|  |  |  |
| --- | --- | --- |
| 이름 | 백성빈 | 손주은 |
| 학번 | 20170680 | 20170702 |
| 학과 | 컴퓨터공학과 | 컴퓨터공학과 |
| ID | sungbin0515 | jeson |

CSED 451 Computer Graphics--Assn1

**Assn 1: 피카츄 배구**

|  |
| --- |
| OpenGL 2D 드로잉 기술을 활용해 간단한 피카츄 배구를 구현하라. |

# 개발 환경:

**IDE:** Visual Studio 2017 community

**Compile mode (VS 2017):** Release x86

**Dependency**: OpenGL / GLEW 2.1.0 / Freeglut 3.0.0

# 요구 사항

## Ball

* 공은 점수 박스를 제외한 모든 객체와 충돌하며, 입사각과 반사각을 고려한 방향으로 반사된다. 이때 중력은 고려할 필요 없다.
* 게임 시작, 혹은 한 플레이어가 득점을 할 때 공은 필드의 중심에 생성된다. 그리고 무작위 높이에서 일정한 속도로, 무작위 각도로 이동한다

## Character

* 플레이어의 키보드 조작을 통해 좌우 두 방향으로 움직일 수 있다. 점프 기능은 구현할 필요 없다.
* 플레이어의 위치는 네트의 왼쪽이며, 오른쪽의 플레이어는 캐릭터가 공의 위치에 따라 자동으로 움직이되, 간헐적인 실수를 구현하여 플레이어가 점수를 얻는 것이 가능하게 한다.

## Net, Boundary

* 네트의 상단은 충돌을 고려하지 않아도 된다.
* 모든 충돌은 반발계수가 1인 완전 탄성 충돌로 고려하여 공의 속도가 유지되도록 한 다.

## Window(zoom-in / out)

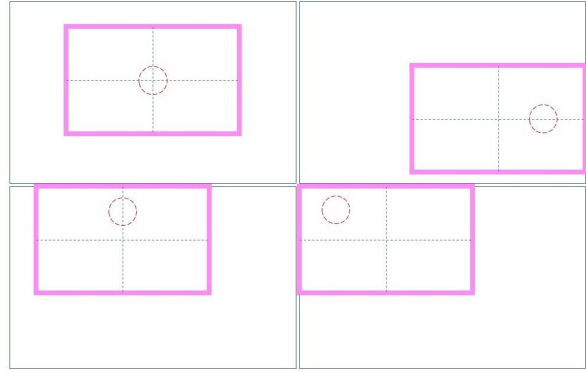
* 적어도 2개 이상의 화면 모드가 있어야 한다.
* 전체화면: 게임공간 경계를 아우르는 window로, 화면의 이동이 없다.
* 부분화면: 공을 중심으로 하는 window로, 화면의 중심에 공이 오도록 하지만, window 중심에 공이 올 수 없을 정도로 공이 경계에 가까이 갈 시에는 window가 게임 공간의 경계를 벗어나지 않도록 제어한다.
* 

Figure Position of clipped window

# **프로그램 설계**

1. **AABB (axis-aligned bounding box) (VriesJoey, 날짜 정보 없음)**

**AABB는 axis-aligned bounding box**의 약자로, World Coordinate의 축과 평행하게 물체를 둘러싸고 있는 사각형의 영역을 의미한다. AABB가 World Coordinate와 항상 평행하므로, 물체 간의 충돌을 편리하고 일관적으로 관리할 수 있다는 장점이 있다.

이번 피카츄 배구에서는 사각형 영역에 더해, 언제나 축과 평행하다고 할 수 있는 원형 영역을 추가함으로써, 충돌이 조금 더 사실적으로 이루어지게 만들었다.

충돌은 사각형과 원, 원과 원, 사각형과 사각형 간의 충돌 세가지로 나눌 수 있다. 먼저 사각형 - 사각형 충돌의 경우를 보자. 이번 과제의 경우에는 플레이어나 네트의 수직적인 움직임이 없기 때문에 horizontal 움직임을 분석함으로써 충돌을 감지할 수 있다. 사각형의 horizontal 영역에 다른 사각형 object의 좌측하단의 꼭지점이 들어가있을 때, 충돌이 일어났다고 판단했다.

