레이션 함.
* 그런데, 같은 게임을 하는 플레이어들이 각자 알아서 게임월드를 독립적으로 돌려버리면 동기화 측면에서의 문제가 발생 가능 하다. (분명 같은 맵에서 게임을 하는데 다른 결과를 볼 수 있다는 의미)
* 게임루프가 각자 돌아가는데 3차원상의 물리등을 시뮬레이션 하다보면 미세한 렉이나 오차등으로 인해 나비효과로 오차가 점점 커질 수 있음. 그렇기에 우리는 중간중간에 호스트가 갖고있는 데이터를 기반으로 게스트들의 씬내 오브젝트들이 올바른 값을 가졌는가를 체크한다.  
  이는 마치 답안지로 학생들이 자신의 시험지 채점하는 것과 유사하다. 만약 아주 사소한 (게임 결과에 유의미한 변화를 주지 않을정도) 오차까지 매번 덮어써버리면 (예: x좌표가 실제로 11.2 인데 게스트에선 11.19987인 경우) 필요 이상으로 버벅일 수 있다. 그렇기에 특정한 bias 값을 지정하여 이 이상의 오차가 벌어진 경우에만 게스트가 자신의 객체의 속성을 수정하도록 한다.

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 서버 쓰레드 코드 작성
* 회의록에 기반해서 다른 팀원이 작업할 수 있는 네트워크 기반 작성

