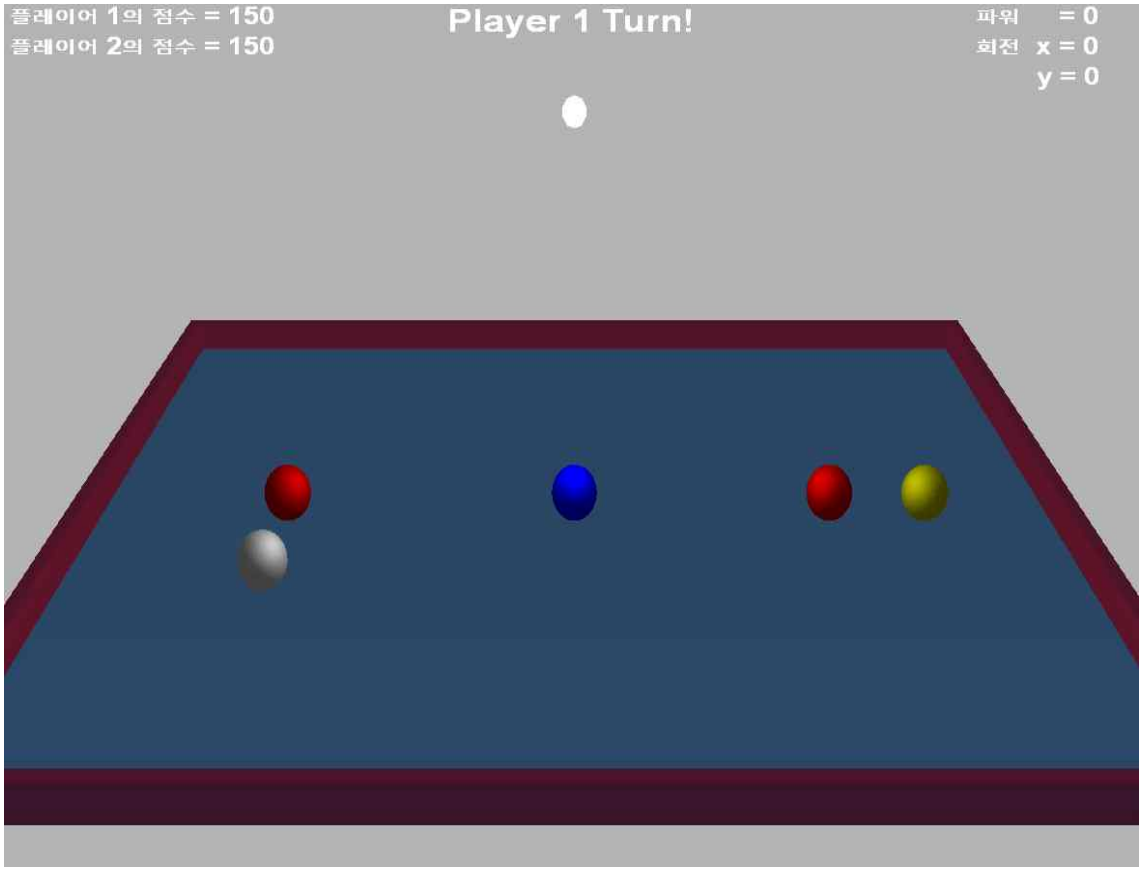
<Project#3 : OOP for Billiard Game>



Team name: 오지는 객체지향 9조

Leader 20185168 이기태

20183622 김태우

20185785 이상윤

Presenter 20184165 이태운

20185949 최예슬

목차

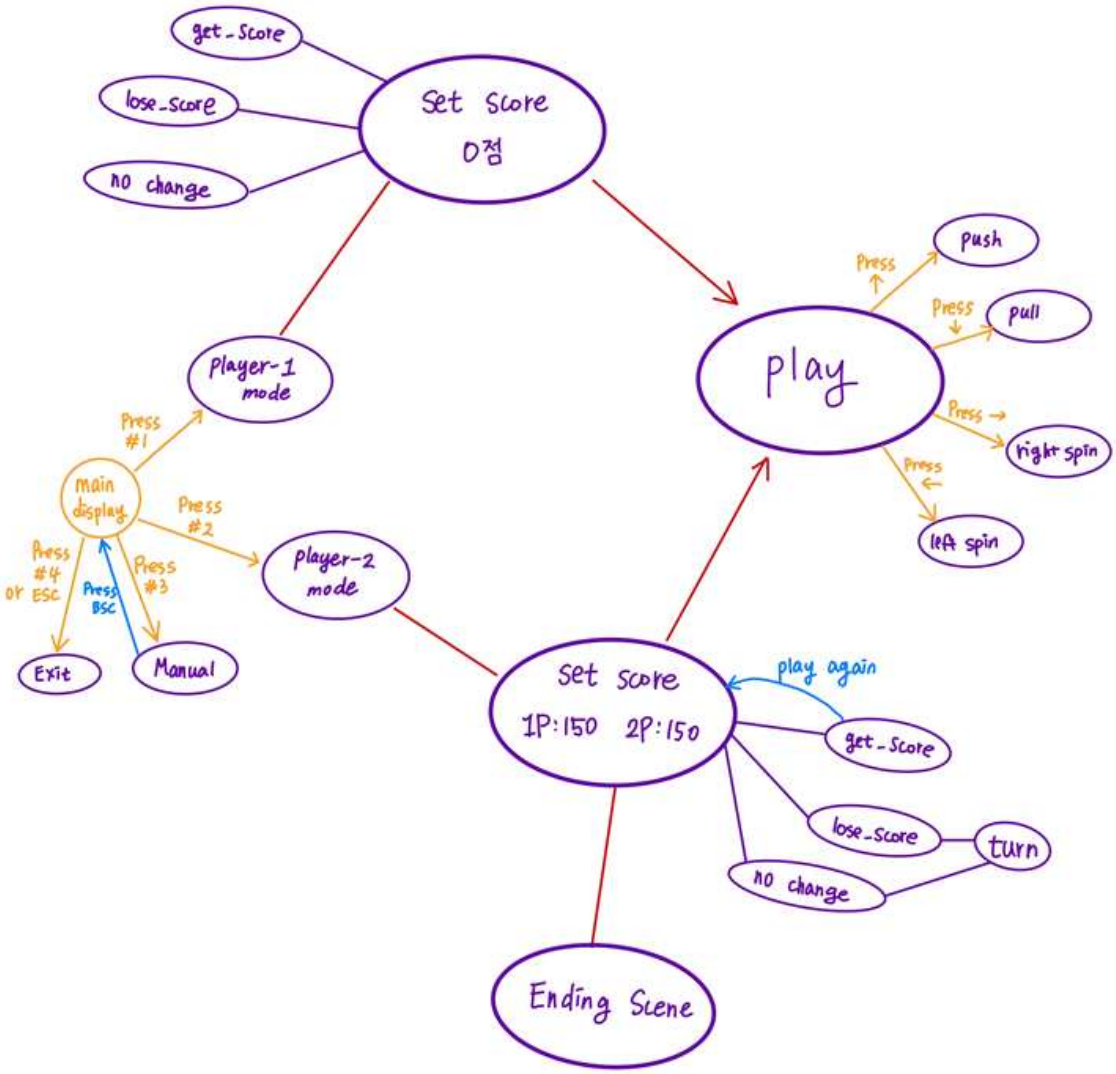
1. 설계 방향

편의성과 현실성있는 부분에 디테일을 강조한 당구 게임을 만드는데 가장 큰 의의를 두었다.

