CSED451 Assignment #1

2D DRAWING REPORT

Team: GPS

CSE 20202405 문민재 csmmj4594

CSE 20202728 김진수 fusion4268

(목차 14pt Bold, Indexing 12pt, 본문 10pt)

**Overview of Program**

**Programming Environment**

Visual Studio 2019

OpenGL - ???

Freeglut-MSVC-3.0.0

Glew-2.1.0

Glm-0.9.9.7

**Design and Implementation**

1. 자료구조 설계 (source.h)
2. Coordinates
3. Object

맵 상의 오브젝트를 정의하는 클래스이다.  
멤버 변수로 type, color, x, y를 가지고 있다. 이들은 각각 오브젝트 타입(0일 경우 rect, 1일 경우 character), 오브젝트의 색깔, 오브젝트를 그릴 때 필요한 x, y좌표에 해당한다.

1. Rect: public object

Object를 상속받아 맵 상의 직사각형 오브젝트(바닥, 벽)을 정의하는 클래스이다.  
rect 오브젝트들은 공통적으로 type이 0이며, x, y는 직사각형의 왼쪽 아래 점의 좌표를 의미한다. 추가적으로 width, height라는 멤버 변수를 가지는데, 이는 각각 직사각형의 가로 길이, 세로 길이를 의미한다.

1. Character: public object

Object를 상속받아 맵 상의 캐릭터 오브젝트(플레이어, 도둑)를 정의하는 클래스이다.  
character 오브젝트들은 type이 1에 해당하며, 화면 상에서 원형으로 표현되기에 x, y는 원의 중심을 나타낸다. 그리고 원의 반지름을 나타내기 위한 변수인 rad와 벽과의 충돌 여부를 나타내는 변수인 isCollided가 있다.

1. 기본 기능 구현
2. 화면 상 오브젝트의 출력

화면 상 모든 출력은 display() 함수에서 이루어진다.

drawRect, drawCircle, writeLife 등 화면에 표시되는 오브젝트의 종류마다, 그리고 화면에 적히는 글씨마다 출력에 필요한 행동이 다르기에 각각을 별개로 함수로 선언하고, 이러한 함수들을 display()함수에서 출력이 필요한 오브젝트마다, 그리고 출력이 필요한 글씨마다 호출하는 방식으로 구현하였다.

1. World 구현
2. 벽의 생성 및 이동

벽의 생성 및 이동은 idle function으로 설정된 moveWall() 함수에서 이루어진다.

moveWall 함수에서는 벽의 이동 속도인 wallSpeed에 맞게 벽을 이동시켜주고, 벽이 플레이어 혹은 도둑 캐릭터와 충돌했는지를 판정하여, 벽이 도둑과 충돌한 경우 벽의 색을 현재 도둑의 색으로 변경시켜주고, 벽이 플레이어와 충돌한 경우 플레이어의 색과 벽의 색을 비교하여 pass/fail 여부를 확인하고, 그에 맞는 action을 수행한다.

벽이 화면 밖(왼쪽)으로 벗어난 경우, 벽을 초기 위치로 이동시켜주어 벽의 재생성을 구현하였다.

1. 충돌 판정

오브젝트들의 충돌 판정은 collisionCheck(object\* a, object\* b) 함수에서 이루어진다. 오브젝트의 충돌은 크게 벽-플레이어/도둑의 충돌, 그리고 플레이어-도둑의 충돌 2가지 경우로 나뉜다.

벽과 플레이어 혹은 벽과 도둑의 충돌은 벽의 왼쪽 모서리가 원형 캐릭터의 중심을 지나는 순간 충돌했다고 판정하고, 캐릭터와 벽이 겹치는 동안 충돌의 중복 판정을 방지하기 위해 캐릭터의 isCollided를 true로 만들어주어서 중복 판정을 방지하였다.

플레이어와 도둑의 충돌은 두 원형 캐릭터가 서로 만나는 순간 충돌이 발생했다고 판정하였다.

1. 플레이어 및 도둑의 포즈 구현

플레이어의 포즈 구현은 방향키 입력을 받는 glutSpecialFunc를 이용하여 구현하였다. 위키를 누르면 빨간색, 아래 키를 누르면 초록색, 왼쪽 키를 누르면 파란색, 오른쪽 키를 누르면 플레이어 캐릭터가 노란색이 되도록 구현하였다.

도둑의 포즈 변경은 idle function인 moveWall 함수에서 구현되었다. 전역변수로 thiefFrame과 colorPeriod가 있는데, idle function인 moveWall이 호출될 때마다 thiefFrame이 1씩 올라가고, 그것이 colorPeriod 이상이 될 때마다 도둑의 색을 랜덤으로 재설정해주는 방식으로 구현되었다.  
플레이어가 벽을 통과하여 도둑과 가까워질수록 colorPeriod를 낮춰서 더욱 자주 포즈를 변경하도록 구현하였다.

