포트폴리오

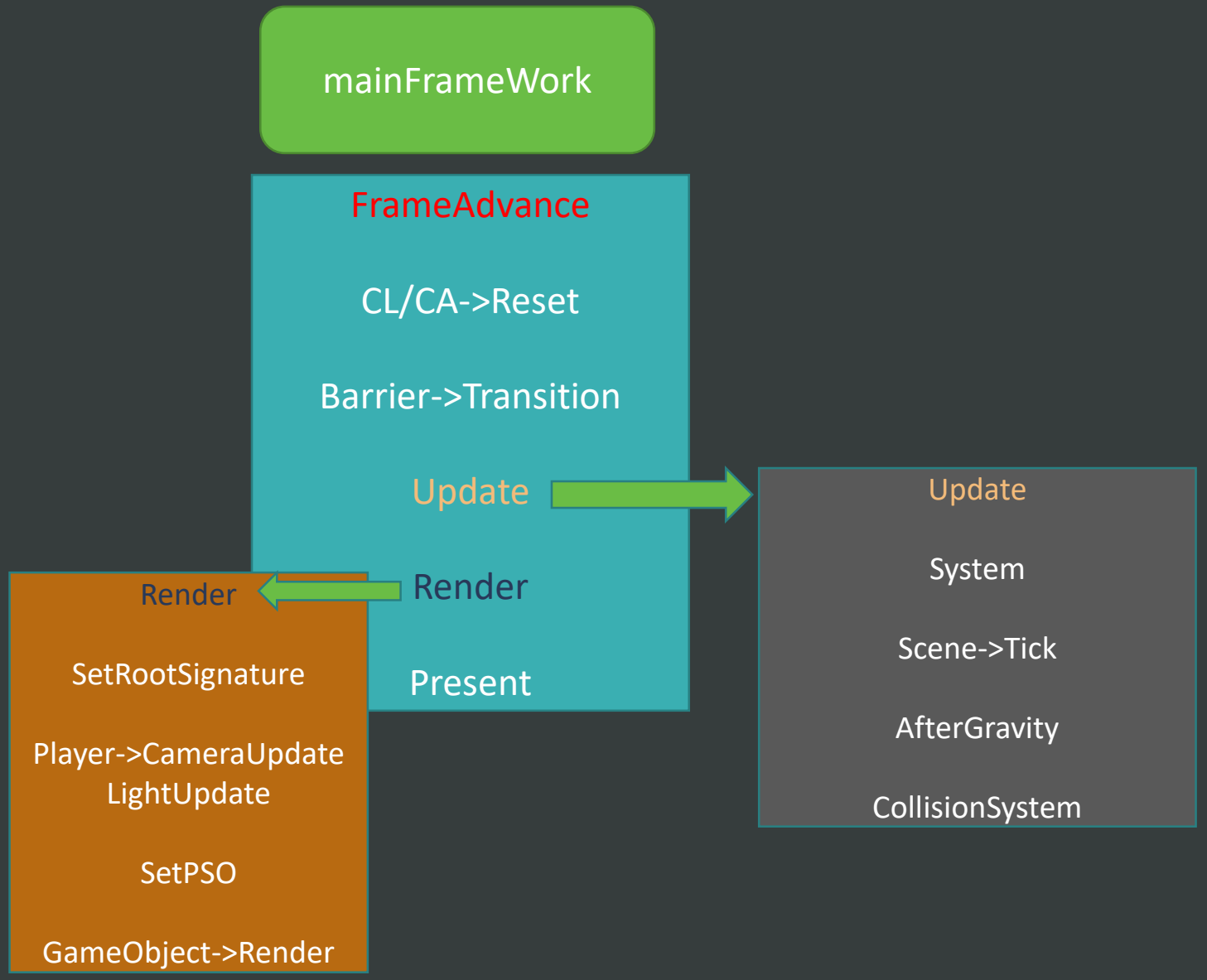
이관구

목차

- Direct X / Open GL
  - 매직 큐브 [메인 포트폴리오 Directx 12]
  - 타워 디펜스 (Directx 11)
  - 아마추어 베이스볼(OpenGL 구버전)

- 유니티
  - 봄버맨 모작
  - 트리 트리 (핀볼 모작)
  - 던전 가드 (SRPG 프로토 타입)

- 게임 이름 : 매직 월드
- 사용 언어 : C++ / DirectX 12
- 장르 : TPS
- 제작 기간 : 8 개월
- 제작 목표 : 다이렉트X12를 이용한 5인 TPS 게임 제작
- 제작 인원 : 3인
- **깃 허브:** https://github.com/ChangminYoo/MagicWorld
- **키워드 : DirectX12 / 리지드바디 구현 / 5인 슈팅게임 / 클라이언트**
- **주요 기술 적용 : 질점 과 리지드 바디를 직접 구현 하여 물리효과를 적용, 안개효과, 애니메이션 구현 , 노멀 매핑 구현 , 몬스터 FSM 구현 , Boost를 이용한 서버 구현**
- 맡은 파트 : 클라이언트 프레임워크 공동 제작 / 질점과 리지드바디 구현 / 애니메이션 구현 / 안개효과 구현 / 파티클 구현 / FSM 구현 / 스킬 4개 구현
- 게임 설명 : 5명이 즐기는 TPS 방식의 개인전 슈팅게임 입니다. 8개의 마법 중 캐릭터마다 4개 씩 다른 조합으로 부여 되며, 플레이어를 죽이거나, 몬스터를 잡으면 점수를 획득 합니다. 게임 시간이 다 지나면 가장 높은 점수를 가진 플레이어가 승리합니다.