시각적으로 보기 편한 파란공(겨냥공)을 두어서 공의 두께조절이 용이하게했고 360도 시점이동을 통해 공 배치를 여러 각도에서 보게했다. 또 TAB을 누르면 기본시점으로 이동하는 편의성을 추가하였다. 공의 당점을 좌표로 받고(mesh_x,mesh_y) 키보드 화살표를 통해 상하좌우 회전조절을 용이하게하고 space 버튼을 두 번 누르는 방식을 채택하여 힘 조절을 용이하게 구현하였다. 이로 인하여 파란 공과 수구와의 거리가 중요한 요소가 아니므로 두께 조절을 하기 더 쉬워졌다.

이처럼 플레이하기 쉽고 현실과 가깝게 개발하는 쪽으로 설계 방향을 잡았다.

2. UML


get_score
lose_score
no change
Set score
0점
Player-1
mode
Press
#1
main
display
Press
#2
Player-2
mode
Press
#4
or ESC
Press
#3
Press
BSC
Exit
Manual
play
Press
push
Press
pull
Press
right spin
Press
left spin
Set score
1P:150 2P:150
play again
get_score
lose_score
turn
no change
Ending Scene

3. 클래스

1) Csphere

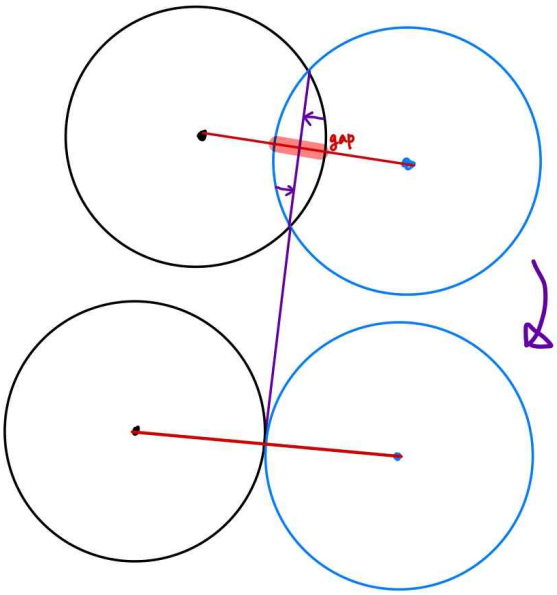
```
#include "CSphere.h"

float mesh_x = 0;
float mesh_y = 0;
float g_mesh_x = 0;
float g_mesh_y = 0;

int mesh_x_count = 0;
int mesh_y_count = 0;
int count = 0;

CSphere::CSphere(void)
{
    for (int i = 0; i < 4; i++) {
        hasHit[i] = 0;
    }
    D3DXMatrixIdentity(&m_mLocal);
    ZeroMemory(&m_mtrl, sizeof(m_mtrl));
    m_radius = 0;
    m_velocity_x = 0;
    m_velocity_z = 0;
    m_tork_x = 0;
    m_tork_y = 0;
    m_pSphereMesh = NULL;

}
CSphere::~CSphere(void) {}


void CSphere::setIndex(int _index)
{
    index = _index;
}
```

-setIndex : 몇 번째 공인지 설정해 준다.

```
bool CSphere::hasIntersected(CSphere& ball)
{
    D3DXVECTOR3 ball_pos = ball.getCenter();
    bool result = pow(center_x - ball_pos.x, 2) + pow(center_z - ball_pos.z, 2) < pow(1.9999999f *

    //if (result)   hashit[ball.index]++;
    return result;
}
```

-hasintersected : 공과 공 사이의 거리가 공의 반지름의 2배보다 작으면 충돌로 인식한다. 여기서 1.9999999f로 설정한 이유는 공끼리 부딪히지도 않았는데 충돌로 인식할 수 있기 때문에 설정해주었다. 이 때문에 공이 겹치는

경우가 생기는데 이는 hitby 함수에서 해결해준다.



