**REPORT**

**2019 객체지향프로그래밍**

**<DirectX 12를 이용하여 ”닷지”게임 구현하기>**

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjSmoi9w4jhAhUOfnAKHfrKAQEQjRx6BAgBEAU&url=https://www.conteenew.com/bbs/board.php?bo_table%3Dartwork%26wr_id%3D1304&psig=AOvVaw0ZXwbAH-dsoKB2qC6AFSiu&ust=1552889789241417)

과 목: 객체지향프로그래밍 03분반

팀 원: 20184060 이현주

20184754 김현주

**목차**

1. 게임 설명
2. Compile, execute 방법
3. 프로그램의 기능
4. 기능 구현 중 생긴 문제점과 해결방법
5. sw 시스템의 객체지향적 디자인
6. 결론

.

1. **게임 설명**

“닷지”라는 게임은 사방에서 날라오는 공을 피해서 오래 살아남는 것을 목표로 하는 게임이다.

날아오는 공과 맞닿으면 게임은 종료된다.

우리는 이러한 “닷지” 게임을 구현하고자 했다.

1. **Compile, execute 방법**

개발 환경: 2019 visual studio, DirectX 12

실행 방법: VirtualLego.dsw 파일을 visual studio로 실행시킨다.

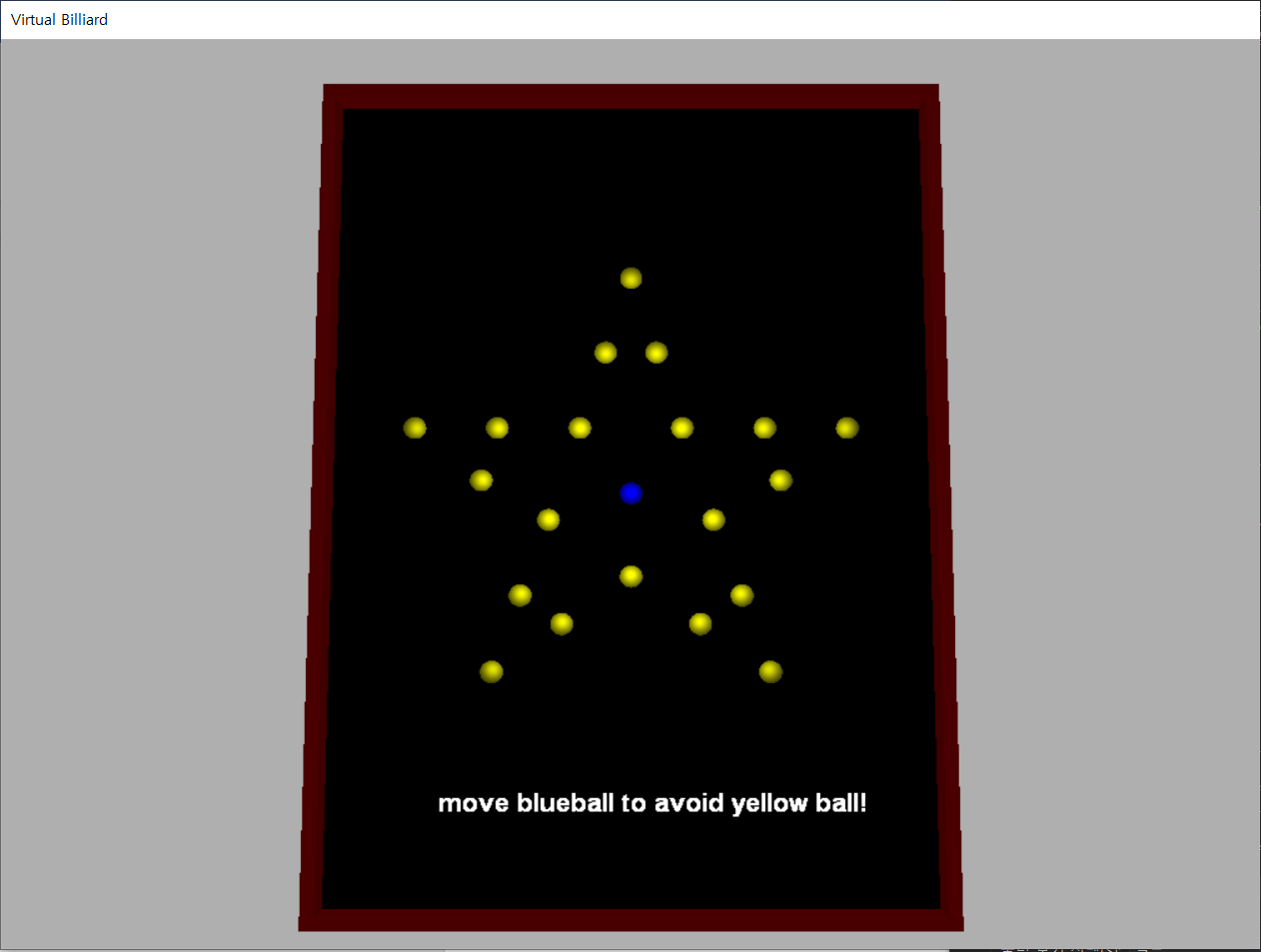
* 프로젝트의 속성에 들어간다.
* VC++디렉터리에 포함 디렉터리를 “(DirectX SDK가 있는 path)\Include”로 적용한다.
* VC++디렉터리에 라이브러리 디렉터리를 “(DirectX SDK가 있는 path)\Lib\x86”로 적용한다.
* C/C++의 코드생성의 함수 수준 링크 사용을 “예”로 변경한다.
* “디버깅” 혹은 “디버그하지 않고 시작”을 한다.

1. **프로그램의 기능**

프로그램의 기능을 실행 순서에 따라 설명하겠다.

1. 처음 프로그램 실행 화면

공들과 게임이 진행되는 공간, 간략한 게임소개가 나온다.

