|  |  |  |
| --- | --- | --- |
| 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)

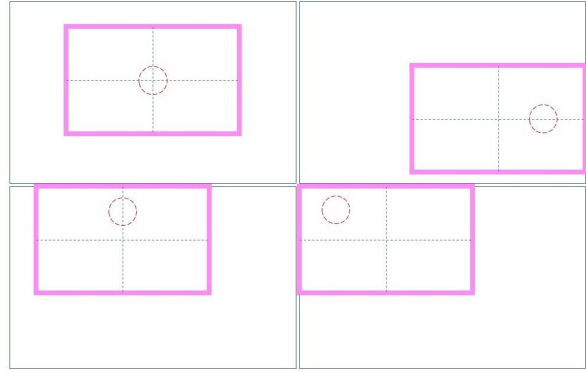
* 적어도 2개 이상의 화면 모드가 있어야 한다.
* 전체화면: 게임공간 경계를 아우르는 window로, 화면의 이동이 없다.
* 부분화면: 공을 중심으로 하는 window로, 화면의 중심에 공이 오도록 하지만, window 중심에 공이 올 수 없을 정도로 공이 경계에 가까이 갈 시에는 window가 게임 공간의 경계를 벗어나지 않도록 제어한다.
* 

Figure 1 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.

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 increment했다. 'glutBitmapCharacter()' 함수를 이용해서 화면에 점수를 출력할 수 있었다.

1. How to use

이 게임의 목적은 플레이어가 피카츄 캐릭터를 좌우로 조종함으로써 공이 왼쪽 영역의 땅에 닿지 않도록 하는 것이다. 게임을 시작하면 간단한 게임의 조작방법과 승리조건이 나온다. 게임을 시작할때 플레이어의 캐릭터는 왼쪽에 위치하게 된다. 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**