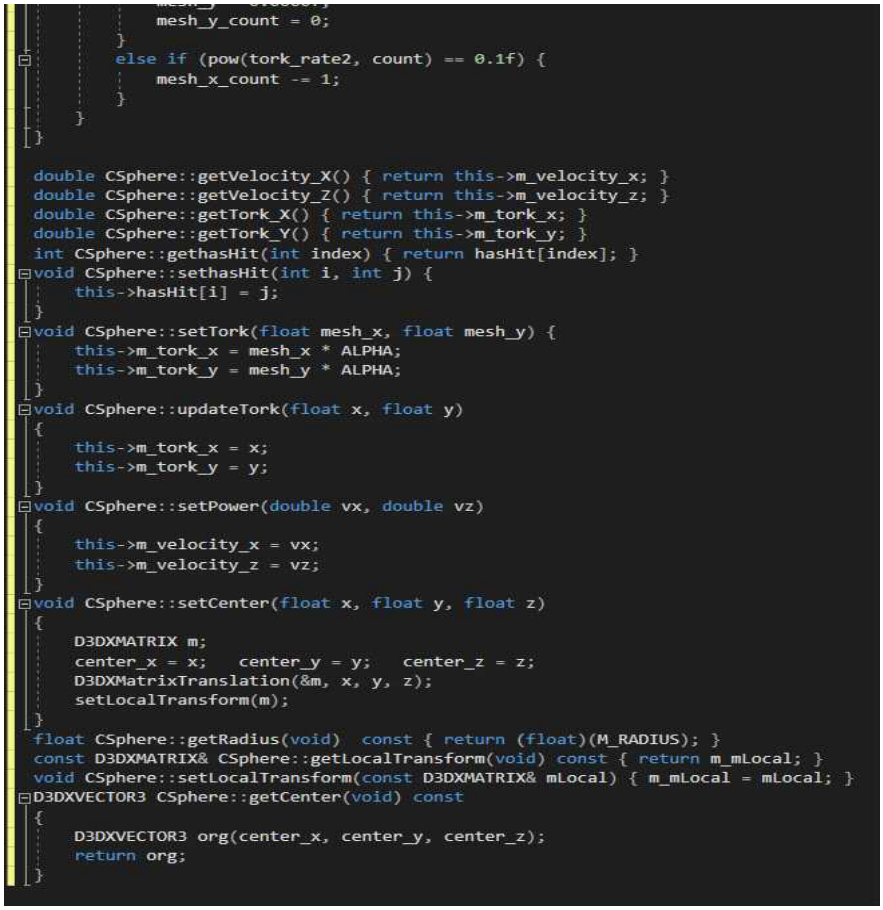
<공과 공 gap 생길 경우>

<hasintersected 함수를 그림으로 표현>

```
void CSphere::setIndex(int _index)
{
    index = _index;
}
bool CSphere::create(IDirect3DDevice9* pDevice, D3DXCOLOR color)
{
    if (NULL == pDevice)
        return false;

    m_mtrl.Ambient = color;
    m_mtrl.Diffuse = color;
    m_mtrl.Specular = color;
    m_mtrl.Emissive = d3d::BLACK;
    m_mtrl.Power = 5.0f;
    this->m_tork_x = 0;
    this->m_tork_x = 0;
    if (FAILED(D3DXCreateSphere(pDevice, getRadius(), 50, 50, &m_pSphereMesh, NULL)))
        return false;
    return true;
}

void CSphere::destroy(void)
{
    if (m_pSphereMesh != NULL) {
        m_pSphereMesh->Release();
        m_pSphereMesh = NULL;
    }
}

void CSphere::draw(IDirect3DDevice9* pDevice, const D3DXMATRIX& mWorld)
{
    if (NULL == pDevice)
        return;
    pDevice->SetTransform(D3DTS_WORLD, &mWorld);
    pDevice->MultiplyTransform(D3DTS_WORLD, &m_mLocal);
    pDevice->SetMaterial(&m_mtrl);
    m_pSphereMesh->DrawSubset(0);
}
```

```
if (hasIntersected(ball)) {
    this->hasHit[ball.index]++;
    if (ball.index == 2 && this->index == 3) {
        ball.hasHit[this->index]++;
    }
    D3DXVECTOR2 vec_ball_to_this(center_x - ball_pos.x, center_z - ball_pos.z);      //ball에서
    D3DXVec2Normalize(&vec_ball_to_this, &vec_ball_to_this);                          //단위벡터

    double gap = M_RADIUS - (sqrt(pow(center_x - ball_pos.x, 2) + pow(center_z - ball_pos.z, 2))

    setCenter(center_x + (gap * vec_ball_to_this.x), (float)M_RADIUS, center_z + (gap * vec_ball
    ball.setCenter(ball.center_x - (gap * vec_ball_to_this.x), (float)M_RADIUS, ball.center_z -
    //gap 만큼 거리 벌리기

    D3DXVECTOR2 vec_this(m_velocity_x, m_velocity_z);
    D3DXVECTOR2 vec_ball(ball.m_velocity_x, ball.m_velocity_z);

    D3DXVECTOR2 vec_ball_to_this_perpend(-vec_ball_to_this.y, vec_ball_to_this.x);   //vec_ball

    D3DXVECTOR2 vec_this_after = D3DXVec2Dot(&vec_this, &vec_ball_to_this_perpend) * vec_ball_to
    D3DXVECTOR2 vec_ball_after = D3DXVec2Dot(&vec_this, &vec_ball_to_this) * vec_ball_to_this +
    D3DXVECTOR2 vec_this_after2 = vec_this_after;

    setPower(vec_this_after.x, vec_this_after.y);
    ball.setPower(vec_ball_after.x, vec_ball_after.y);

    if (this->getTork_Y() > 0) {      // 밀어치기
        D3DXVec2Normalize(&vec_this, &vec_this);
        D3DXVec2Normalize(&vec_this_after2, &vec_this_after2);
        if (D3DXVec2Dot(&vec_this, &vec_this_after2) <= 0.5 && D3DXVec2Dot(&vec_this, &vec_this_
            D3DXVECTOR2 vec_tork(m_velocity_x, m_velocity_z);
            //D3DXVECTOR2 vec_this_after1 = (-ball.m_velocity_x, ball.m_velocity_z);
            setPower((vec_this.x * (this->getTork_Y() / ALPHA) * 10 + vec_this_after.x), (vec_t
            //setTork(getTork_X(), getTork_Y()*0.2);///final
            setTork(getTork_X(), 0);
            mesh_y = 0.000f;
            mesh_y_count = 0;
        }
    }
    else if (this->getTork_Y() < 0) {   //끌어치기
        if (D3DXVec2Dot(&vec_this, &vec_this_after2) <= 0.5 && D3DXVec2Dot(&vec_this, &vec_this_
            setPower((vec_this.x * (this->getTork_Y() / ALPHA) * 5 + vec_this_after.x), (vec_th
            setTork(getTork_X(), 0);
            mesh_y = 0.000f;
            mesh_y_count = 0;
        }
    }
```

```
void CSphere::ballUpdate(float timeDiff)
{
    const float TIME_SCALE = 3.3;
    D3DXVECTOR3 cord = this->getCenter();
    double vx = abs(this->getVelocity_X());
    double vz = abs(this->getVelocity_Z());

    if (vx > 0.01 || vz > 0.01)
    {
        float tX = cord.x + TIME_SCALE * timeDiff * m_velocity_x;
        float tZ = cord.z + TIME_SCALE * timeDiff * m_velocity_z;

        //correction of position of ball
        // Please uncomment this part because this correction of ball position is neces
        if (tX >= (4.5 - M_RADIUS))
            tX = 4.5 - M_RADIUS;
        else if (tX <= (-4.5 + M_RADIUS))
            tX = -4.5 + M_RADIUS;
        else if (tZ <= (-3 + M_RADIUS))
            tZ = -3 + M_RADIUS;
        else if (tZ >= (3 - M_RADIUS))
            tZ = 3 - M_RADIUS;

        this->setCenter(tX, cord.y, tZ);
    }
    else { this->setPower(0, 0); }
    double rate = 1 - (1 - DECREASE_RATE) * timeDiff * 400;
    if (rate < 0)
        rate = 0;
    this->setPower(getVelocity_X() * rate, getVelocity_Z() * rate);
    if (!(this->getVelocity_X() == 0 && this->getVelocity_Z() == 0)) {
        double tork_rate1 = 1 - (1 - TORK_DECREASE_RATE) * timeDiff * 500;
        double tork_rate2 = 1 - (1 - TORK_DECREASE_RATE) * timeDiff * 2000;
        if (tork_rate1 < 0)
            tork_rate1 = 0;
        if (tork_rate2 < 0)
            tork_rate2 = 0;
        float nexttorkX = this->getTork_X() * (float)tork_rate1;
        float nexttorkY = this->getTork_Y() * (float)tork_rate2;
        this->updateTork(nexttorkX, nexttorkY);
        float a = mesh_x;
        mesh_x *= tork_rate1;
        mesh_y *= tork_rate2;
        count++;
        if (abs(mesh_x) < (M_RADIUS / 100)) {
```



```
            mesh_y_count = 0;
        }
        else if (pow(tork_rate2, count) == 0.1f) {
            mesh_x_count -= 1;
        }
    }
}

double CSphere::getVelocity_X() { return this->m_velocity_x; }
double CSphere::getVelocity_Z() { return this->m_velocity_z; }
double CSphere::getTork_X() { return this->m_tork_x; }
double CSphere::getTork_Y() { return this->m_tork_y; }
int CSphere::gethasHit(int index) { return hasHit[index]; }
void CSphere::sethasHit(int i, int j) {
    this->hasHit[i] = j;
}
void CSphere::setTork(float mesh_x, float mesh_y) {
    this->m_tork_x = mesh_x * ALPHA;
    this->m_tork_y = mesh_y * ALPHA;
}
void CSphere::updateTork(float x, float y)
{
    this->m_tork_x = x;
    this->m_tork_y = y;
}
void CSphere::setPower(double vx, double vz)
{
    this->m_velocity_x = vx;
    this->m_velocity_z = vz;
}
void CSphere::setCenter(float x, float y, float z)
{
    D3DXMATRIX m;
    center_x = x;   center_y = y;   center_z = z;
    D3DXMatrixTranslation(&m, x, y, z);
    setLocalTransform(m);
}
float CSphere::getRadius(void)  const { return (float)(M_RADIUS); }
const D3DXMATRIX& CSphere::getLocalTransform(void) const { return m_mLocal; }
void CSphere::setLocalTransform(const D3DXMATRIX& mLocal) { m_mLocal = mLocal; }
D3DXVECTOR3 CSphere::getCenter(void) const
{
    D3DXVECTOR3 org(center_x, center_y, center_z);
    return org;
}
```