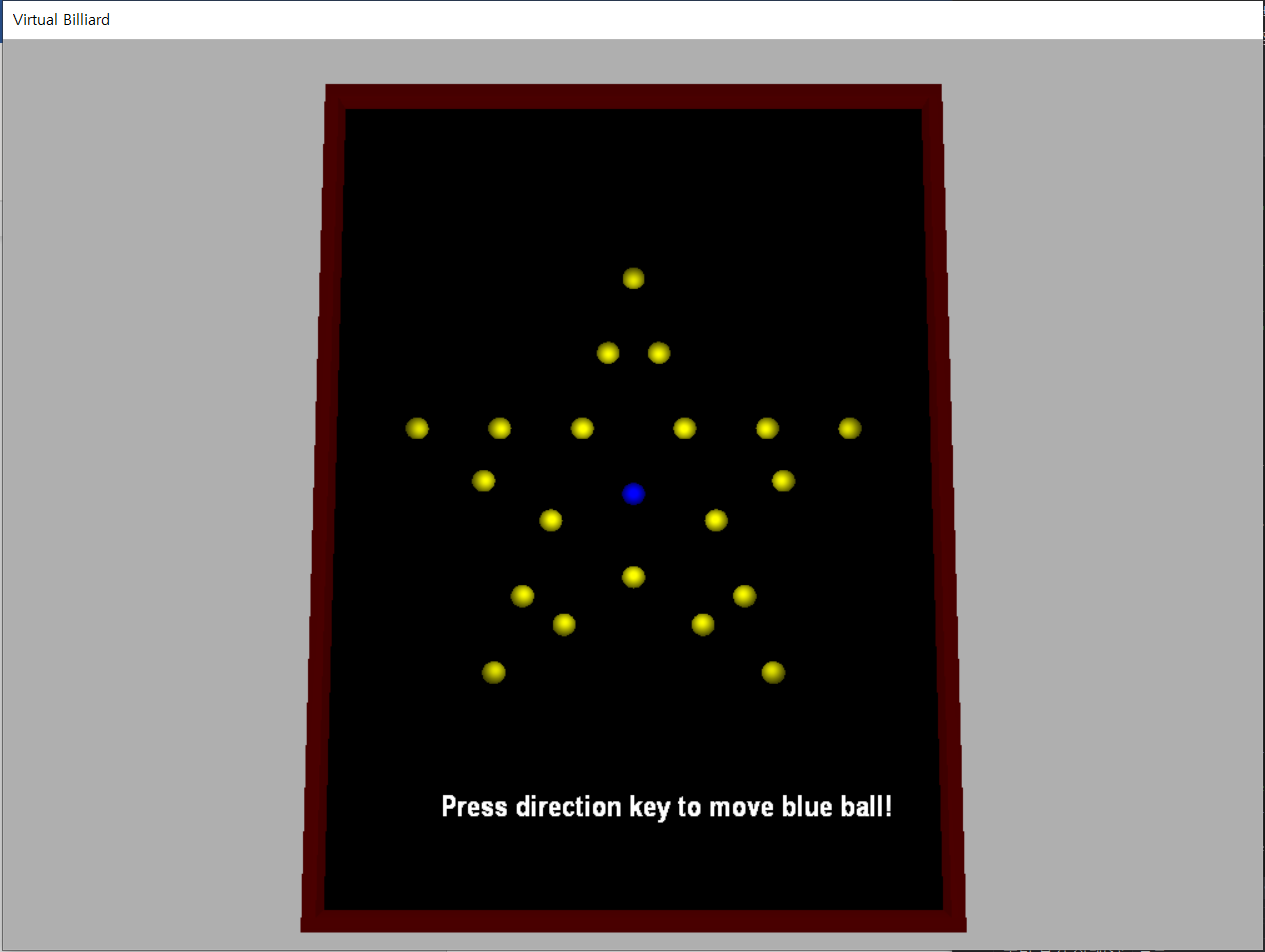


* 구현 방식

|  |  |  |  |
| --- | --- | --- | --- |
| 설정(생성) 대상 |  | Setup()  \*각 객체들을 생성하고  초기화 | Display()  \*Setup()에서 초기화한  객체를 화면에 표시 |
| 공 |  | -공 생성  CSphere::create(device,색)  -공 위치 설정  CSphere::setCenter(x,y,z)  -공 속도 설정  CSphere::setPower(xv,zv) | -공 화면에 표시  CSphere::draw() |
| 공이 움직이는  공간 |  | -plane 생성  CWall::create()  -위치 설정  CWall::setPosition(x,y,z) | -plane 화면에 표시  CWall:draw() |
| 공의 운동범위를 표시하는 벽 |  | -공이 움직이는 공간과 동일 | -공이 움직이는 공간과 동일 |
| 게임 설명 |  | -글자의 크기와 글꼴 설정  CText::Init()  - | -글 내용 char형 배열에 입력  -생성한 글 화면에 표시  CText::Print() |