- 클라이언트 코드 핵심 요소

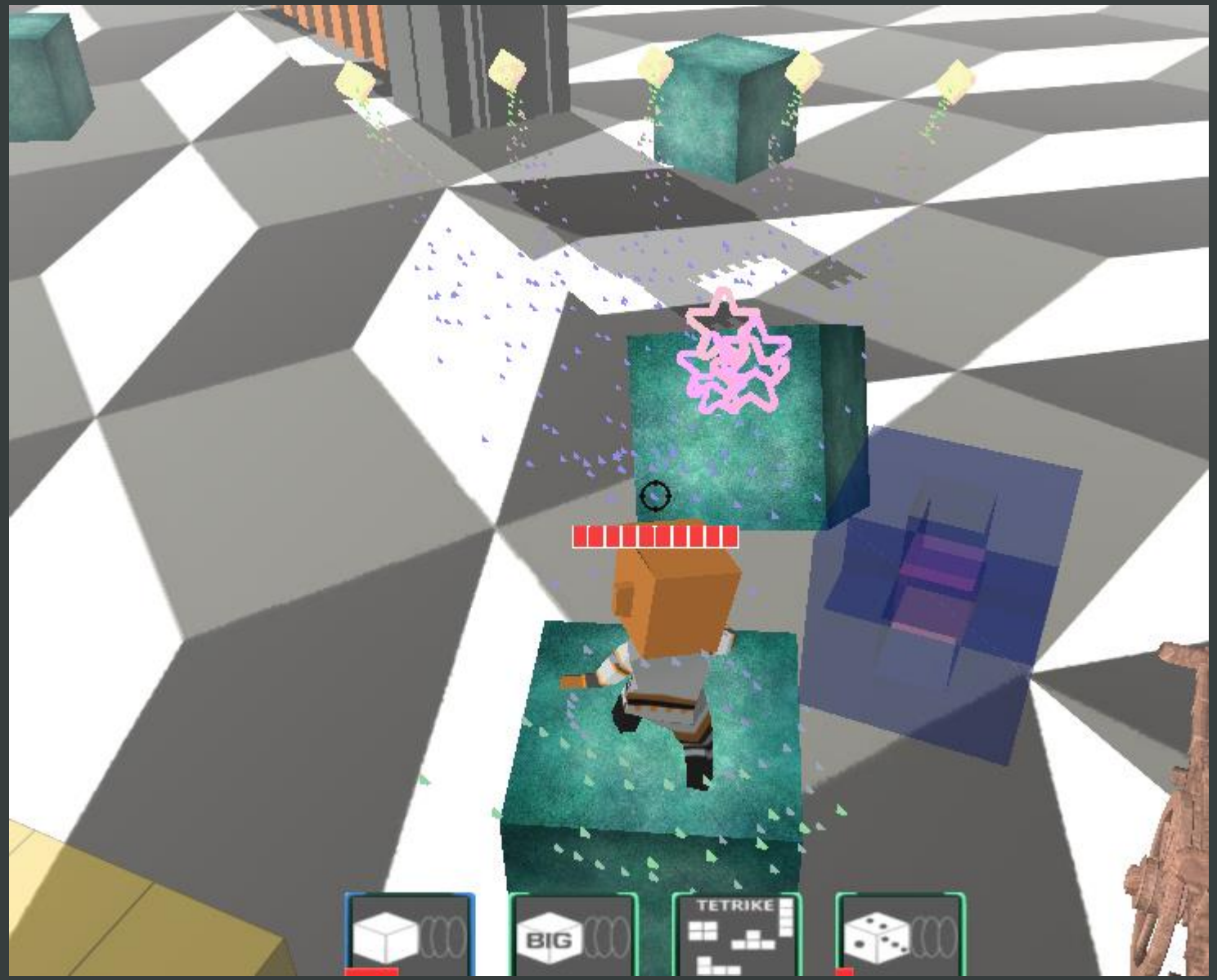
1) FrameWork – 디바이스,커맨드리스트, 커맨드 큐, 스왑체인, 펜스 등 기본적인 Direct용 인터페이스를 생성합니다.

2) mainFrameWork – FrameWork를 상속받았으며, FrameAdvance라 불리는 매 프레임마다 호출되야 하는 함수를 가집니다. FrameAdvance는 커맨드 리스트와 커맨드 알로케이터를 초기화 시킨 후, 후면 버퍼를 리소스 베리어를 사용해 전이 시킵니다. 뷰포트와 시저렉트를 연결 후 Update함수에서 중력 / Scene클래스의 Tick / 충력 후 처리 / 충돌 처리 를 합니다. 이후 객체들을 렌더링 합니다.