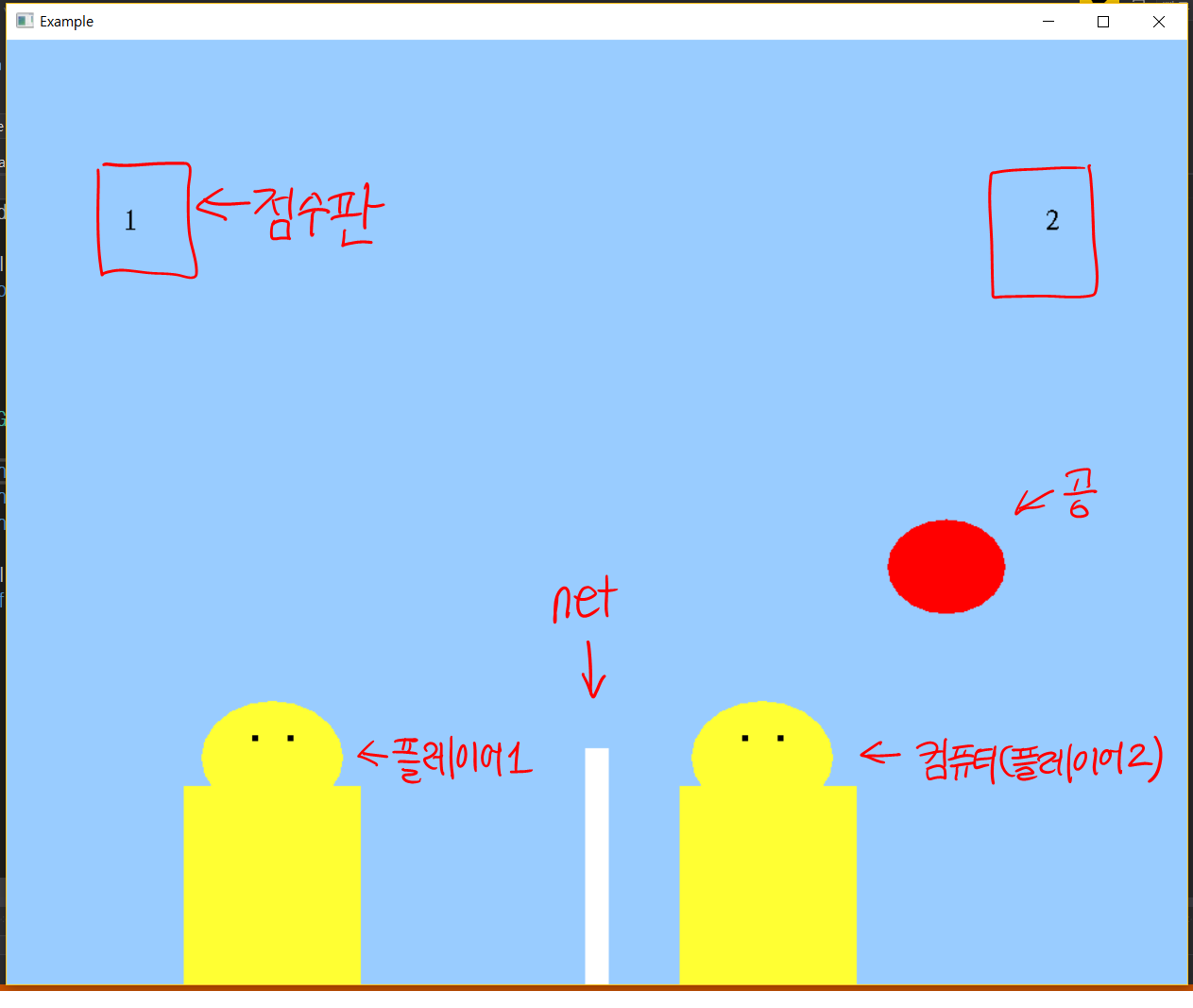


Figure 2 프로그램을 실행하면 플레이어, 공, 네트, 점수판으로 이루어진 화면이 출력된다.

'z'와 'c' 키를 눌러 좌우로 이동할 수 있다.