-hitby : 위 그림에서 보이는 겹치는 부분의 절반이 gap인데 그 gap만큼 양쪽 공이 멀어지는 방향으로 벌려주면 두 공사이의 거리가 정확히 반지름의 2배가 된다. 이걸로 겹치는 문제가 해결되고 2번 충돌하지 않는다.

-ballupdate : 파란공(겨냥공)이 밖으로 못벗어나게 하며 시간이 지날수록 공의 속도와 토크가 점점 줄어든다.

-getTork : 토크(돌림힘) 반환

-setTork : 객체가 얼마나 토크를 가지고 있는지 mesh_x, mesh_y 변수를 받아서 일정값(ALPHA)을 연산해줌

-updateTork : tork가 점점 줄어들기 때문에 이걸 반영시키기 위한 함수

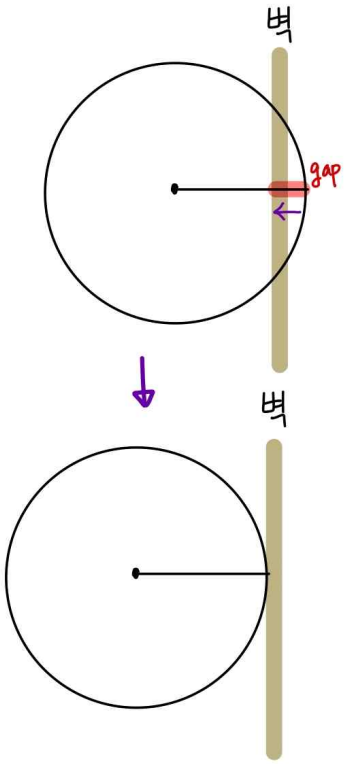
2) CWall

```
#include "CWall.h"

int signOf(double num) {      //수의 부호를 구해주는 함수
    if (num >= 0)
        return 1;
    return -1;
}
void rotate(CSphere& ball, float velocity_theta) {
    ball.setPower(ball.getVelocity_X() * cos(velocity_theta) - (ball.getVelocity_Z() * sin(velo
}
CWall::CWall(void)
{
    D3DXMatrixIdentity(&m_mLocal);
    ZeroMemory(&m_mtrl, sizeof(m_mtrl));
    m_width = 0;
    m_depth = 0;
    m_pBoundMesh = NULL;
}
CWall::~CWall(void) {}
bool CWall::create(IDirect3DDevice9* pDevice, float ix, float iz, float iwidth, float iheight,
{
    if (NULL == pDevice)
        return false;

    m_mtrl.Ambient = color;
    m_mtrl.Diffuse = color;
    m_mtrl.Specular = color;
    m_mtrl.Emissive = (D3DXCOLOR)D3DCOLOR_XRGB(20, 20, 40);
    m_mtrl.Power = 5.0f;

    m_width = iwidth;
    m_depth = idepth;

    if (FAILED(D3DXCreateBox(pDevice, iwidth, iheight, idepth, &m_pBoundMesh, NULL)))
        return false;
    return true;
}
void CWall::destroy(void)
{
    if (m_pBoundMesh != NULL) {
        m_pBoundMesh->Release();
        m_pBoundMesh = NULL;
    }
}
```

- rotate : 공이 벽에 충돌했을 때 원래 가려던 벡터를 회전량에 비례해서 진행할 수 있게 해준다.

```
void CWall::draw(IDirect3DDevice9* pDevice, const D3DXMATRIX& mWorld)
{
    if (NULL == pDevice)
        return;
    pDevice->SetTransform(D3DTS_WORLD, &mWorld);
    pDevice->MultiplyTransform(D3DTS_WORLD, &m_mLocal);
    pDevice->SetMaterial(&m_mtrl);
    m_pBoundMesh->DrawSubset(0);
}
bool CWall::hasIntersected(CSphere& ball)
{
    D3DXVECTOR3 ball_pos = ball.getCenter();

    if (m_width > m_depth) {   //가로벽 충돌
        if (ball_pos.z - m_z <= M_RADIUS + 0.06f && ball_pos.z - m_z >= -M_RADIUS - 0.06f)
            return true;
    }
    else {              //세로벽 충돌
        if (ball_pos.x - m_x <= M_RADIUS + 0.06f && ball_pos.x - m_x >= -M_RADIUS - 0.06f)
            return true;
    }
    // Insert your code here.
    return false;
}
```

- hasintersected : 공의 중심좌표와 벽의 x 혹은 z 거리가 (공 반지름 + 0.06f)보다 작으면 충돌로 인식한다.