3) Scene – 모든 게임객체를 스테틱/다이나믹/불렛/리지드바디/UI 등으로 나눠서 리스트로 저장을 하며, 게임이 종료 시 반드시 모든 게임객체를 지우는 역할을 합니다. 내부적으로 Shader객체 또한 가지며, 루트 시그니처를 포함합니다. 매 프레임마다 Tick함수를 호출하며, 제거되야 할 객체를 제거하고, 게임 객체들의 Tick을 호출해서, 물리처리 등을 합니다. 이후 렌더링 함수를 호출하는데, 루트 시그니처 – 카메라/광원 연결 – 객체 별 PSO 연결 - 월드 행렬 갱신 및 연결 – 텍스처 연결 – 렌더링 과정을 거칩니다. 단 이때 최적화를 위해 카메라 뒤에 있는 객체는 렌더 대상에서 제외 됩니다.

4) Shader 클래스 – 실제 쉐이더 파일이 아니라 그려져야 할 객체에 맞는 PSO를 모두 저장하고 있습니다. Scene이 가지고 있는 객체의 모든 리스트를 포인터로 공유합니다.

5) GameObject 클래스 – 모든 게임오브젝트의 조상 클래스 이며, 모든 게임오브젝트는 해당 클래스를 상속받습니다. 위치 정보, 방향 정보, 게임데이터, 애니메이션에서 사용될 버퍼, 룩/업/라이트 벡터 등을 가집니다.



- 핵심 파트 설명 [클라이언트]
- **물리엔진의 리지드바디 , 질점 직접 구현**
- 사진에 보이는 청록색 큐브들은 모두 강체들이 적용 되어있으며, 투사체들은 모두 질점 객체로 구현되었습니다.
- 주요 클래스 및 함수
- 1) Class – GeneratorGravity : 리지드바디와 질점 객체에 중력을 매 프레임마다 가하도록 인터페이스를 제공
- 2) Class – RigidBody : 속도/가속도/중점/방향/질량/토크/힘 등 리지드바디의 정보들을 담고 있으며, 적분함수를 통해 위치와 방향정보를 갱신합니다.
- 3) Class –PhysicsPoint : 리지드바디와 비슷하지만 방향의 변화를 처리하지 않으므로 각속도 등을 변경치 않습니다. 플레이어 객체나, 탄환, 고정된 객체에 적용됩니다.
- 4) Method – AfterGravitySystem : 중력에 대한 후처리를 합니다. 물체가 땅에 파고들거나, 리지드바디 객체가 바닥에 부딪혀서 튕기거나 하는 것 을 처리합니다.
- 5) Method – CollisionSystem : 물체와 물체의 충돌을 처리 합니다. 충돌 후 속도와 겹쳤을 때 처리 등이 일어납니다.

```
//그후 사잇각이 특정각도 이하면 보정시킨다.
//단 이게 double로 해도 0이아닌데 0이나오는경우가 생긴다.
//따라서 0일경우 그냥 충격량을 가해서 각도를 변경시킨다.

if (abs(stheta) <= MMPE_PI / 20 && abs(stheta) != 0 &&
    abs(impurse) < obj->rb->GetMaxImpurse() && obj->rb->AmendTime <= 0)
{
    //회전축을 구하고..
    XMFLOAT3 mAxis = XMFloat4to3(Float4Cross(sV1, sV2));
    mAxis = Float3Normalize(mAxis);
    //보정을 시킨다.
    AmendObject(mAxis, stheta, obj);

    //그리고 재귀 시킨다. 왜냐하면 보정이되었으면
    //allpoint,tempcollisionpoint,contactpoint , penetration 모두 다 바뀌어야 하기 때문이다.
    //재귀 후 아마 2가지 경우의수가 있다. 충돌이 일어나거나, 아니면 살짝 떠있거나..
    //어쨌든 잘 해결 된다.
    obj->rb->SetAngularVelocity(0, 0, 0);

}
else if (abs(theta) <= MMPE_PI / 25 && abs(theta) != 0 &&
    abs(impurse) < obj->rb->GetMaxImpurse() && obj->rb->AmendTime <= 0)
{
    //회전축을 구하고..
    XMFLOAT3 mAxis = XMFloat4to3(Float4Cross(V1, V2));
    mAxis = Float3Normalize(mAxis);
    //보정을 시킨다.
    AmendObject(mAxis, theta, obj);


    obj->rb->SetAngularVelocity(0, 0, 0);
```