**▶ [박준영]**

* 네트워크 로직 검수 및 제안
* 2d 관련 엔진 기능 추가

**▶ [엄장헌]**

* 물리엔진에서 강체끼리의 충돌 시 물리량 변화 작성

**■ 작업 일지**

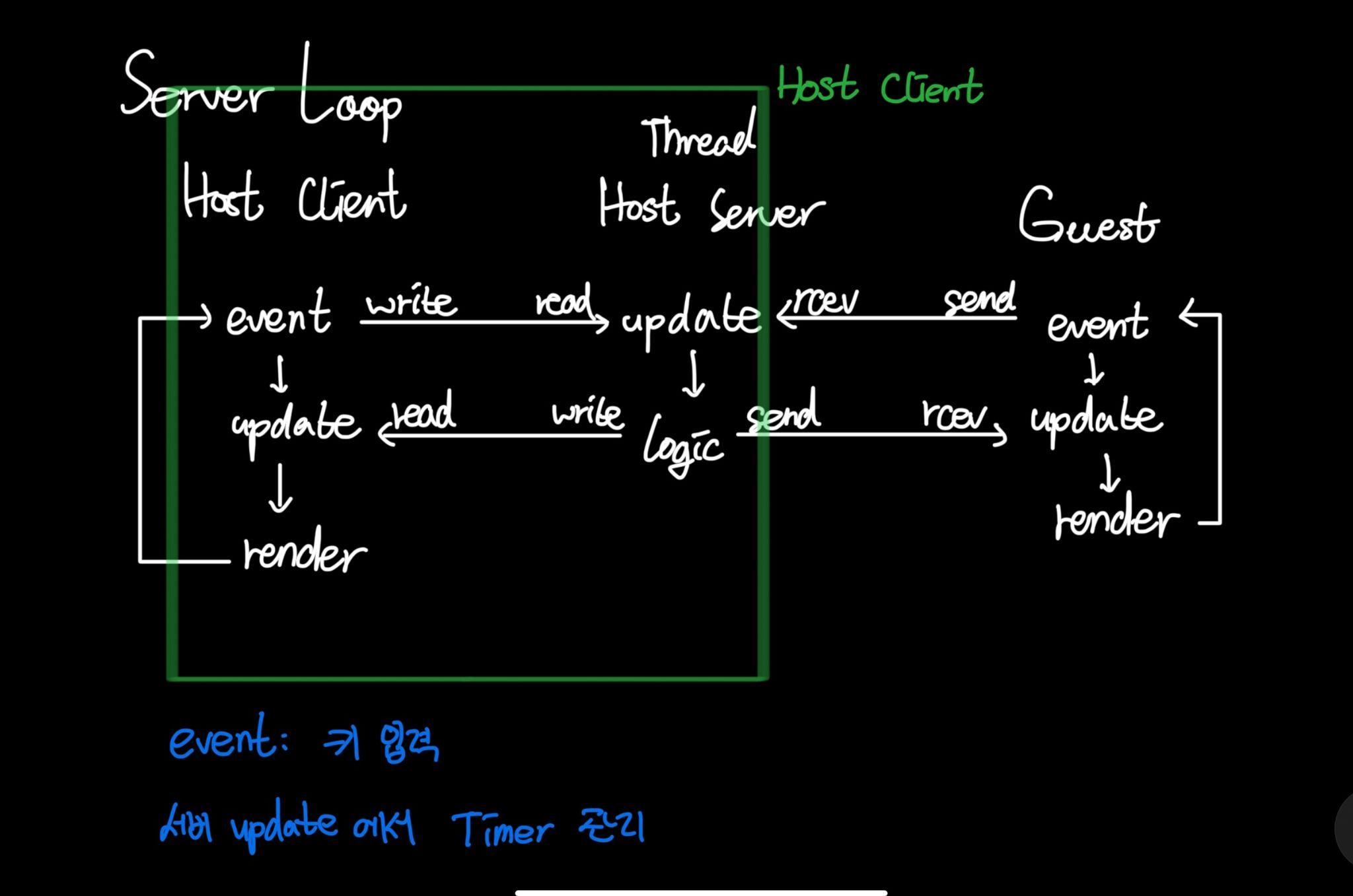
**▶ [권순원]**

서버 관련된 코드 생성

일단 호스트 역할을 할 때 사용될 Server 클래스와 호스트와 게스트 사이에 통신에 사용할 PacketManager(PM) 클래스 두 가지를 생성했다.

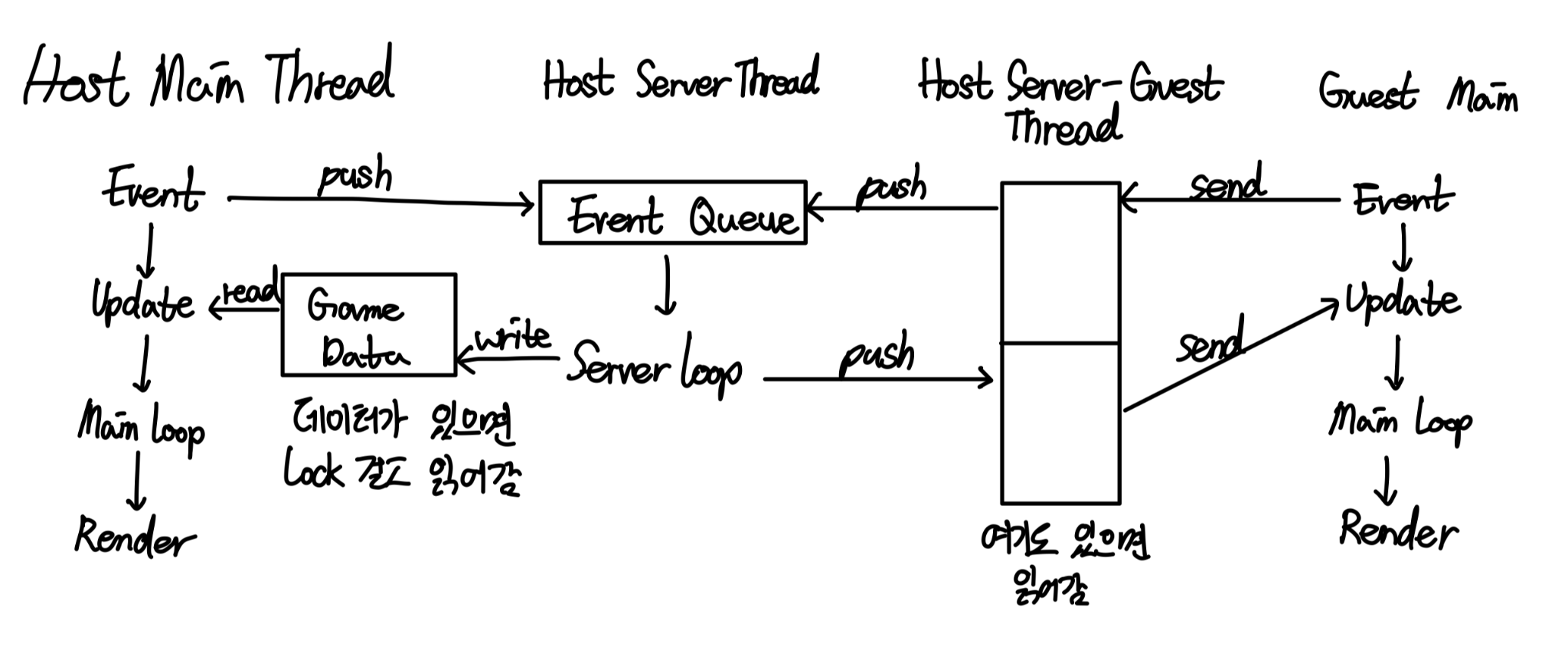
Server는 기본적으로 클라에서 게임이 시작되면 스레드로 같이 생성된다.