1. 첫 화면에서 엔터를 누른 후의 화면

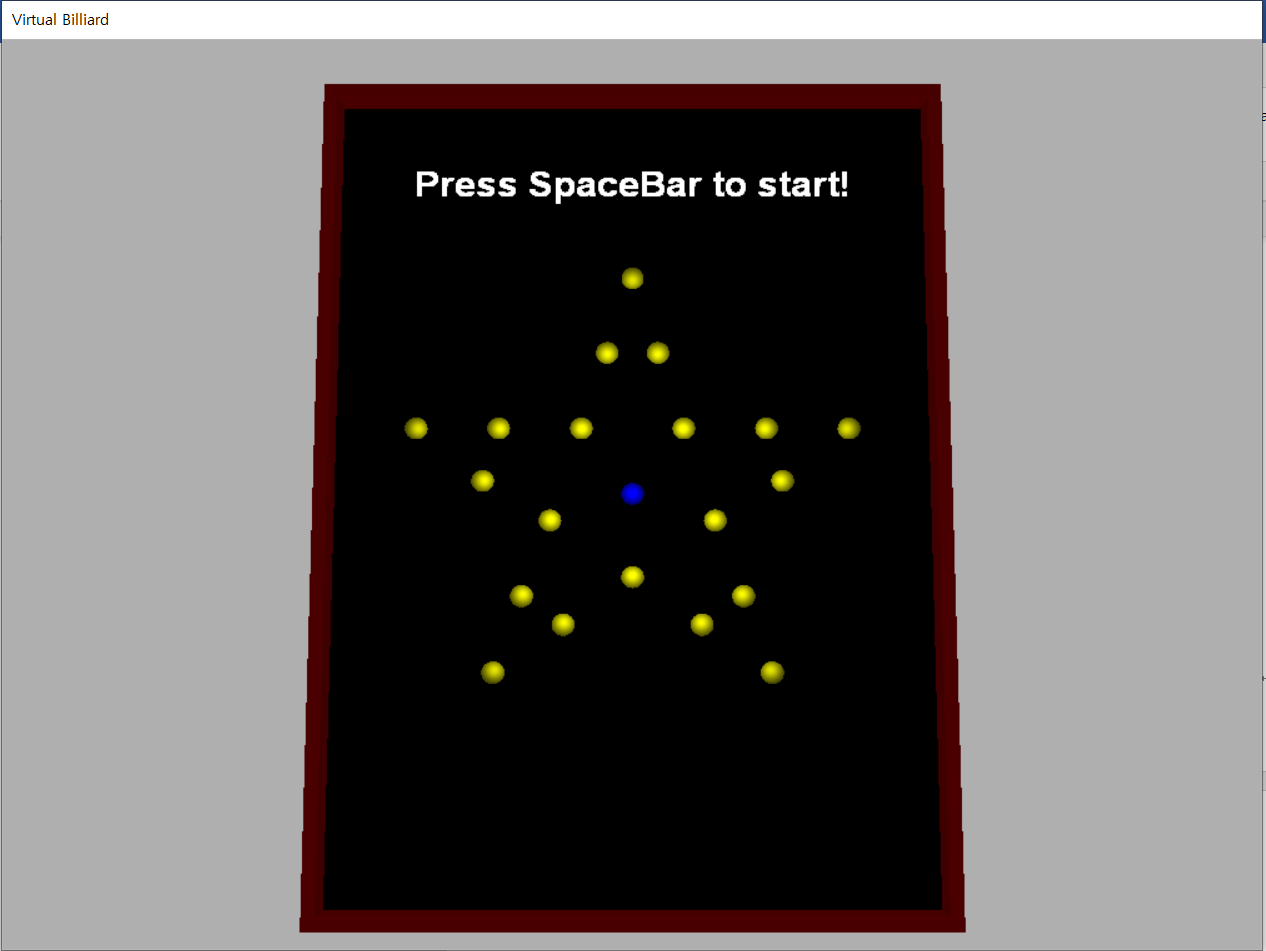
간략한 게임 방법이 나온다.



* 구현 방식은 1)에서 text 내용만 변형(char형 배열 새로 생성)

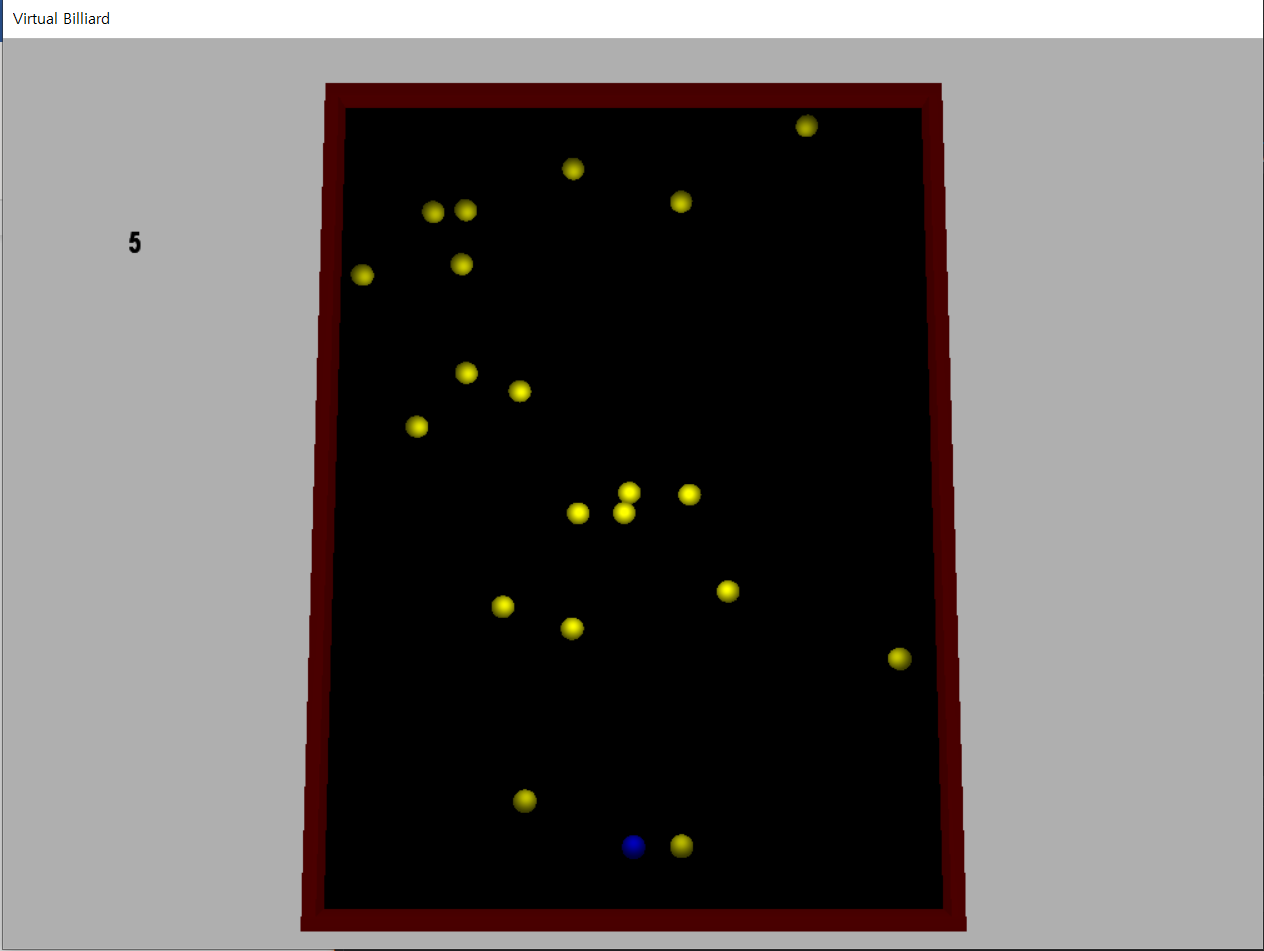
1. 두번째 화면에서 키보드의 엔터를 다시 누른 화면

게임을 시작하려면 스페이스바를 눌러야 함을 알려준다.