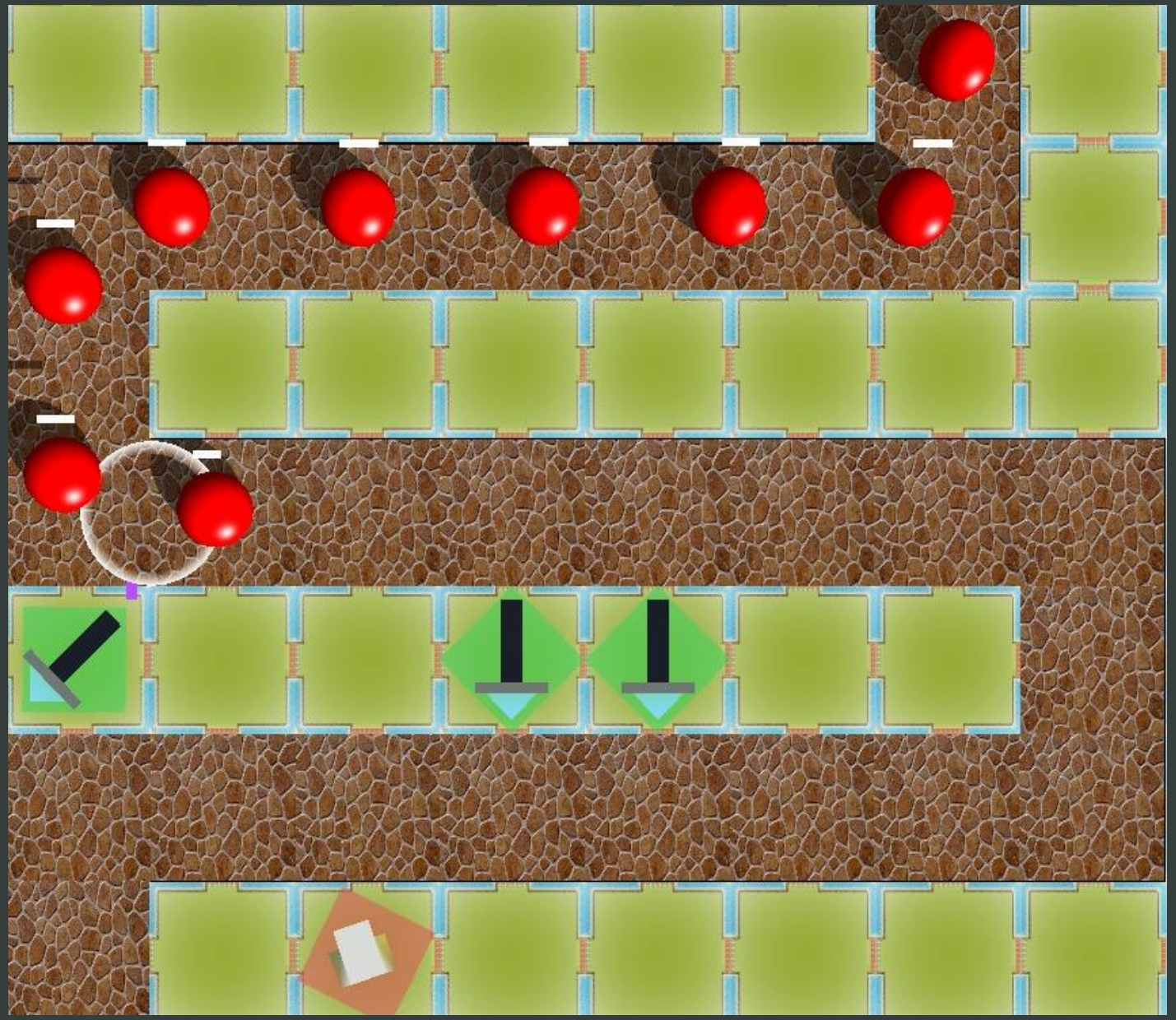
```
void AmendObject(XMFLOAT3 axis, float radian, CGameObject* obj)
{
    XMFLOAT4 q = QuaternionRotation(axis, radian);
    obj->Orient = QuaternionMultiply(obj->Orient, q);
    obj->UpdateLookVector();
}
```

```
//Jm = J/M
//임펄스의 비율을 나눈다.  즉, 일반적인 1:1 관계에서 1.3: 0.3 정도로 둔다.
//나머지 0.7은 소실된 에너지라 치자.
if (obj->rb->AmendTime <= 0)
{
    auto ratioImpurse = impurse * 0.3;
    auto Jm = Normal;

    Jm.x *= obj->rb->GetMass()*(impurse + ratioImpurse);
    Jm.y *= obj->rb->GetMass()*(impurse + ratioImpurse);
    Jm.z *= obj->rb->GetMass()*(impurse + ratioImpurse);



    //각속도 계산
    //W = 기존 각속도 + ((Q-P)Ximpurse)*Inversel
    auto W = obj->rb->GetAngularVelocity();
    XMVECTOR rxi = XMLoadFloat3(&XMFloat4to3(Float4Add(fp.Pos, obj->CenterPos, false)));
    rxi = XMVector3Cross(rxi, XMLoadFloat3(&Normal));
    rxi *= (ratioImpurse);
    rxi = XMVector3Transform(rxi, XMLoadFloat4x4(&obj->rb->GetIMoment()));

    XMFLOAT3 ia;
    XMStoreFloat3(&ia, rxi);

    W = Float3Add(W, ia);
    W = Float3Float(W, obj->rb->GetE() / 2);
    XMFLOAT3 lastvel = obj->rb->GetVelocity();
    obj->rb->SetVelocity(Float3Float(Float3Add(lastvel, Jm), obj->rb->GetE()));
    obj->rb->SetAngularVelocity(W);
}

//이제 속도와 각속도는 변경 했으니, 겹쳐진 부분 해소
//가장 작은값의 penetration(가장 깊은)만큼 올리면 된다.
auto px = fabsf(contactpoint[0].penetration)*Normal.x;
auto py = fabsf(contactpoint[0].penetration)*Normal.y;
auto pz = fabsf(contactpoint[0].penetration)*Normal.z;
obj->CenterPos.x += px;
obj->CenterPos.y += py;
obj->CenterPos.z += pz;
```