1. 호스트의 경우 계속 Server 스레드를 통해 데이터를 계산 및 주고받고, 게스트가 참여하면 PM을 통해 데이터를 주고받는다.
2. 게스트의 경우 호스트의 서버 코드를 받아 원래 동작하고 있던 서버를 호스트의 서버로 교체하고 해당 서버와 PM을 통해 데이터를 주고받는다.

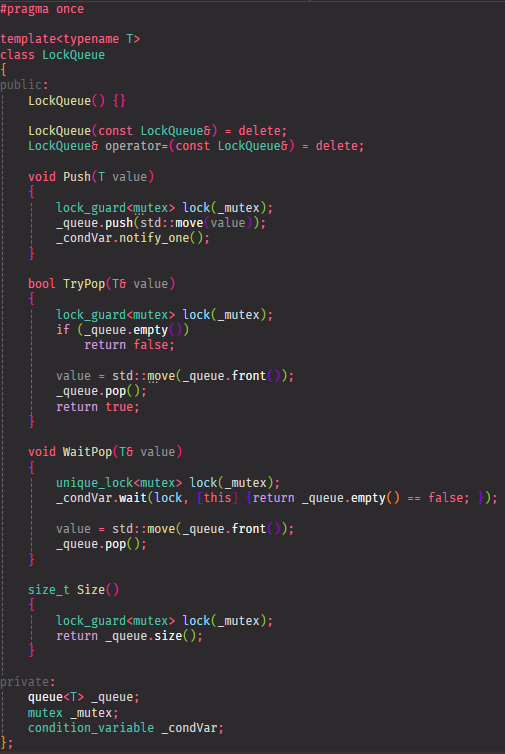
일단 기본적으로 두 클래스 다 스레드로 동작해야 하는 것이기 때문에 멤버함수를 스레드로 실행시켜 사용하는 것으로 해야 될 것 같은데, 함수 별도의 함수로 스레드를 동작하게 만들었기 때문에 이 부분을 공부해야 할 것 같다.

작업 중에 헷갈리는 부분이 있어서 작업에 편의를 위해 간단한 흐름도를 그려보았다.

그림을 그리면서 호스트의 메인 루프(게임 업데이트 및 랜더링) 부분에서 이대로 진행해도 괜찮은가에 대한 의문이 생겼고 네트워크 프레임워크 구성에 대해 다른 팀원과 상의가 필요하다고 생각했다. 주간회의 안건으로 올렸다.

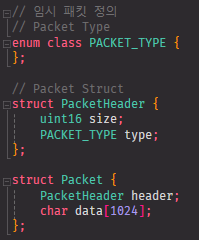


주간회의 이후 팀원과 상의한 내용을 기반으로 다시 간단한 흐름도를 그려보았다.

기본적으로 모든 데이터는 LockQueue를 기반으로 하여 송수신/공유가 되도록 만들 것으로 결정했기 때문에, 프로젝트에 LockQueue를 추가했다.