* 구현 방식은 2)와 동일

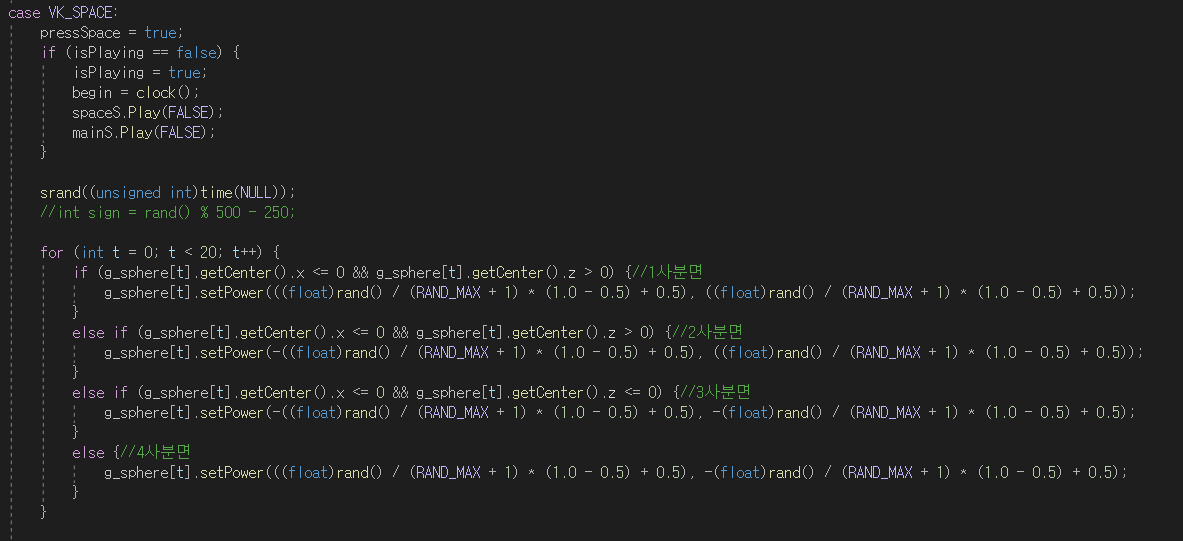
1. 이후 스페이스바를 누르면 게임이 실행된다.



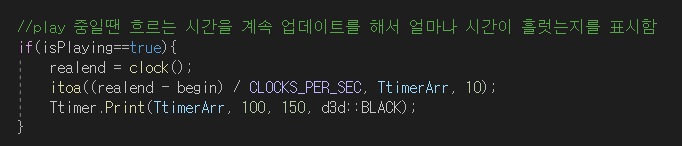
- 왼쪽의 숫자: 파란공이 노란 공과 부딪히지 않고 살아남은 시간을 보여준다.

- 구현 방식

|  |  |  |  |
| --- | --- | --- | --- |
| 설정(생성) 대상 |  | Display()  \*객체를 화면에 표시 | WndProc()의 switch 부분  \*외부 input(키보드, 마우스)에  대한 output 설정 |
| 공(노란색) |  | -공의 변화를 화면상에 표시하기 위해 기준시간마다 공을 업데이트  CSphere::ballUpdate(기준시간)  -공이 벽에 부딪히는지를 확인하고 그에 따른 행동 실행  CWall::hitBy(공)  -파란공과 만나면 멈춤  CSphere::setPower(0,0) | -처음에 실행될 때(spacebar누른 직후) srand()와 rand()를 이용하여 노란 공의 운동량을 적정범위 안에서 랜덤으로 설정한다.  -CSphere::setPower() |
| 공(파란색) |  | -노란색 공과 만나는지 확인  CSphere::hasIntersected(파란공) | -방향키에 따라 파란공의 위치를 이동시킨다.  CSphere::setCenter(),  CSphere::getCenter() |
| 공의 운동범위를 표시하는 벽 |  | -기준시간마다 시간을 받은clock() 함수를 통해 받은 시간을 기준시간마다 업데이트해줌으로써 play 시간을 표시  CText::Print() | -space바를 실행시키면 그때부터 clock()이용하여 시간 측정 시작 |