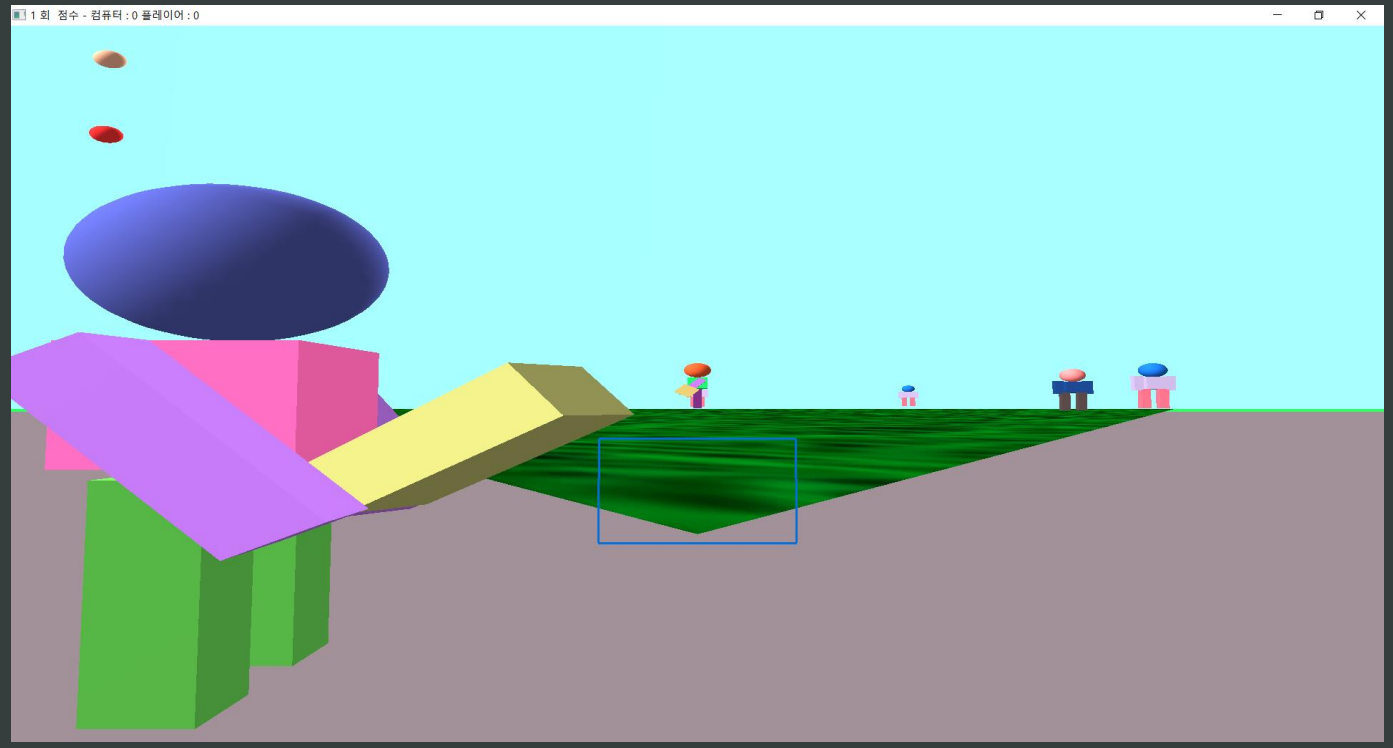
- 물리엔진 처리시 어려웠던 점
- 처리하면서 어려웠던 점이 한두개는 아니지만, 그중 기억에 가장 남는 것은 바닥과의 육면체 충돌 시 처리를 하는 RigidBodyCollisionPlane 함수입니다.
- 이 함수는 바닥과 육면체(OBB)를 평면의 방정식을 이용해 해당 점이 평면의 뒤에 몇 개의 점이 있는지를 검사한 후, 그 점의 개수에 맞게 충돌 처리를 합니다.
- 만약 충돌을 했다면 점은 1/2/4 개의 점을 가질 수 있습니다.
- 먼저 4개일 때는 토크를 가하지 않아도 되기 때문에, 선속도만 바꾸는 처리를 하였습니다.
- 문제는 1개와 2개일때의 처리였는데, 그중 유독 2개일 때 처리가 상당히 고달팠습니다.
- 충돌을 한 점들을 더해 2로 나눈 중점 을 이용해 그곳으로 충격량 을 가한 것으로 처리를 하게 되면, 탭댄스를 추는 문제가 일어났습니다.
- 이러한 탭 댄스 문제 처리 때문에 상당히 골머리를 앓다가, 보정이라는 영역을 추가 하였습니다.
- 보정은 특정 각도 이하이면서, 충격량 이 크지 않은 경우, 해당 물체를 땅에 닿게 고정시키면서, 회전을 하지 않게 하였습니다.