두번째는 원과 사각형의 충돌이다. 이 경우는 공(Ball)이 네트나 플레이어와 충돌하는 경우이다. 사각형 객체에서 좌, 우측과 상측으로 공의 반지름만큼 키운 사각형을 지정해두고 이 영역 안에 원의 중심의 좌표가 들어오면 1을 return 하는 함수를 작성했다. 원의 중심을 기준으로 비교할 것이기 때문에 기존 사각형에서 좌우와 위쪽방향으로 반지름만큼 키웠다.

마지막으로 두 원 간의 충돌이다. 이 경우는 두 원의 중심을 잇는 방향 벡터와 x축 간의 각도를 라고 하면, **- 도 회전 연산을** 통해 두 원의 접선을 y축으로 하는 새로운 좌표계로 전환한다.

그리고 전환한 좌표계의 X축의 좌푯값에 -1을 곱하는 **완전 탄성 충돌 연산을** 수행한다.

마지막으로 전환한 좌표계를  **도 회전 연산을** 통해 기존의 좌표계로 전환한다. 이 과정에서 연산이 너무 빠르게 일어나면 두 원이 서로 붙어 있는 문제가 생기므로, 회전 연산을 수행한 후 물체의 속도 벡터의 x축 방향 값이 음수일때만 -1을 곱하는 조건부 연산을 취한다. 이를 행렬 연산으로 나타내면 다음과 같다:

1. **Scoring**

공 객체의 (원의 중심) y 좌표가 공의 Radius 이하로 내려가면 점수를 update하게 만들었다. 이때의 x좌표을 읽음으로써 화면의 어느 쪽에 떨어졌는지에 따라 Player1과 Player2 (컴퓨터)의 점수를 1씩 증가하고, 배구 규칙을 따라 한쪽의 점수가 15점이 되면 승패 판정을 내린 뒤 이를 화면에 보여준다. 그리고 게임을 정지한다. 'glutBitmapCharacter()' 함수를 이용해서 화면에 점수를 출력할 수 있었다.

1. **Structure**

이 프로그램을 구성하는 Class는 크게 물체를 관리하는 Class, 충돌을 관리하는 Class, 게임 전체를 총괄하는 Class로 구분할 수 있다.

* Object Class
  + 게임에 등장하는 모든 물체(Net, Player, Ball)가 상속한 Base Class
  + 위치, 속도, 가속도를 관리할 수 있는 인터페이스를 가지고 있으며, 한 Object Class에 여러 AABB를 할당할 수 있도록 하여 복잡한 Collision Boundary도 관리할 수 있게 작성했습니다.
  + 매 프레임마다 자기 자신을 그리는 함수인 Draw(), 매 프레임마다 이동거리 등을 계산하는 Step(int dt) 함수, 그리고 다른 물체와 충돌했을 때 물리 연산을 수행하는 onCollide method를 virtual method로 선언해 상속받는 모든 Class로 하여금 이를 구현할 수 있도록 했습니다.
* AABB Class
  + 게임에 등장하는 물체의 충돌 범위를 관리하는 Class인 RectAABB과 CircleAABB가 상속한 Base Class
  + 크기와 다른 AABB와 충돌했는지 여부를 계산할 수 있는 인터페이스를 기본적으로 제공합니다.
  + RectAABB와 CircleAABB 두 종류의 Derived Class를 선언, 각 Class 상에서 다른 AABB와 충돌 여부를 감지하는 method를 AABB의 종류에 따라 다르게 구현, 이를 함수 포인터를 활용해 일관적으로 관리하도록 작성했습니다.
* GameManager Class
  + 게임의 시작과 끝, 매 프레임 별 물리연산, 오브젝트 소환 및 관리, 점수 체계를 총괄하는 Class입니다.
  + 오브젝트를 추가, 제거, 탐색할 수 있는 인터페이스를 제공하며, 매 프레임마다 등록된 모든 오브젝트의 물리 연산 + 충돌 연산을 수행합니다. 그리고 모든 오브젝트를 화면에 그리고, Graphic Buffer를 Swap하면서 이를 동시에 화면에 그림으로써 자연스러운 움직임을 그릴 수 있습니다.
* Main
  + 맨 처음에는 OpenGL과 GLUT 관련 초기화를 수행하고, 여러 오브젝트를 생성합니다. 그리고 게임을 시작합니다.
  + GLUT Library에 제시된 glutTimerFunc를 활용해서 지정된 주기로 Step()을 호출해서 프레임 단위 연산을 수행합니다. 그리고 한 플레이어(플레이어 or 컴퓨터)의 점수가 미리 정의한 점수인 15점이 되면 공의 속도를 0으로 조율하고, 한쪽이 이겼다는 텍스트를 출력합니다.