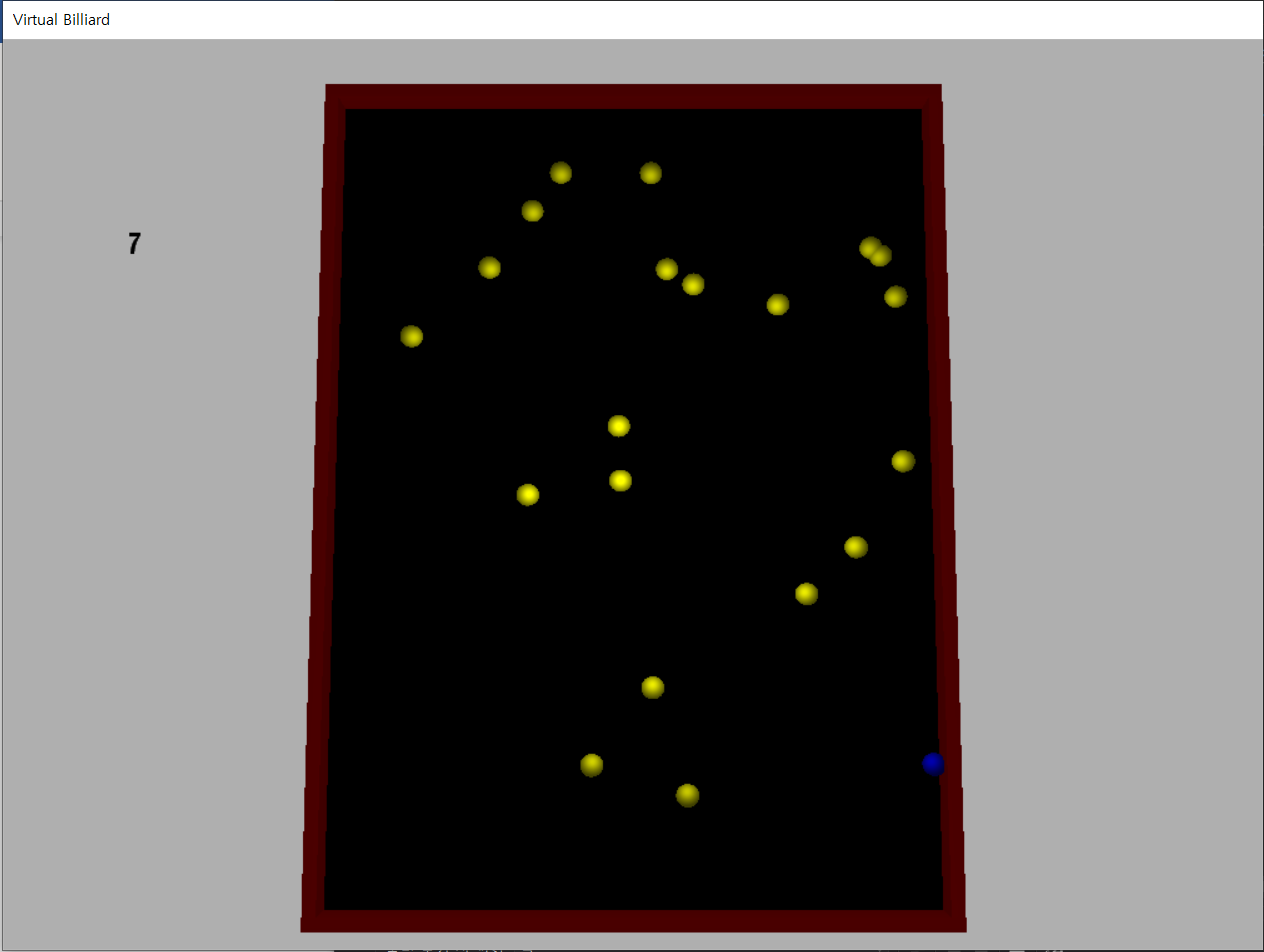


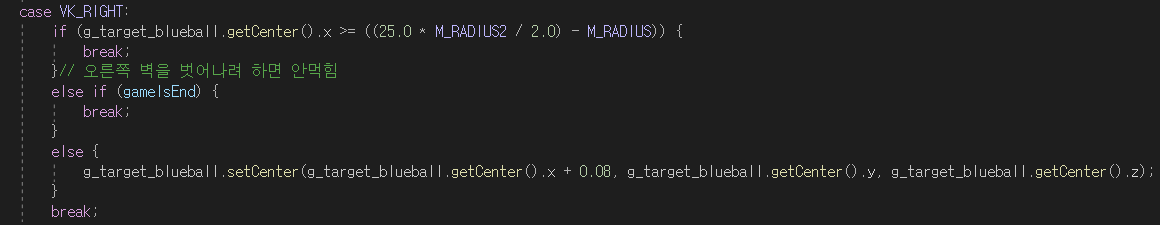
WndProc 함수 안에 있는 switch문. 게임을 실행시키기 위해 스페이스바를 누르면 타이머가 시작된다.



Display() 함수 안에 있는 if문. 계속해서 시간을 받아와서 흐르는 시간을 화면에 출력한다.

※파란공은 사각형 틀 안에서만 움직일 수 있으며 범위 밖으로 나가는 것은 불가능하다.





예시로 오른쪽 방향키를 누를 때의 switch문을 보면 blue공의 center가 정해진 범위 밖으로 나가면 break로 인해 case문 밖으로 나가기 때문에 아무런 변화가 일어나지 않는다.

1. 게임을 진행하다 파란 공이 노란 공과 만나면 게임을 종료한다.

* 게임이 종료되면 Game Over이라는 문구와 함께 몇 초 동안 살아남았는지, 그 기록을 출력해준다.



|  |  |  |
| --- | --- | --- |
| 설정(생성) 대상 |  | Display()  \*객체를 화면에 표시 |
| 게임 종료  문구 |  | - 파란공과 노란공이 만났는지 조사  (CSphere::hasIntersected())하는 함수의 값이 true가 되면 Csphere의 hasIntersected() 함수에서 clock()함수를 이용하여 그 때의 시간을 측정하여 얼마나 살아남았는지 계산하여 시간과 함께 문구 text띄움.  CText::Print() |

* 실행 방법

**※부가적 설명① 기존에 제시된 파일에선 공과 공이 만나면 어떻게 할 것인지는 hitBy()에서 구현했는데 프로젝트를 진행하며 hasIntersected()의 return값을 받아 Display()에서 구현하게 변경한 이유:**

class끼리의 independency를 증가시키기 위하여cohesion을 줄이기 위함이다.

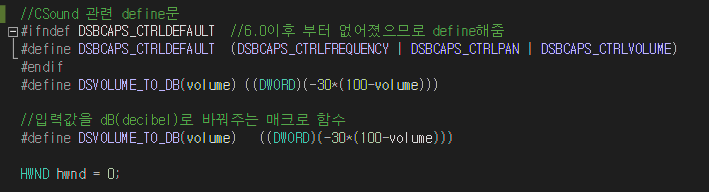
지금 이 프로그램에서 공과 공이 intersected가 되면 1)공의 운동량이 0이 되어야 하고(CSphere class), 2)clock()함수로 멈춘 시간을 받아와야 하고 3)게임이 종료되었다는 문구를 띄워야 한다(CText class). 그런데 만약 이를 CSphere class의 hitBy에서 구현하면 hitBy함수 내부에 CText class의 함수인 Print()가 사용되어야 하기 때문에 두 class간의 dependency가 증가하여 oop의 목적을 이루기 힘들 것 같다고 판단하였다. 때문에 workspace인 VirtualLego의 Display에서 hasIntersected를 이용해 구현하게 되었다.