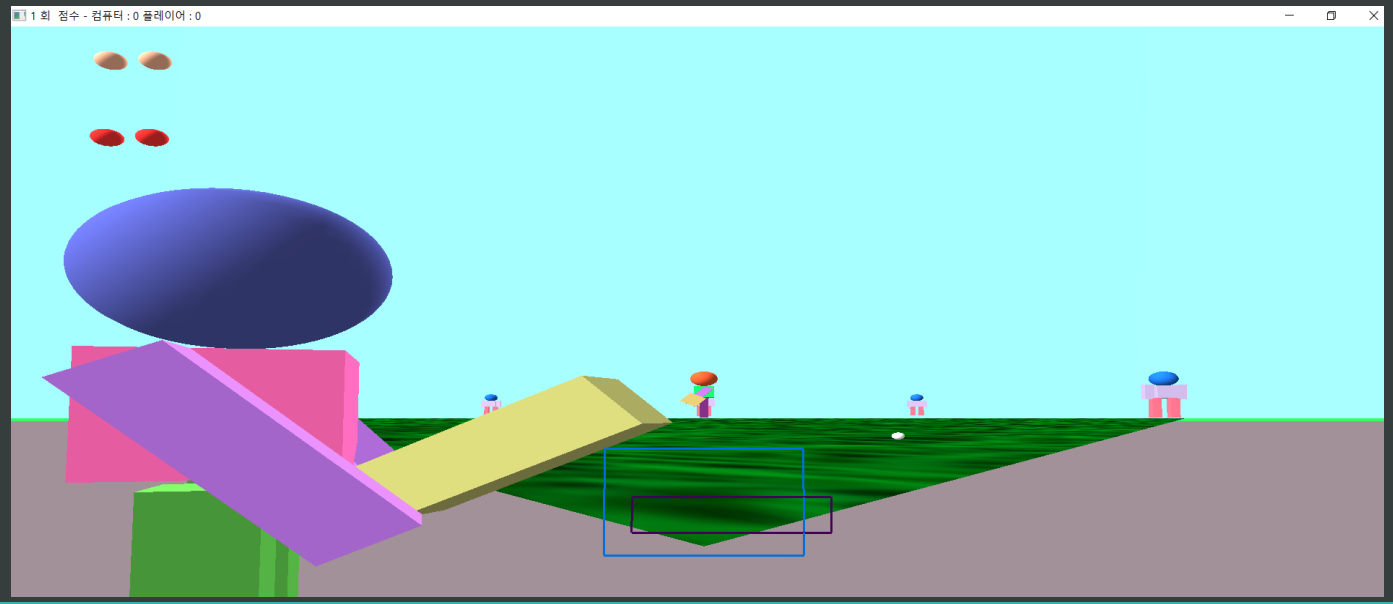
```
XMFLOAT3 temp;
XMStoreFloat3(&temp, XMVector3Dot(sik1, N));

float t = temp.x;

//이제 t를 이용해 교점X가 유한평면 안에 있는지 검사한다.
//이후 유한평면안에있으면서 dir와 testP-playerpos를 내적시 0보다 크거나 같으면
//mint를 갱신한다.SavePos도 갱신한다. 아니라면 continue

auto X = Ro + Rd * t;
//X가 6개의 평면에서 하나라도 0보다 큰게 있으면 바깥에있는것이며 충돌안한것이다.
bool In = true;

for (int j = 0; j < 6; j++)
{
    XMVECTOR testN = XMLoadFloat3(&Plane[j].Normal);
    XMVECTOR testP = XMLoadFloat3(&Plane[j].DefaultCenterPos);
    auto testv = XMVector3Dot(testN, (X - (testP + objpos)));
    XMFLOAT3 result;
    XMStoreFloat3(&result, testv);

    if (result.x > 0)//0보다 크면 바운딩박스 안에 없다.
    {
        In = false;
        break;
    }

}
```

- 물리엔진 처리시 어려웠던 점 2
- 또다른 어려웠던 점으로 피킹 처리 하는 RayCasting 함수입니다.
- 저는 먼저 레이캐스트를 위해 RayCastObject 라는 클래스를 만들었습니다. 이곳에는 6개의 면과, 기본 피벗이 되는 DefaultCenterPos란 데이터가 들어가 있습니다.
- 이는 육면체를 이용해 레이캐스트를 하기 위해서 입니다.
- 고려 해야 할 게 생각보다 좀 많았습니다.
- 먼저 인자로, 광선에 대한 정보와 플레이어, 검사 대상 등을 받았습니다.
- 다음으로 반복문을 돌면서 6개의 평면을 검사하여, 직선위의 한점과 평면위의 한점을 사용해서, 교점을 얻었습니다.
- 이때 육면체의 경우 교점이 하나가 아닐 수 있기 때문에, 다음과 같은 처리를 추가적으로 하였습니다.
- 1) 해당 교점이 6면 중 하나라도 앞에 있으면 안된다.
- 이유 : 육면체 범위를 벗어났다는 의미
- 2) 반드시 플레이어와 카메라 사이의 거리보다 길어야 한다.
- 이유 : 플레이어보다 뒤에 있을 교점인 경우, 투사체가 뒤로 날아갑니다.
- 3) 기존 교점보단 카메라와의 거리가 가까워야 한다.
- 이유 : 가장 가까운 교점을 찾으므로.
- 이러한 방식으로 레이캐스트를 처리 하였습니다.