<공과 벽 gap 생길 경우

<hasintersected를 그림으로 표현>

```
void CWall::setPosition(float x, float y, float z)
{
    D3DXMATRIX m;
    this->m_x = x;
    this->m_z = z;

    D3DXMatrixTranslation(&m, x, y, z);
    setLocalTransform(m);
}
void CWall::setIndex(int n) {
    this->index = n;
}
float CWall::getHeight(void) const { return M_HEIGHT; }
void CWall::setLocalTransform(const D3DXMATRIX& mLocal) { m_mLocal = mLocal; }
```

- setIndex : 공이 어느 벽에 맞았는지 공 객체가 알 수 있게 도와주는 함수

```
void CWall::hitBy(CSphere& ball)
{
    D3DXVECTOR3 ball_pos = ball.getCenter();

    double gap;      //벽과 공 사이의 겹치는 부분

    if (hasIntersected(ball)) {
        if (m_width > m_depth) {        //가로벽이면 z 방향 속도만 바꾸면 됨!

            gap = M_RADIUS + 0.06f - abs(ball_pos.z - m_z);         //z 방향 gap 계산

            ball.setCenter(ball_pos.x, (float)M_RADIUS, ball_pos.z - gap * signOf(ball.getVeloci
            ball.setPower(ball.getVelocity_X(), -ball.getVelocity_Z());
        }
        else {                          //세로벽이면 x 방향 속도만 바꾸면 됨!

            gap = M_RADIUS + 0.06f - abs(ball_pos.x - m_x);         //x 방향 gap 계산
            ball.setCenter(ball_pos.x - gap * signOf(ball.getVelocity_X()), (float)M_RADIUS, bal
            ball.setPower(-ball.getVelocity_X(), ball.getVelocity_Z());
        }

        if (ball.getTork_X() != 0) {
            float velocity_theta = ball.getTork_X() / 100;
            float theta;
            if (this->index == 0 || this->index == 1)
                theta = abs(atan(ball.getVelocity_Z() / ball.getVelocity_X()));
            else {
                theta = abs(atan(ball.getVelocity_X() / ball.getVelocity_Z()));
            }

            switch (this->index) {
            case 0:
                rotate(ball, theta*velocity_theta);
                break;
            case 1:
                rotate(ball, theta*velocity_theta);
                break;
            case 2:
                rotate(ball, theta*velocity_theta);
                break;
            case 3:
                rotate(ball, theta*velocity_theta);
                break;
            }
```

- **hitby : hasintersected가 참일 때 실행되며 위 그림처럼 gap만큼 공을 떨어뜨린다. 가로벽이면 z 속도벡터를 부호변환하고 세로벽이면 x 속도벡터를 부호변환한다. getTork가 있으면 4가지 경우에 따라 충돌 직후에 갈 속도 벡터를 rotate 함수를 이용해서 회전시켜준다.**

3) CLight

```
ass CLight {
blic:
  CLight(void);
  ~CLight(void);
blic:
  bool create(IDirect3DDevice9* pDevice, const D3DLIGHT9& lit, float radius);
  void destroy(void);
  bool setLight(IDirect3DDevice9* pDevice, const D3DXMATRIX& mWorld);
  void draw(IDirect3DDevice9* pDevice);
  D3DXVECTOR3 getPosition(void) const;
ivate:
  DWORD               m_index;
  D3DXMATRIX          m_mLocal;
  D3DLIGHT9           m_lit;
  ID3DXMesh* m_pMesh;
  d3d::BoundingSphere m_bound;
```

-> 기존 제공된 클래스와 비슷하다.

4) CPlayer

```cpp
#include "CPlayer.h"

CPlayer::CPlayer(int _score) {
    score = _score;
}
int CPlayer::printscore() {
    return this->score;
}
void CPlayer::getscore() {
    if (this->score)
        this->score -= 10;
}
void CPlayer::losescore() {
    if (this->score != 0)
        this->score += 10;
}
void CPlayer::setscore(int score) {
    this->score = score;
}
void CPlayer::set_first_player(bool fp) {
    this->first_player = fp;
}
bool CPlayer::get_first_player() {
    return this->first_player;
}
void CPlayer::turn() {
    this->first_player = !(this->first_player);
}
```

- **getscore : 득점**
- **losescore :실점 -> 1인모드에서는 득점/실점 반대**
- **set_first_player : 어느 플레이어의 차례인지 초기 설정**
- **turn : 플레이어의 차례를 바꿈**

5) Sound

```
class Sound
{
private:
    LPDIRECTSOUNDBUFFER g_lpDSBG;

    LPDIRECTSOUND8  g_lpDS;
    BOOL            g_bPlay;

public:
    Sound()
    {
        g_lpDSBG = NULL;
        g_lpDS = NULL;
        g_bPlay = NULL;
    }

    BOOL CreateDirectSound(HWND hWnd);
    BOOL LoadWave(LPSTR lpFileName);
    BOOL SetVolume(LONG lVolume);
    BOOL SetPan(LONG lPan);
    ~Sound() {};

    void DeleteDirectSound();
    void Play(BOOL Loop);
    void Stop();

};
```

-DirectSound를 사용하기 위해 dsound.lib과 같은 라이브러리 참조
-인트로 사운드, 부딪히는 효과음을 Sound 형식의 객체 intro, ddack로 전역 변수로 선언
-CreateDirectSound : directsound 버퍼 생성
-LoadWave : .wav 파일명을 문자열로 입력받아 읽음
-Play : 버퍼에 저장된 파일 재생, 메소드등을 활용하여 각 상황에 맞는 소리 재생

6) font

```
ass font

ivate:
  static  font* TextInst;
  ID3DXFont* m_pFont;
  int         m_nMax_X;
  int         m_nMax_Y;


blic:
  font(void);
  ~font(void);
  void    Init(IDirect3DDevice9* Device, int height, int width);//최종수정
  void    Print(LPCSTR cSTR, int nX/*문자열의 왼쪽좌표*/ = 0, int nY/*문자열의 윗쪽좌표*/ = 0, D3DXCOLOR ARGB = 0xFFFFFFFF);
  static  font* GetInst(void);
  void        FreeInst(void);
```