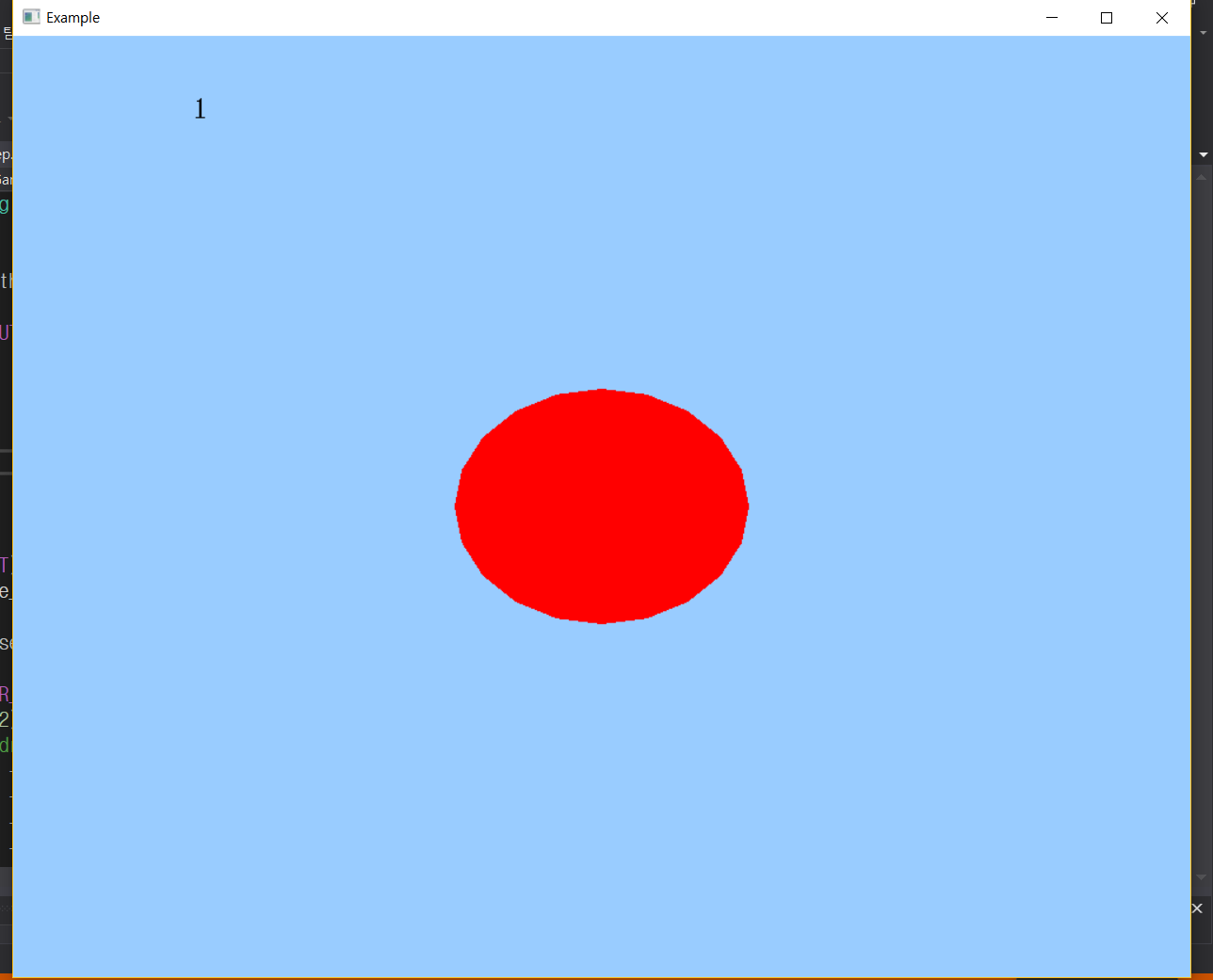
ㅇ

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

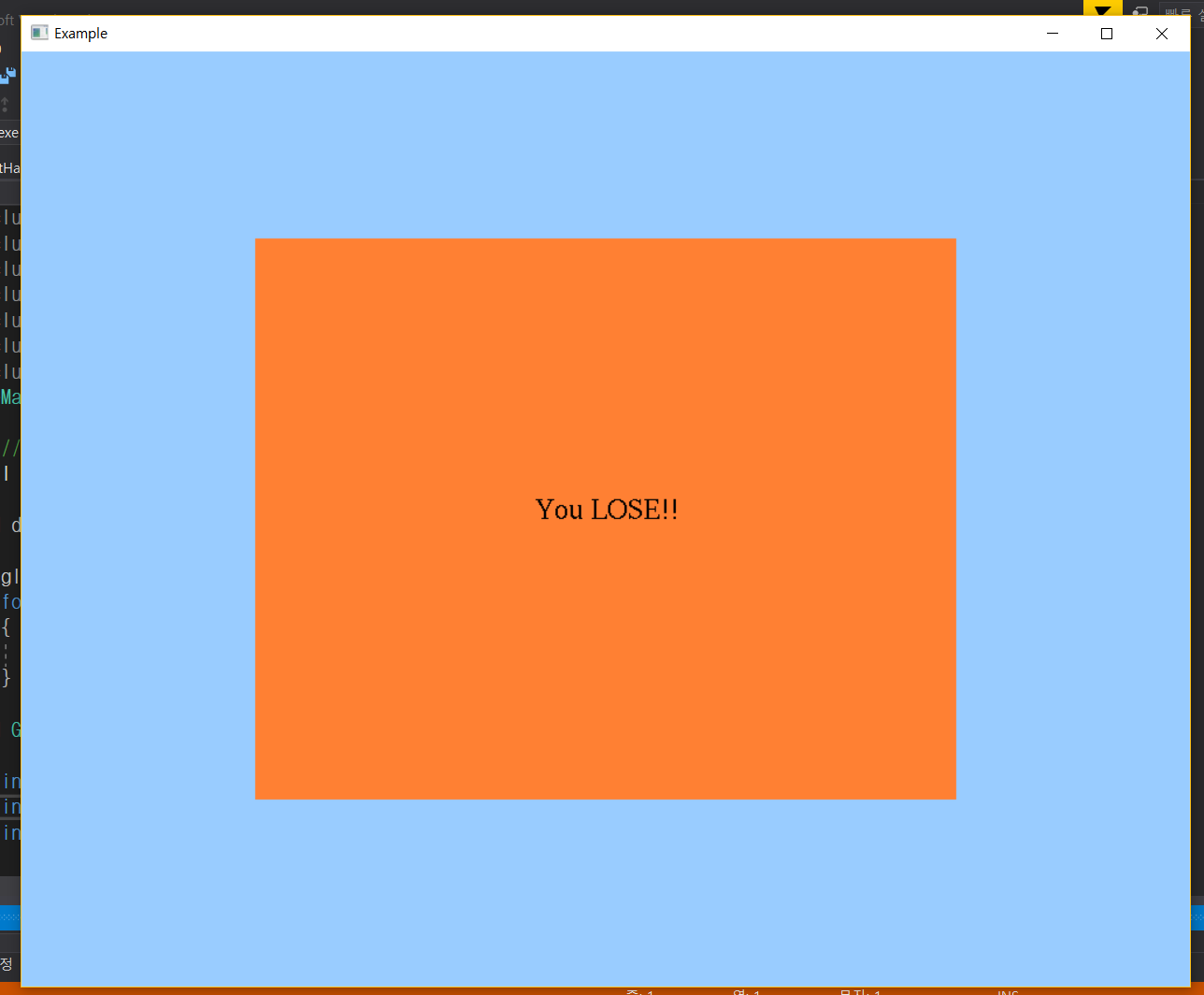


Figure 4 컴퓨터나 플레이어가 15점을 얻으면 게임은 종료되고 승부 결과를 알려주는 창을 띄운다.

# 토론

* 이번 Assign의 요구 사항 중 하나는 프로그램 상에서 Player Cell이 삭제되면 필드의 모든 생물체들이 사라진 뒤 플레이어 Cell이 사라지고 프로그램이 종료되도록 설계하는 것이다. 이를 위해 설계 단계부터 Player Cell Object에 다른 세 종류의 Object를 저장하는 링크드 리스트를 저장하도록 해서 플레이어의 삭제가 다른 Object의 삭제와 같은 타이밍에 일어나도록 할 수 있었다.
* 이번 Assign에서 Qt를 처음 접하고 사용해 보았다. 처음에는 Qt의 구조가 매우 난해했지만 조교님들이 제공해주신 강의 자료를 읽고 직접 코딩하면서 점차 Qt가 객체간의 정보 교환을 응용한 Signal – Slot 개념을 사용하는 매우 객체지향성이 강한 환경임을 알 수 있었다.

# 결론

* 이번 Assign에서는 CSED232 강의에서 배운 OOP의 개념 중 Inheritance, Polymorphism, Virtual Method 등을 사용해서 세포 증식 게임을 구현해 보았다. 그리고 GUI 구현을 위해서 객체지향성이 매우 강한 GUI 구현 환경인 Qt를 활용했다.