**※부가적 설명② 타이머 구현 방법**

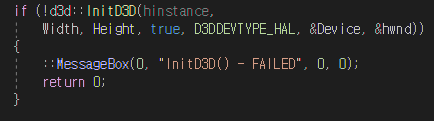
타이머는 게임 실행 중에 화면 왼쪽에 뜨는 text를 말하는 것인데 시간이 흐르는 것을 보여준다. 이는 먼저VirtualLego의 WndProc()함수에서 키보드의 SpaceBar가 눌렸을 시 그때의 시간을 clock()함수를 이용하여 begin 변수에 저장해준다. 그리고 Display()에서 단위 시간마다 clock()함수로 시간을 받아와 realend변수에 저장해준다. 그리고 이 둘을 이용해 계산한 결과 또한 Display()함수에서 단위시간마다 CText의 Print()함수를 이용하여 출력해준다.

**※부가적 설명③ 소리 구현**

소리는 CSound class를 추가하여 구현하였다. 이를 위해 관련 메크로상수와 HWND 변수를 선언해준다. (HWND는 창의 정보에 관한 변수이다)

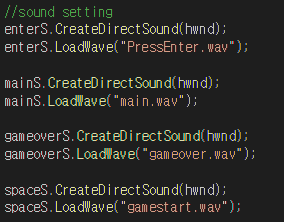


또 소리를 받기 위해선 게임 실행으로 인해 열리는 window창의 정보를 받아와야 하므로 Virtual Lego의 WinMain 함수에서

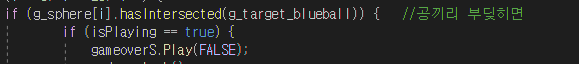


이렇게 현재 윈도우창의 정보를 받아오는 &hwnd를 매개변수로 추가해주어야 한다. 이렇게 추가해주기 위해 d3dUtility.cpp파일의 모든 HWND를 포인터 변수로 변경한다.

이렇게 코드를 다 변경해준 뒤 각 음악의 객체를 선언하고 Setup 함수에서 아래와 같이 객체를 초기화해주고(CSound::CreateDirectSound) 파일과 연결해준다(CSound::LoadWave).



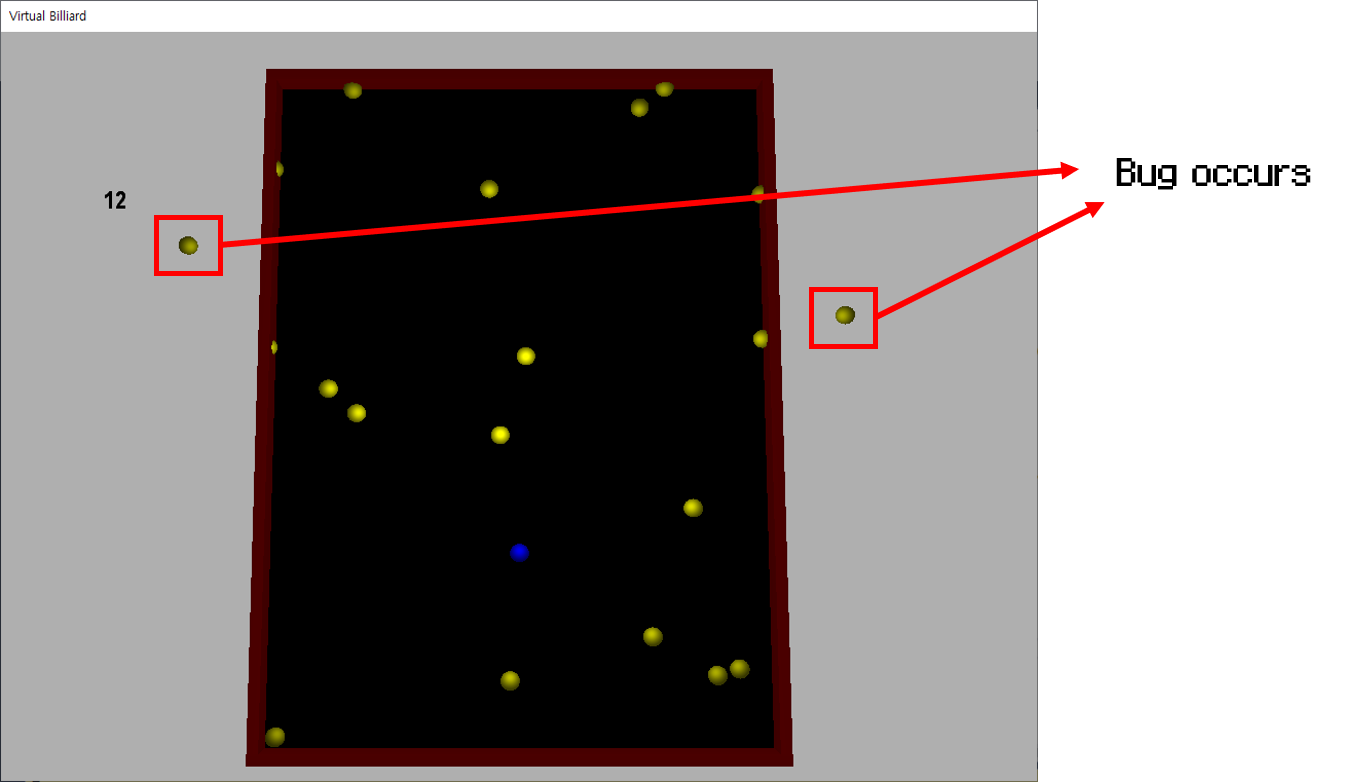
그리고 아래와 같이 사용해야하는 부분에 CSound::Play()를 FALSE를 인자로 넣어주면 실행이 된다.