1. **How to use**

이 게임의 목적은 플레이어가 피카츄 (처럼 생긴 도형) 캐릭터를 좌우로 조종함으로써 공이 왼쪽 영역의 땅에 닿지 않도록 하는 것이다. 게임을 시작하면 간단한 게임의 조작방법과 승리조건이 화면에 등장한다. Esc 키를 누르면 게임을 시작할 수 있으며, 키보드의 'z'와 'c' 키를 통해 각각 좌와 우로 이동할 수 있다. 'x' 키를 누르면 공이 중앙에 오는 화면 모드로 전환할 수 있다. 이때 공이 World의 구석으로 갈때 화면은 world의 가장자리 너머를 보여주지 않고 공은 더 이상 화면의 중앙에 위치하지 않게 된다. 플레이어나 컴퓨터가 15점을 얻게 되면 화면에 승부 결과를 알리는 창을 띄움으로써 게임은 종료된다.

# **시연**

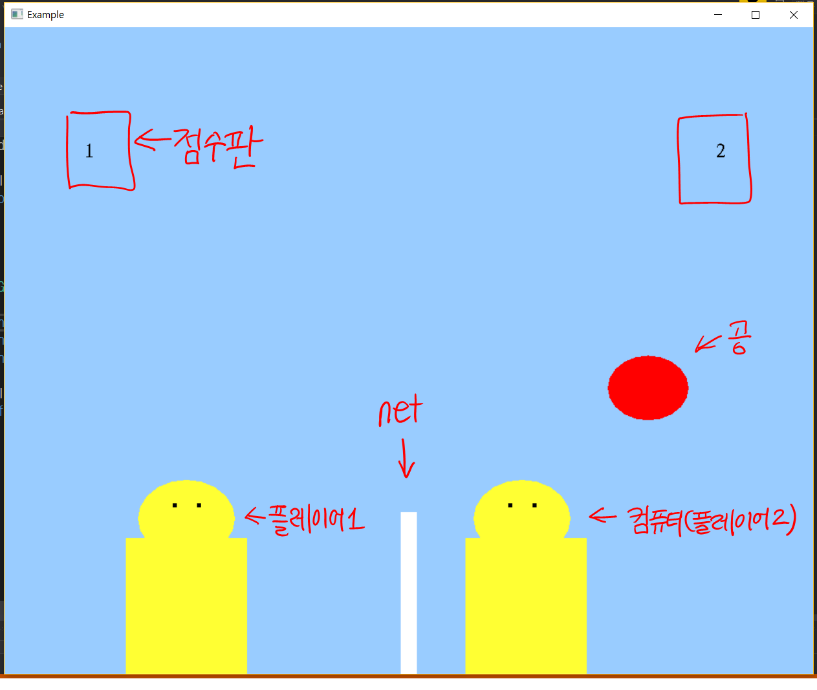


Figure 2 프로그램을 실행하면 플레이어, 공, 네트, 점수판으로 이루어진 화면이 출력된다.

'z'와 'c' 키를 눌러 좌우로 이동할 수 있다.