-Init : 어느 디바이스에 글자를 쓸 것인가를 결정, 높이 길이 설정

-Print : DrawTextA 호출 / 좌표, 색깔 설정

-GetInst : 메모리 할당

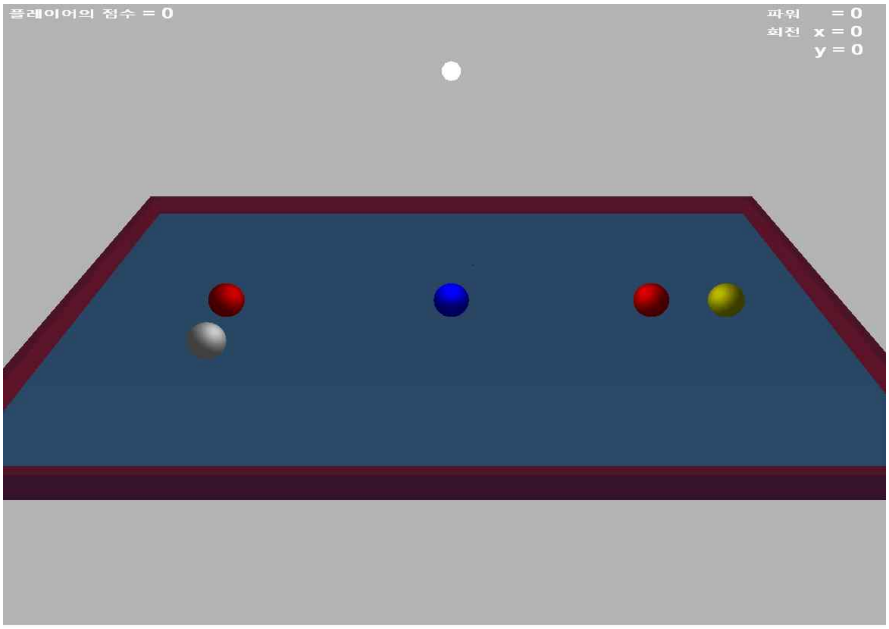
-freeInst : 메모리 해제

4. 특징

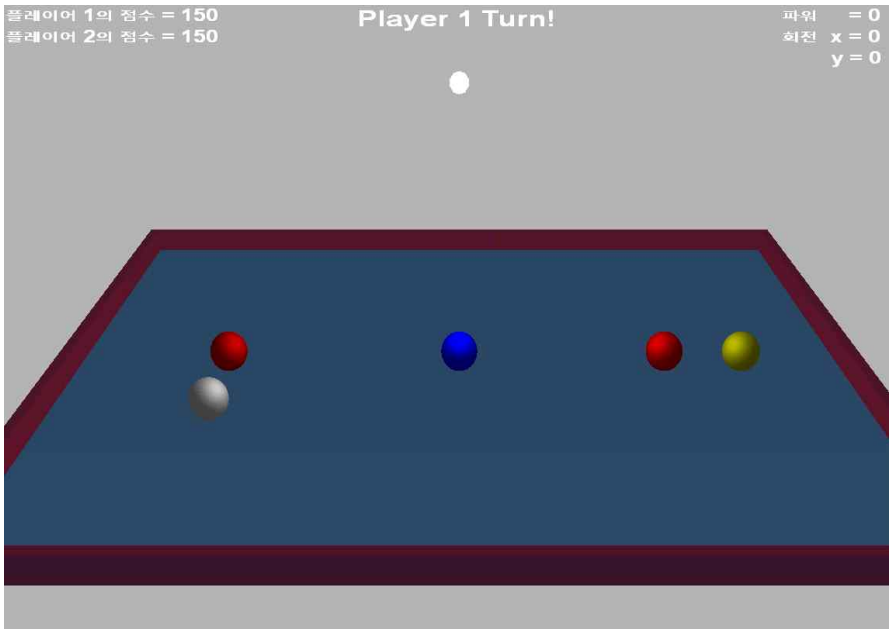
1) 메인 메뉴 / 엔딩

<메인 메뉴>

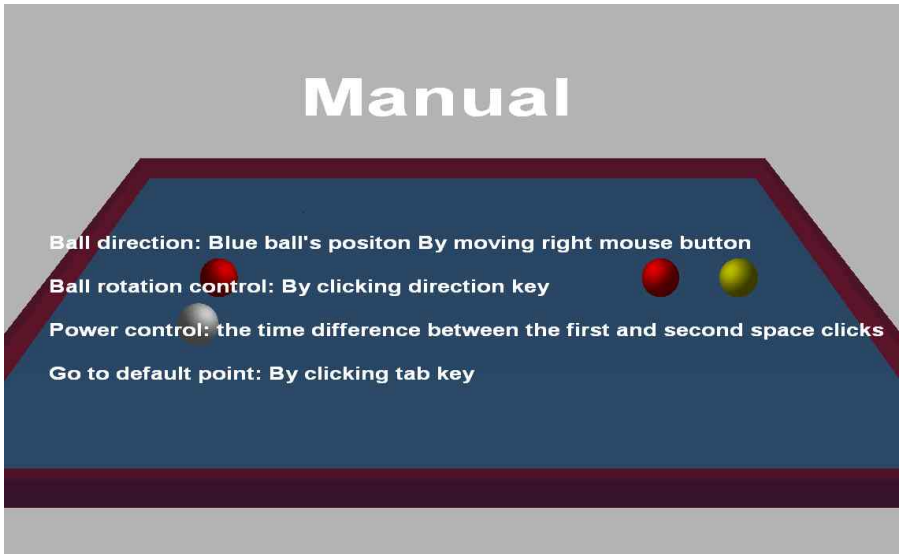
<엔딩 화면>



<메인메뉴에서 1 선택 시: 1인 플레이 모드>



<메인메뉴에서 2 선택 시: 2인 플레이 모드>



<메인메뉴에서 3 선택 시: Manual>

- 매뉴얼화면에서 Backspace 버튼을 누르면 메인메뉴로 돌아간다

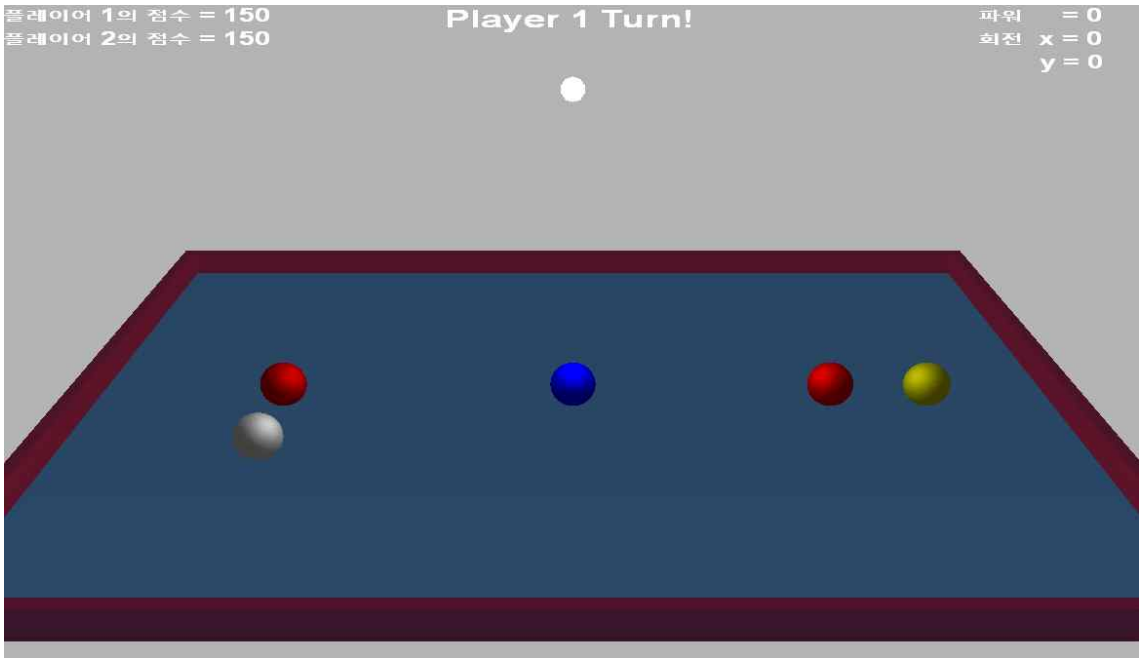
메인 메뉴에서 4 선택 시: 게임 종료

모든 화면에서 ESC 누를 시: 게임 종료

2) 점수판

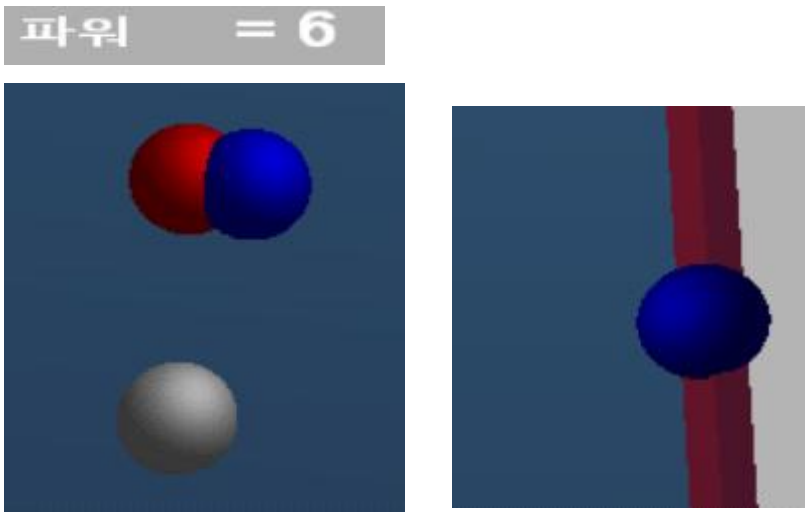
1인모드에서는 플레이어1의 점수만, 2인모드에서는 플레이어 1, 2의 점수만 나타나게 했다. 1인모드의 경우 0점에서 시작해 한번 성공 시 10점씩 증가, 한번 실패 시 10점씩 감소하는 방식이며 2인모드에서는 초기 점수 150점으로 설정해 한번 성공 시 10점씩 감소하며 한번 더 칠 기회를 갖는다. 