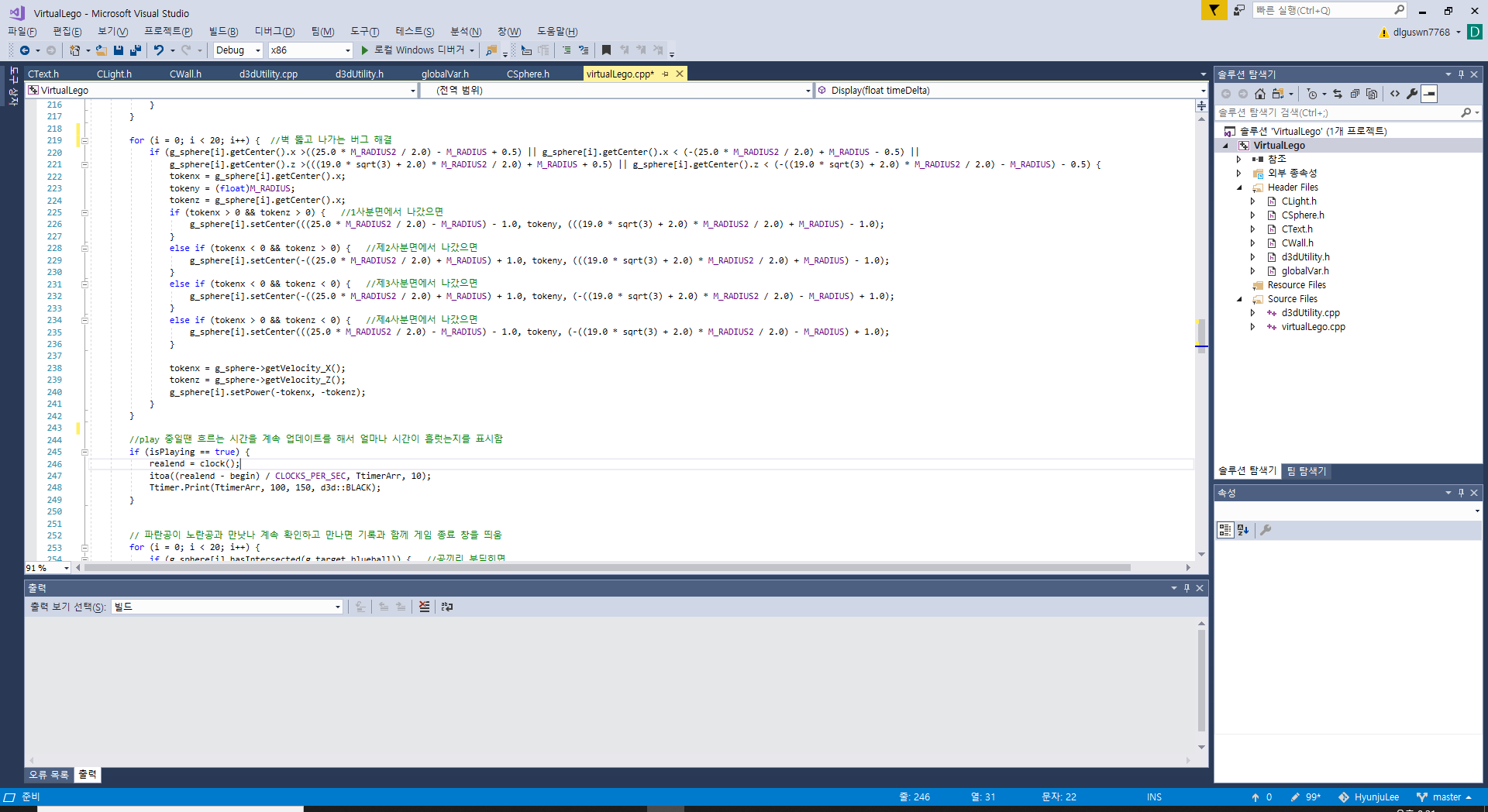
게임에서 bgm이 없으면 무언가 심심한 느낌이 드는 것 같아 구현해보았다.

1. **기능 구현 중 생긴 문제점과 해결방법**

게임을 진행하는 과정에서 CWall class의 hitBy 메소드가 정상적으로 호출되지 않아 가끔 공이 벽을 뚫고 게임 화면 밖을 벗어나는 버그가 발생하며 이 버그를 수정하지 않으면 게임 진행 과정에서 노란색 공격 공의 수가 줄어들어 게임 난이도가 시간이 지날수록 하락하게 되며 결국 게임 창에 어떠한 노란색 공도 남지 않아 게임이 종료되지 않는 심각한 상황을 초래하게 된다.



이 게임의 특성상 공의 생성 위치 및 운동량이 랜덤이어야 하고 단 화면 내에 있는 공의 개수가 줄어들어서는 안되는 점에 집중하여 버그로 인해 게임 화면을 벗어나게 되는 공은 그 즉시 잡아내어 벗어난 좌표에서 재생성 되어 반대 운동량으로 다시 게임 화면 안에서 기능을 수행하도록 코드를 추가하였다.

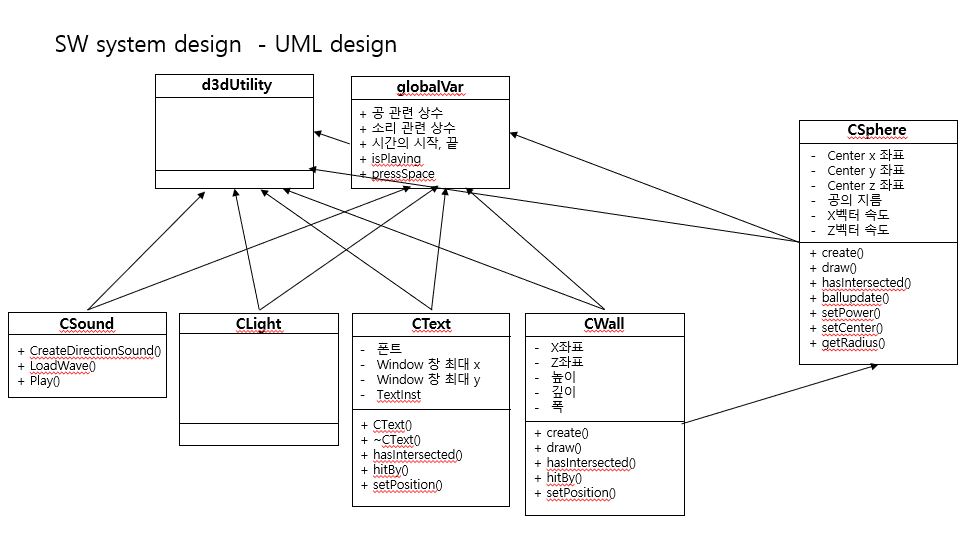


그 결과 게임 진행 내내 게임 화면 내에서 공의 개수가 줄어드는 일이 없어 문제없이 게임을 진행할 수 있음을 확인할 수 있다.