- 게임 이름 : 타워 디펜스
- 사용 언어 : C++ / DirectX 11
- 장르 : 디펜스
- 제작 인원 : 1인
- 제작 기간 : 2 개월
- 제작 목표 : 다이렉트X11 기본기를 위한 게임 제작
- **키워드 : DirectX11 / 기본기**
- **주요 기술 적용 : 마우스 피킹 적용**
- 게임 설명 : 6가지의 타워를 빈 타일에 설치하고 몰려오는 적들을 막는 게임입니다. 스테이지는 총 20 스테이지고 각 스테이지마다 몬스터의 외형은 그대로 지만 색상과, 이동속도 및 체력 등을 달리 처리하였습니다.
- 제작 하며 느낀 점
- 당시 제작하면서 가장 문제였던 것은, 마우스 피킹의 처리였습니다. 뷰포트 에서 역변환을 통해 이동하는 개념 자체를 잘 잡지 못해서 만들다가 책을 더 많이 본 게 기억이 납니다.
- 이 게임을 만들 때 모델임포트의 방식을 배운 적이 없어서 얇은 쿼드 객체처럼 만들어서 사용했습니다.





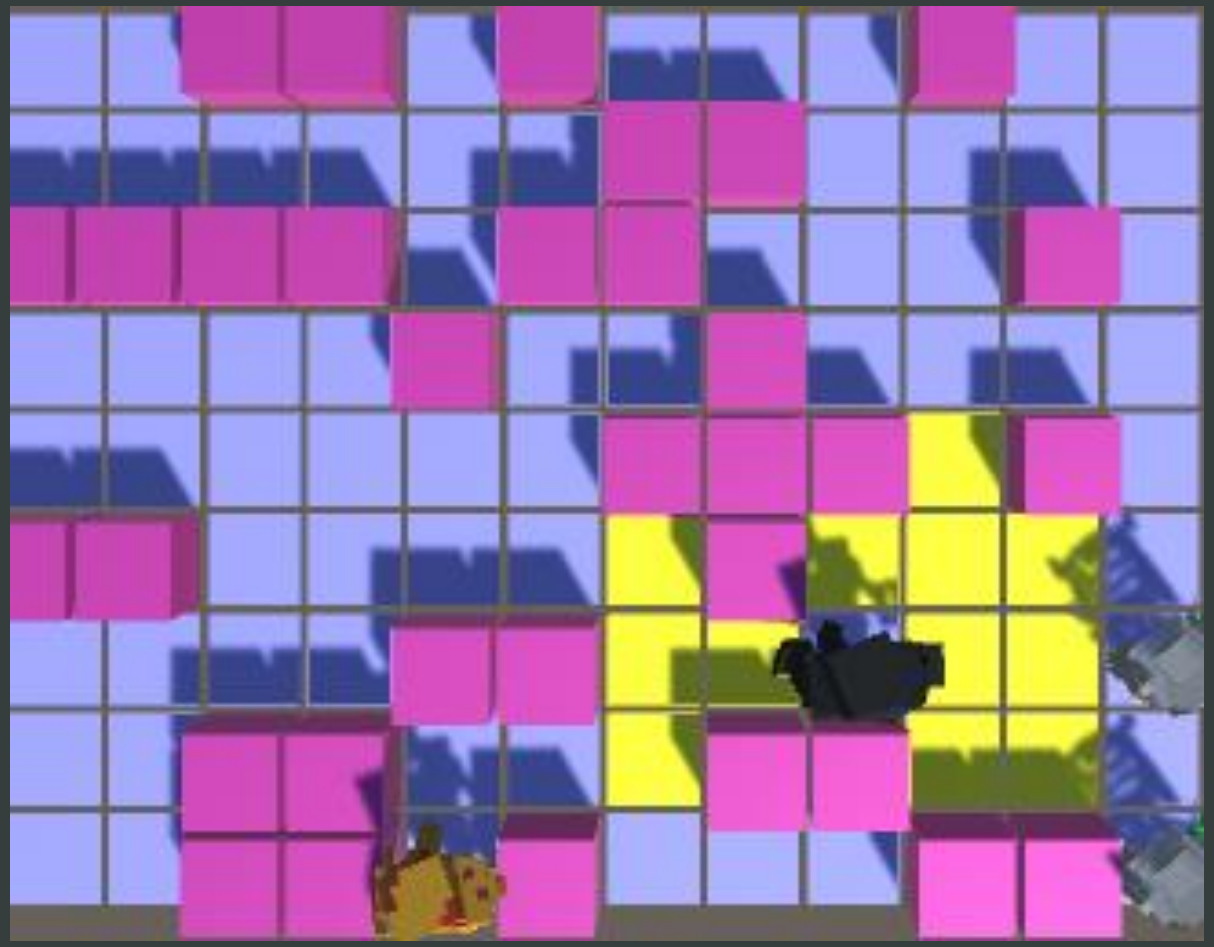
- 게임 이름 : 아마추어 베이스볼
- 사용 언어 : OpenGL
- 장르 : 스포츠
- 제작 인원 : 1인
- 제작 기간 : 2 개월
- 제작 목표 : 3D 개념을 이해하고 야구 게임을 만들어 보기
- **키워드 : OpenGL 구버전 / 첫 3D게임**
- **주요 기술 적용 : 간단한 물리 적용, 어설픈 AI 처리**
- 게임 설명 : 2학년 때 처음 만들어 본 3D 게임입니다. 매 회 마다 투수와 타자 조작을 바뀌며, 어설프지만 슬라이더, 커브, 직구를 구현했습니다. 타격의 경우 그냥 타격 범위 안에 있을 때, 타이밍 맞게 누르면 공과 그 사각형 중심과 비교해 날려 보내도록 했습니다. 타자가 치면 수비수들이 알아서 달려가서 잡게 했습니다.
- 제작 하며 느낀 점
- 첫 3D 게임이고, 2번째 만들어 봤던 게임입니다. 당시 프레임의 개념도 모르고, 인공지능 처리 등도 잘 몰라서 if문 덩어리로 만들었습니다. 버그도 많고 외적, 내적들도 어설프게 알고 있어서 파울라인을 처리할 때 고민했던 게 기억이 남습니다. 그래도 당시에 알고 있는 것을 쥐어 짜고 밤새면서 만들 던 기억이 납니다. 이것을 만들면서 가장 크게 배운 것은 간단한 물리 개념을 배운 것입니다.