한번 실패 시 10점씩 증가하며 차례가 바뀌며 0점이 되어야 게임이 종결된다.

3) 시점 초기화



공의 배치를 시점을 이동하며 여러 각도에서 확인하고 난 뒤 기본시점이 아닌 다른 시점일때는 파란공(겨냥공)이 이상하게 이동하기 때문에 VK_TAB을 인식하면 디바이스에 위 사진처럼 기본 시점으로 되돌리는 기능을 구현하였다.

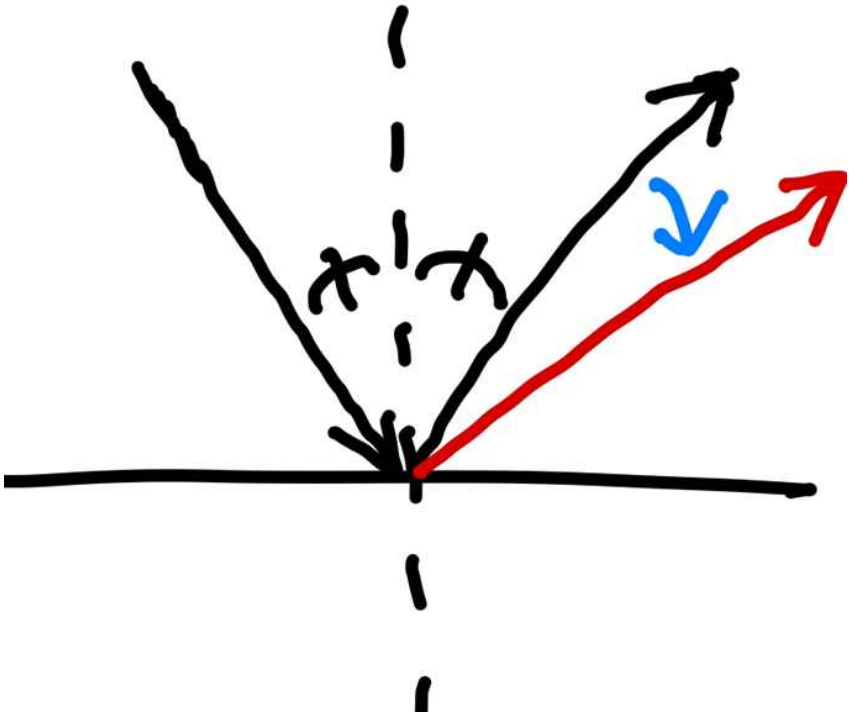
4) 공 두께조절, 파워조절



기존에 제공된 코드는 수구와 파란공(겨냥공) 사이의 거리에 비례해 힘을 조절하는 방식으로 구현했었다. 하지만 이 방식대로 하면 수구와 1적구의 정확한 두께 판단이 힘들어 게임을 진행하는데에 불편함이 있었다. 그래서 힘 조절을 space키를 한번 누르면 위 사진처럼 게임 우측상단에 '파워 = 0~100'를 반복하게 만들었고 원하는 힘에서 space를 한번 더 누르면 그 파워로 공이 진행하게 구현하였다. 이로써 파워 조절을 용이하게 하였다. 1적구에 파란공(겨냥공)을 겹쳐 둠으로써 좀 더 정확한 두께 판단이 가능해졌다. 또 파란공(겨냥공)이 당구대 밖에 못나가게 하여 게임을 진행하는데 불편함을 줄였다.