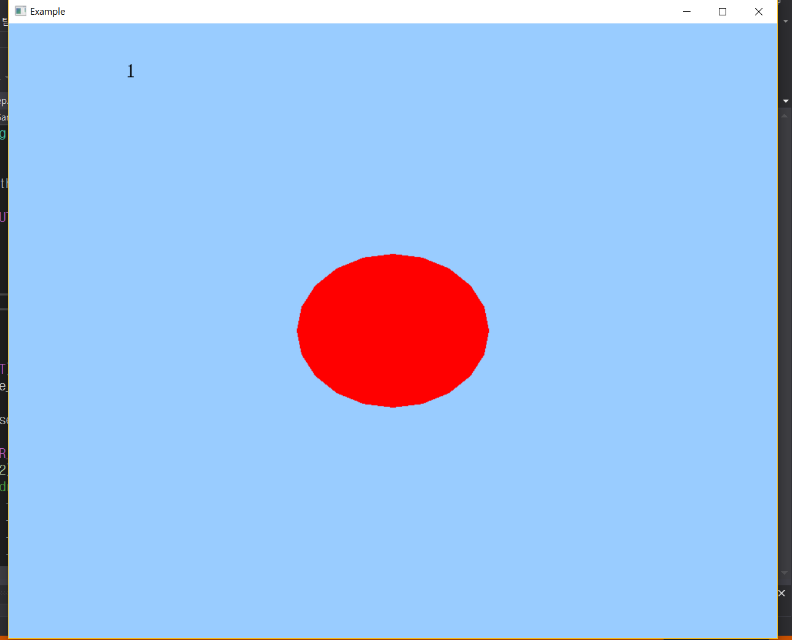
ㅇ

Figure 3 'x' 키를 눌러 화면 중심에 공이 오게 하는 모드이다.