넷겜프 팀플에서 사용했던 LockQueue를 기반으로 제작을 했다. 규모가 더 큰 경우에는 다른 동기화 방식을 사용한다는 이야기를 보았지만, 우리 게임은 P2P 방식이고 최대 3인이 플레이 하기 때문에, 3인 플레이인 팀플에서 사용한 락큐를 그대로 사용하기로 하였다.

함수 내부를 모두 작성하기 전에 필요한 함수나 변수를 정리해 만들어 놓는게 개발에 있어 편리할 것 같아 서버를 만드는데 사용할 함수와 변수를 정리해놓았다.



추가로 패킷 관련해서 생각을 해보았다.

이전에 서버를 만들 때는 주고받을 데이터가 많지 않았기 때문에 float 2개와 byte 하나를 비트마스킹해서 패킷으로 사용했는데, 이 게임에서는 주고받을 데이터의 양과 종류(키 입력, 시간, 몬스터 정보, 플레이어 정도 등등…)가 다양할 것이라고 생각해 헤더를 하나 작성해 그 크기와 종류를 정의하고 이에 따라 패킷을 주고 받는 것을 생각해봤다. 서버 프레임워크 작업이 끝나면 회의를 통해 결정해야 될 것 같다.

**▶ [박준영]**

아쉽게도 이번주 근무시간 대부분동안 제설작업에 투입된 관계로 작업량이 많지는 않았다.

2D - 사실상 우리 프로젝트에 개발하는 것 보다 유니티 디자인 패턴에 대한 공부 시간이 더 많았던 것 같다. 이왕이면 컴포넌트 패턴의 원칙에 기반한 완벽한 canvas 와 그에 대한 하위 구조를 만들어내고 싶지만 우리 엔진이 유니티와 완전 동일하게 컴포넌트 패턴을 구현한 것이 아니다 보니 어느정도 맞춤 작업이 필요할 것 같다.

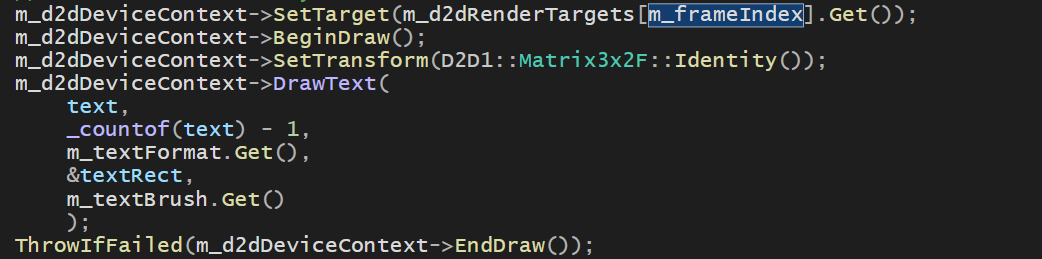
일단 브랜치도 파서 작업하는 김에 그냥 하드코딩이라도 구현부터 하고 리팩토링 하는 형태로 진행해봐야 될 것 같다.

여기에서 ui의 레이아웃 형성하는 문제에 대해 고민을 해보았다. 유니티의 경우 2d ui의 오브젝트들을 배치할 때, 피벗 등을 이용하여 이용하는 디바이스의 화면 비율, 해상도에 맞게 자동 조절되는 기능을 제공한다.

이것이 가능하고 불가능 하고의 구현 난이도나 구조가 상당한 차이를 보일 것으로 생각되어 해상도 변경 기능에 대해 고민을 해보았다.

기본적으로 현재 pc게이밍 시장에서 가장 보편적으로 쓰이는 16:9 화면 비율을 기반으로 하여 1280\*720, 1920\*1080, 2560\*1440 등의 자주 쓰이는 해상도를 선택할 수 있는 옵션을 구현해 보려고 했다. 여기서 문제가 발생할 여지는 화면 비율 부분인데 최근 게이밍 노트북들이 위아래로 더 넓어진 16:10의 화면을 사용한다거나 21:9의 화면비율을 지원하는 ultra wide 모니터들을 사용하는 유저들 또한 늘어나는 추세여서 이러한 화면비를 제공하는 과정이 필요한가 고민하였다.