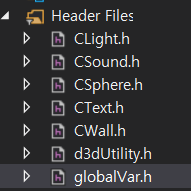
1. **sw 시스템의 객체지향적 디자인**

-간단한 class diagram

\* 우리가 새로 구현하지 않고 제공된 코드를 그대로 사용한 class의 내부 변수와 함수는 생략하였다. 또 우리가 사용하지 않은 함수와 변수는 생략하였다.



객체지향적으로 프로그래밍을 하기위해 원래 VirtualLego.cpp에 모두 있던 class들을 아래와 같이 헤더파일로 분류해주었다.



위의 Class를 간략히 설명하자면 아래와 같다.

1. D3dUtility: 제공된 Class에서 HWND를 받아오고 사용하기 위해 변수를 추가해주고 포인터로 변경해준 것 외에는 제공된 program과 동일하게 사용하였다. 이 class는 모든 다른 class가 참조한다.
2. globalVar: 원래 제공된 program을 헤더파일로 다 분류를 하다 보니 모든 class에서 사용하는 메크로상수들 또한 따로 헤더파일로 만들어 그것을 사용해야하는 class에 include 해줘야 하는 필요성을 느껴 method는 없는, 변수 및 상수 선언만 있는 class를 만들게 되었다. 모두가 사용해야하는 메크로 상수여서 모든 class가 이 class를 참조하고 있다.
3. CSound: 게임을 구현하던 중 소리를 추가하면 완성도가 높을 것 같아 제공되지 않은 새로운 class를 구현하였다. 이 class는 “소리”를 생성, 초기화, 멈춤, 음악 파일과 프로그램을 연결하는 등의 역할을 한다. 이 class를 구현하기 위한 메크로 상수와 library를 globalVar class에 추가해 주었다.
4. CLight: 원래 제공된 program을 그대로 사용하였다.
5. CText: 훨씬 사용자가 이해하기 좋고, 사용하기 쉬운 interface를 구현하기 위해 텍스트를 추가하는 것이 꼭 필수적이라 생각하여 제공되지 않은 새로운 class를 구현하였다. 이 class는 text를 생성, 초기화, 삭제, 내용추가 등을 하는 역할을 한다. 이 class를 구현하기 위해 csting 라이브러리를 추가하였다.
6. CWall: 주어진 program에서 hitBy method와 hasIntersected method를 만들어 사용하였다. 이 class는 벽을 생성, 초기화, 그리기, 벽과 공이 부딪혔는가, 벽과 공이 부딪혔다면 어떠한 action을 취하는가 등의 역할을 수행한다. 이 class는 공과의 충돌여부를 판정해야하므로 CSphere class를 참조하였다.
7. CSphere: 주어진 program에서 hasIntersected method를 만들어 사용하였다. 이 class는 CWall과 비슷하게 공을 생성, 초기화, 그리기, 공의 위치와 운동량 control, 공끼리 부딪혔는지 여부를 판단하는 등의 역할을 수행한다.

위와 같이 많은 class를 나누고 생성하고 object를 생성하고 사용하는 과정을 통해서 아래와 같은 점을 느끼고 배우게 되었다.

1. Encapsulation의 필요성을 느꼈다. 초반에 어떠한 한 class와 연관되는 변수를 class내부에 private으로 생성하지 않고 외부에 global 변수로 생성하여 사용하였다. 생성 후에 많은 코드를 작성 수정하다 보니 그 변수의 값이 계속해서 변하는 바람에 원하는 결과가 나오지 않아 고생하였는데 그 이유가 global변수로 선언하였기 때문임을 알게 되어 encapsulation의 필요성을 몸소 체험하며 깨닫게 되었다.
2. Reusability의 필요성과 큰 프로젝트를 진행할 때 그 중요함에 대하여 알게 되었다. 이 게임을 프로그래밍하면서 상당히 많은 동일한 class의 객체를 생성하였는데 만약 이를 object oriented programming이 아닌 다른 모든 method와 변수를 필요할 때마다 생성하거나 재정의했어야 했다면 아마 코드가 지금 보다 2배이상 길었을 것이다.
3. **결론**

Setup(), Display(), setPower(), getCenter(), WndProc(), hitBy(), hasIntersected() 등의 다양한 메소드와 CWall, CSphere, CText, CSound 등의 여러 클래스들을 생성 및 수정하고 기능과 역할별로 필요한 class를 나누어 서로의 independency를 높이고 module화가 적절히 이루어질 수 있도록 프로그래밍을 하고자 노력하였다.

곰플레이어에 있던 “닷지” 게임을 주어진 코드를 응용하여 필요한 기능과 인터페이스를 성공적으로 구현하였다. 또한 더 적합한 게임 환경을 구현하기 위해 여러 문제를 해결하고, 새로운 CText, CSound 클래스를 생성하여 실감나는 게임 효과를 구현하였다.

그리고 더 구현하고 싶었으나 시간 또는 능력 부족으로 인해 하지 못한 아쉬운 부분은 다음과 같다.

* 사용자명을 입력 받아 게임 기록을 저장하고 상위 플레이어의 게임 랭킹을 인터페이스에 표시하는 기능
* 재디버깅하지 않고 리플레이가 가능하도록 구현하는 기능
* 원래 오리지널 닷지 게임을 비슷하게 구현하기 위해 배경을 회색이나 검정색이 아닌 우주느낌의 외부 image파일을 끌어와서 구현

이번 프로젝트를 통해 객체지향적으로 프로그래밍하기 위해 class의 성격에 맞게 method사용과 구현, 변수 사용과 선언하기 위해 많이 고민하는 과정을 통해 객체지향 프로그래밍을 하는 방법과 그 성격에 대해 좀 더 깊이 있게 이해할 수 있게 되었다.