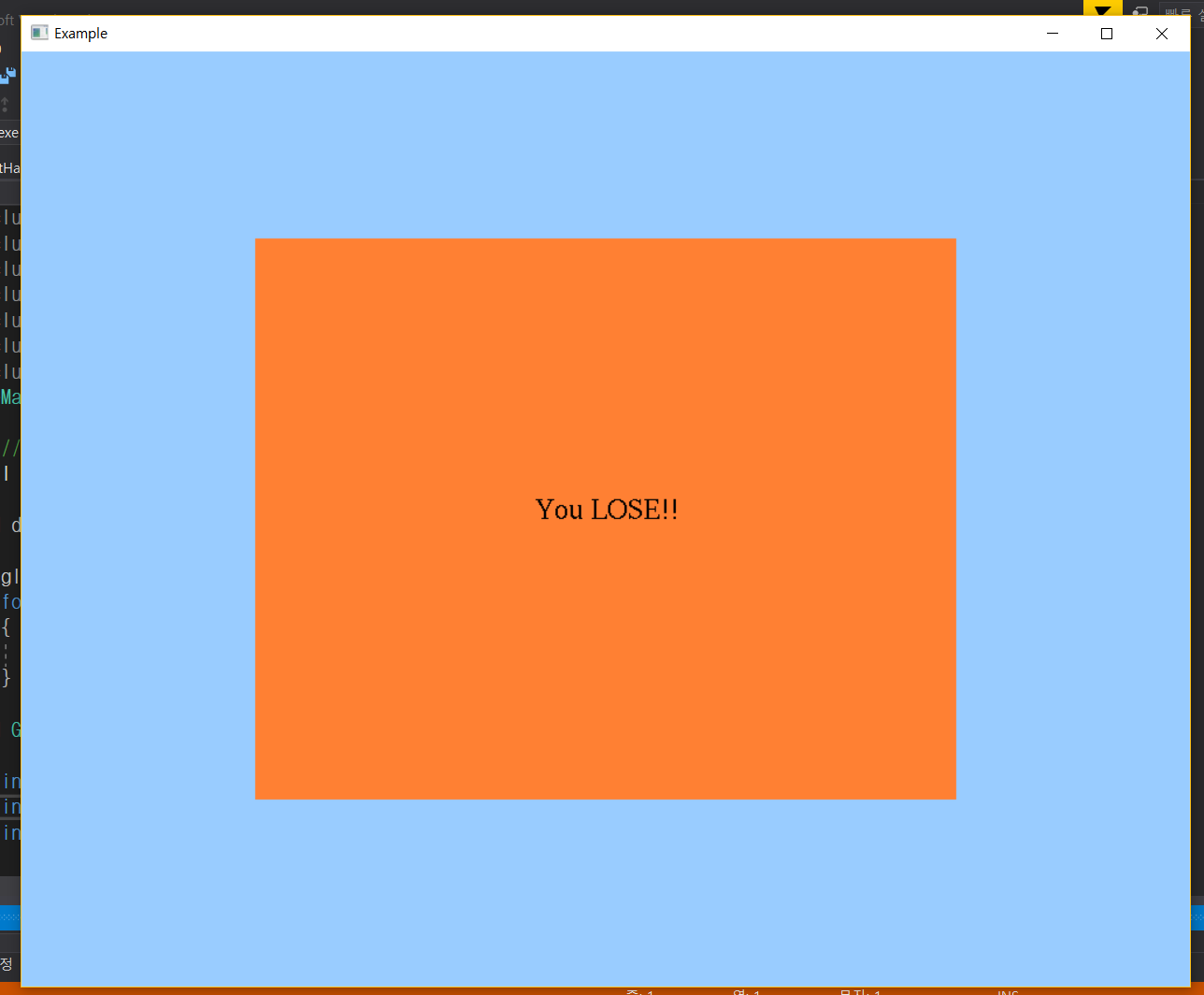


Figure 4 컴퓨터나 플레이어가 15점을 얻으면 게임은 종료되고 승부 결과를 알려주는 창을 띄운다.

# 토론

프로그램을 구현하면서 물리 연산을 어떻게 구현할지 토론을 진행했고, 이미 널리 상용화된 Unity 3D 게임 엔진이나 Box2D 물리 엔진의 구조를 참고하자는 결론을 얻었다. Object에 연결된 AABB에 대해 기본적인 Method나 Field가 선언되어 있는 AABB Base Class와 오브젝트의 전반적인 구조를 지시하는 Object Base Class를 만들고 이를 바탕으로 피카츄 배구를 구성하는 여러 오브젝트의 Class를 만들었다. 또한 충돌 감지는 AABB를 활용해서 일관적으로 수행하고, 충돌 후 물리연산은 2차원 상에서 매우 단순화된 완전 탄성충돌을 하도록 구현했다.

오브젝트를 스크린에 띄우는 과정은 다음과 같이 구현했다: 먼저 모든 오브젝트를 저장하는 Class를 생성하고, 이 Class로 하여금 저장된 Object를 순회하면서 각 오브젝트 자신이 glut 라이브러리를 활용해서 스크린 상에 오브젝트를 그리게 하였다. 모든 오브젝트 Drawing이 끝나면 Graphic Cache를 Swap 해줌으로써 모든 오브젝트가 그려진 화면을 Screen에 띄웠다. 이 과정을 미리 정해진 Frame 주기(20ms)마다 반복했으며, 그 결과 부드러운 움직임을 그려낼 수 있었다.

# 결론

이번 과제를 통해 OpenGL과 GLUT를 활용해서 Window 창에 2D 그래픽 요소를 그리는 방법을 연습해 볼 수 있었다. 그리고 물리적 요소(충돌)가 들어가는 게임을 만들때는 구조를 체계화 해서 작성하면 프로그램을 유연하게 작성할 수 있음을 확인했다.

# 개선 방향

현재 구현 구조는 Hierarchical Object를 제작하기 번거롭고, 또 구현한다 하더라도 일관적이지 않다. 따라서 시스템적으로 이 부분에 대한 개선이 필요할 것으로 보인다.

피카츄 배구라고 보기에는 그래픽도 저급하고, 점프 및 스파이크 등 배구에서 사용되는 기능 역시 구현되어 있지 않다. 이를 구현해서 게임을 조금 더 개선할 수 있을 것으로 보인다.

# 인용 자료

VriesdeJoey. (날짜 정보 없음). “Learn OpenGL”. Collision Detection: https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection에서 검색됨