조사 해보니 유니티는 기본적으로 pc 외에도 다양한 모바일 디바이스 제공을 염두하다 보니 자연스럽게 제각기 다양한 화면비와 해상도를 지원하는 모바일 디바이스들에 맞게 ui 레이아웃을 변화시키는 로직이 필요한 것이며 우리 게임에는 거기까지 필요 없을 것으로 보여 그냥 내가 직접 해상도별로 레이아웃을 조정하기로 했다.



디바이스 세팅등을 제외하면 텍스트 드로우 자체는 매우 간단한 작업이었다.

마치 win32api에서의 draw를 연상시킨다.

**▶ [엄장헌]**

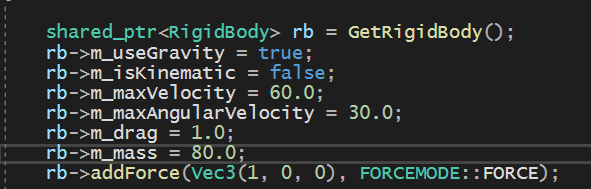
3박 4일의 여행과 건강 이슈로 진도를 별로 나가지 못했다.

우선 rigidBody클래스에 drag, angularVel, angularVelDrag, isKinematic 변수를 추가했다. 각각 공기저항, 각속도, 각속도 공기저항, 물리적인 영향을 받을지에 대한 변수이다. 공기저항에 경우 깃털과 금속의 움직임에서 차별점이 필요하다. 0일 경우 공기저항이 없는 상태 무한대로 설정시 즉시 정지된다. 일반적인 설정값은 0.001의 경우 단단한 금속 덩어리이고, 10의 경우 깃털을 표현할 수 있다. isKinematic 변수의 기본값은 false이며, useGravity의 기본값은 true이다.

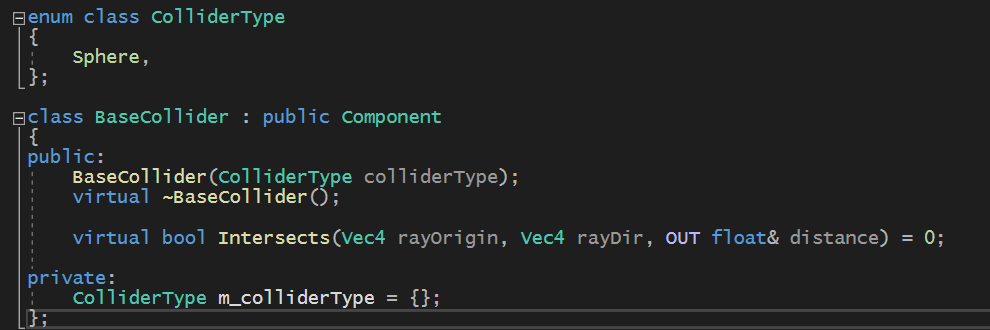
각속도와, 각속도 공기저항은 회전에 관한 성질이다. 저번주에 짜둔 중력함수와, 움직임 함수에 해당 변수들이 반영되었다.

그 다음으로 addForce의 경우 4가지 타입이 있다. Force, Acceleration, Impulse, VelocityChange로 각각 질량사용 지속적인 힘, 질량무시 가속도 추가, 질량사용 즉각적인 힘, 질량무시 속도 추가 이다. 일단 즉각적인 물리량 변화의 경우 간단하게 구현할 수 있었다. 지속적인 힘에 경우는 코드를 사용하는 사람이 지속적으로 호출되는 코드에 사용한다. 유니티에 경우는 fixedUpdate를 통해서 0.02초마다 호출되는 함수에 사용하는 경우가 많으나 우리의 코드에는 그런 update문이 존재하지 않는다. 이를 구현하는 것보다는 1초동안 가하길 원하는 force량을 사용하는 것이 더 좋을거라 생각했다.

사용 예시는 각 monovihaviour 코드에서 다음과 같이 변수를 세팅하고 사용한다.



그 다음으로 collider의 경우 baseCollider를 기본으로 만들고 타입을 정해 여러 종류의 collider 컴포넌트를 만들 계획이다.



이를 기반으로 구체, 직육면체, 원기둥 등의 collider를 만드는 것이 목표이다.