* Qt에 물체를 띄우기 위한 코드를 포함해서 대부분의 코드는 객체간의 정보 교환 및 그에 따른 행동을 나타내며, 실제로 Player Cell의 방향을 전환하기 위해 사용자에게 입력을 받는 것을 제외한다면 객체 간의 정보 교환을 통해 전체 프로그램이 구동된다. 가령, 15초마다 새로운 객체를 형성하는 작업의 경우, Cell Object의 data field 중 하나인 QTimer Objcet에서 매 15초마다 Cell Object의 NewCell Method를 호출하는 방식으로 이루어진다.
* 이 게임에서 등장하는 생물체는 4가지 종류가 있으며, 네 생물체는 공통적인 data, method와 각 생물체 별로 존재하는 특수한 data 및 method가 존재한다.. 공통적인 인자는 Inheritance 및 Polymorphism을 통해 하나의 상위 Class로 일관성 있게 관리했으며, 특수한 Data 및 Method는 dynamic casing을 통해서 접근할 수 있었다.
* 이처럼 이전(특히 CSED101에서)에 사용했던 절차 지향성 프로그래밍과 완전히 다른 패러다임인 객체 지향성 패러다임을 적용한 프로그램을 여태까지 배운 OOP의 하위 개념을 활용해서 구현해볼 수 있었다.

# 개선 방향

* 테스트를 진행한 컴퓨터의 성능이 좋지 않은건지, 아니면 최적화가 이루어지지 않아서인지는 모르겠지만 화면에 생물체가 100개 정도를 넘어가자 움직임이 뚝뚝 끊기며 거슬릴 정도로 렉이 걸렸다. 이는 Qt에 대해 잘 모른 채로 코드를 작성했기 때문으로 보이며, Qt의 구조를 더 자세히 이해한다면 불필요한 요소를 줄임으로써 렉을 줄일 수 있을 것이다.
* 또한, 이번 Assign에서 ‘크기’를 각 생물체의 지름 혹은 한 변의 길이로 정의하였는데, 이렇게 되자 생물체를 먹었을 때 상식적으로 예상했을 때 보다 훨씬 커지거나, 작아지는 경우가 자주 목격되었다. 지름이 아니라 넓이로 다시 정의한다면 더 현실감 있는 게임을 만들 수 있을 것이다..