- 게임 이름 : 봄버맨 모작
- 사용 언어 : C# / Unity
- 장르 : 아케이드
- 제작 인원 : 1인
- 제작 기간 : 1 개월
- 제작 목표 : Unity2D를 이용해 고전 게임 제작.
- **키워드 : Unity / 랜덤 맵 생성 / 2D**
- **주요 기술 적용 : 무작위 장애물로 랜덤한 맵 구현 , 폭탄이 주변 폭탄과 연쇄 폭발하기 , 아이템 구현**

- 게임 설명 : 봄버맨의 모작으로, 플레이어는 차근차근 폭탄을 이용해 풀숲을 불태워서 탈출 점 으로 들어가야 합니다.
- 제작하며 느낀 점
- 유니티 2D를 이용해 처음 만드는 2D 게임 이었습니다. 레이어 처리나 충돌처리, 불꽃 생성 처리 및 애니메이션 처리 방법을 배웠습니다.
- 어려웠던 점 중 플레이어가 폭탄을 설치한 경우 에는 폭탄을 무시하고 지나갈 수 있어야 하지만, 그 폭탄 위를 벗어나는 순간 더 이상 지나가면 안되는데, 이것을 처리하는데 꽤 고생했습니다.

- 게임 이름 : 트리 트리
- 사용 언어 : C# / Unity 3D
- 장르 : 아케이드
- 제작 기간 : 1 개월
- 제작 목표 : Unity 3D를 이용해 전반적인 유니티 동작 방식을 학습하고, 그것을 위한 고전 게임 제작.
- 제작 인원 : 3인
- 키워드 : Unity 3D/모바일/고전
- 맡은 파트 : 막대기 처리와 피버를 통한 막대기 강화 효과(길어지기/공이 관통하게 하기) 및 사운드 처리를 맡았습니다.

- 게임 설명 : 기존 벽돌 게임을 약간 변형하여 한손으로 쉽게 즐길 수 있게, 막대기를 고무줄 당기듯 처리해 모바일에서 느끼기 힘든 터치감을 살렸습니다.
- 3개의 스테이지로 구성되어 있으며, 각 스테이지 마다 벽돌 구성이 다릅니다.
- 제작하며 느낀 점
- Unity를 이용해 처음으로 만든 3D 게임입니다. 게임 자체는 만들기가 어렵지 않았으나 기본적으로 모바일 게임은 터치감을 살리기 쉽지 않아, 조작방식에 대해 고민을 많이 하였습니다. 이는 모바일과 PC의 차이점을 확연히 느끼게 해주는 귀한 경험이었습니다.
- 씬 전환 등 게임의 흐름 처리 등을 익혔습니다.



- 게임 이름 : 던전 가드 [프로토 타입]
- 사용 언어 : C# / Unity
- 장르 : SRPG
- 제작 인원 : 1인
- 제작 기간 : 2 개월
- 제작 목표 : 유니티를 이용한 턴 제 알고리즘 및 타일 맵 알고리즘 학습.
- **키워드 : Unity / 랜덤 맵 생성 / 턴 방식 플레이**
- **주요 기술 적용 : 무작위 장애물로 랜덤한 맵 구현 , 몬스터가 공격 할 수 있는 최단거리 위치를 찾아 이동하기**

- 게임 설명 : 유닛을 배치하고 전투를 준비하여 적들을 막는 게임입니다. 현재는 프로토 타입으로 프로토 타입 유닛과 몬스터만 구성 하였습니다. 유닛 배치 및 전투만 가능 합니다.
- 제작하며 느낀 점
- 턴 제 방식 RPG는 실시간 RPG와 달리 이동과 공격 등 확실한 규칙대로 동작해야 하기 때문에, 익숙하지 않아서 고생했습니다.
- 몬스터 AI를 구현하면서 가장 적합한 위치를 찾도록 하는데 상당히 애를 먹었습니다. 이를 해결하기 위해 Flood Fill 알고리즘을 이용해 모든 타일의 코스트를 계산하면서 공격 가능한 범위 의 타일을 찾았고, 그 타일과 가장 가깝고 이동 가능한 타일을 찾도록 하였습니다.