5) 공의 회전

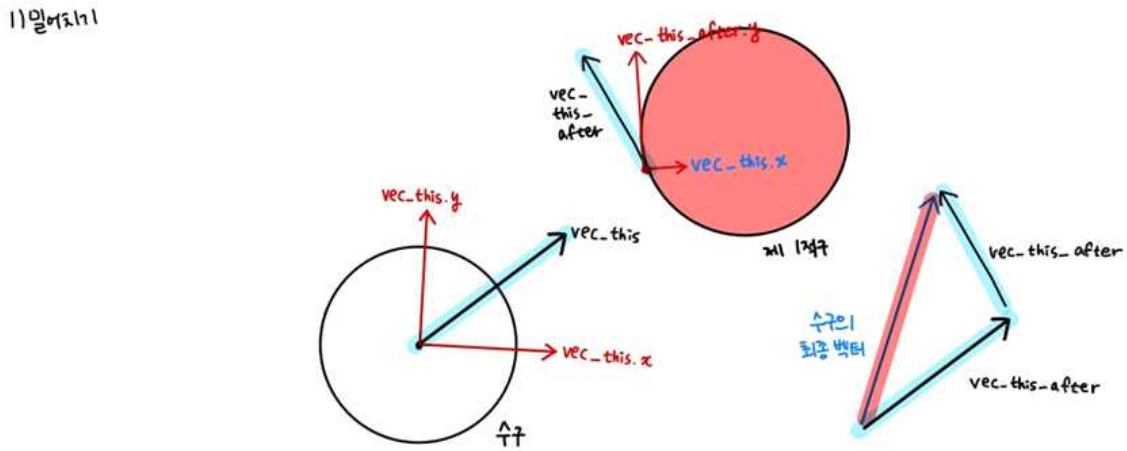
1. 오른쪽 / 왼쪽 회전

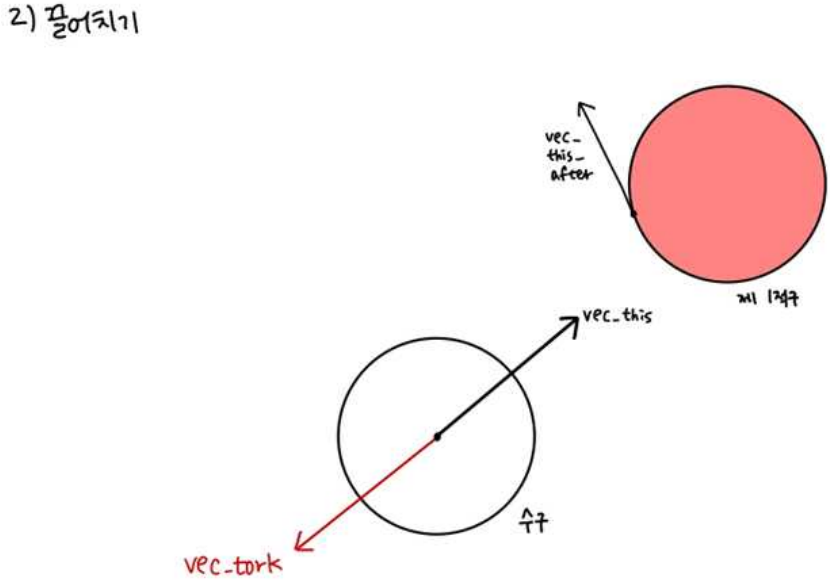


입력값으로 mesh_x가 들어오면 거기다 일정한 상수 알파를 곱해서 그값(tork_x)을 -30~30(%)의 범위를 갖게 만들어준다.
토크값을 위와 같이 정한 이유는 퍼센트로 하지않고 항상 회전량을 고려하여 일정하게 속도벡터를 돌려주게 되면 입사각이 속도벡터를 회전시킬 각 theta보다 작으면 충돌후 공의 속도벡터가 벽쪽을 향하게 되기때문에 벽에 붙어서 진행하게된다. 따라서 입사각의 최대 30퍼센트만큼 회전시켜주어 다음과 같은 오류를 피할수 있게 하였다.
회전을 가진 공이 벽에 충돌하면 그공의 진행 방향과 토크값을 고려하여 속도벡터가 충돌후 어느 방향으로 진행할지를 생각하여 그 방향으로 rotate함수를 이용하여 돌려준다. 토크 값이 음수일경우 공의 진행 방향에 상관없이 항상 충돌 후 속도 벡터의 진행 방향이 반시계 방향으로 회전되어야 하고 토크값이 양수일 경우 충돌 후의 진행 방향이 시계방향으로 회전되어야 한다. rotate함수는 theta만큼 반시계 방향으로 돌려주기 때문에 토크가 음수일

경우 2파이 - theta 를 넣은 것과 같기 때문에 theta 만큼 시계방향으로 돌려줄수 있게 된다. 실제 게임에서는 왼쪽/오른쪽 방향키로 왼쪽/오른쪽 당점을 조절할 수 있다.

2. 밀어치기 / 끌어치기





- 보다 현실감있는 당구 게임을 위해 일정 두께 이상을 쳤을 때 당점이 위 / 아래면 밀어치기 / 끌어치기가 되도록 구현하였다. 수구가 1적구를 맞기 전 벡터(vec_this), 수구가 1적구를 맞고난 후의 벡터(vec_this_after)라 하면 일정 두께 이상을 맞았는지 알기 위해서 우선 vec_this와 vec_this_after를 크

기가 1인 기저벡터로 바꾼 후 vec_this와 vec_this_after를 내적한 값이 일정 크기 이상이면 밀어치기 / 끌어치기가 구현되도록 하였다. 수많은 수행착오 끝에 일정 크기를 0.5로 결정했다.

- 밀어치기와 끌어치기의 방향과 힘은 vec_this와 vec_this_after 각 벡터를 x,y축 속도로 분해한 후 각각을 더해서 합성시킴으로써 방향과 힘을 결정했다. 여기서 끌어치기는 tork_y의 값이 음수이 때문에 vec_this에 -를 곱해서 연산해줘야 한다. 실제 게임에서는 위/아래 방향키로 위/아래 당점을 조절할 수 있다.

5. 보완할 점

1. 당구 게임을 기본 시점을 기준으로 했기 때문에 기본시점이 아닌 다른 시점일 때는 파란공(겨냥공)을 우클릭으로 이동하려하면 이상하게 이동한다.

2. 공과 벽의 마찰력이 없어서 현실과는 다르게 공끼리 충돌해도 에너지가 완벽하게 보존된다.