1. 치트 기능

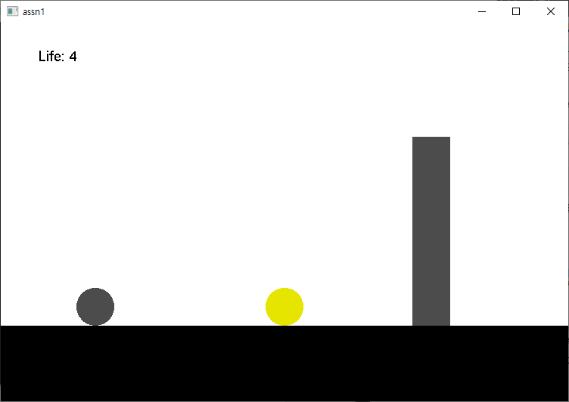
치트 기능은 glutKeyboardFunc를 이용하여 구현하였다. 과제의 지시사항대로 ‘c’를 입력하면 all pass 모드로, ‘f’를 입력하면 all fail 모드로 진입시키고 콘솔에 출력시켜 치트를 사용했음을 알 수 있게 구현하였다.

1. Pass/Fail 시의 Action

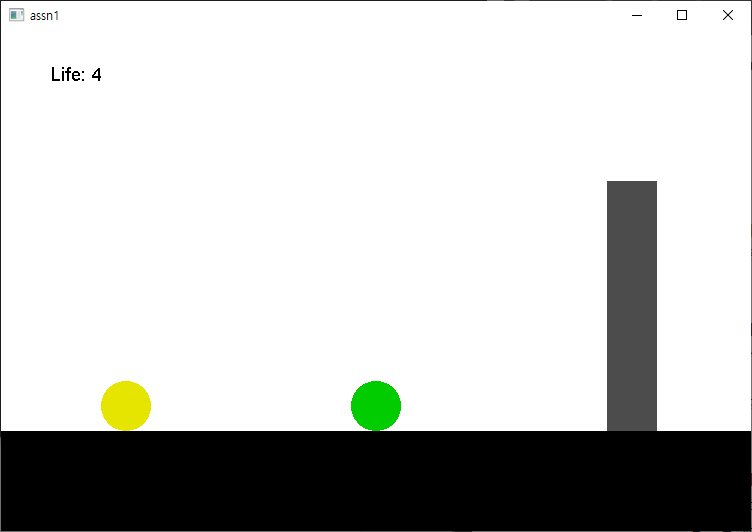
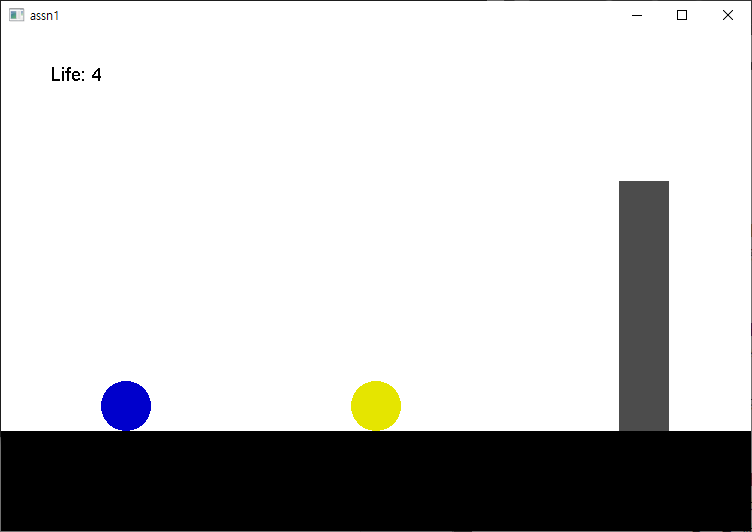
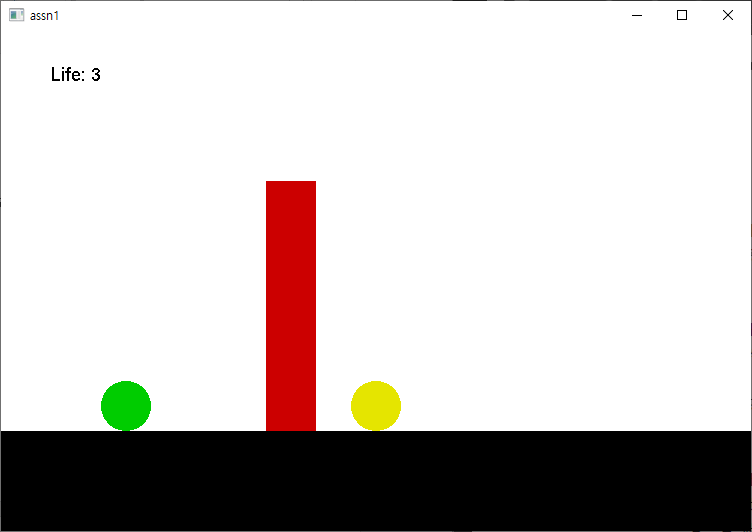
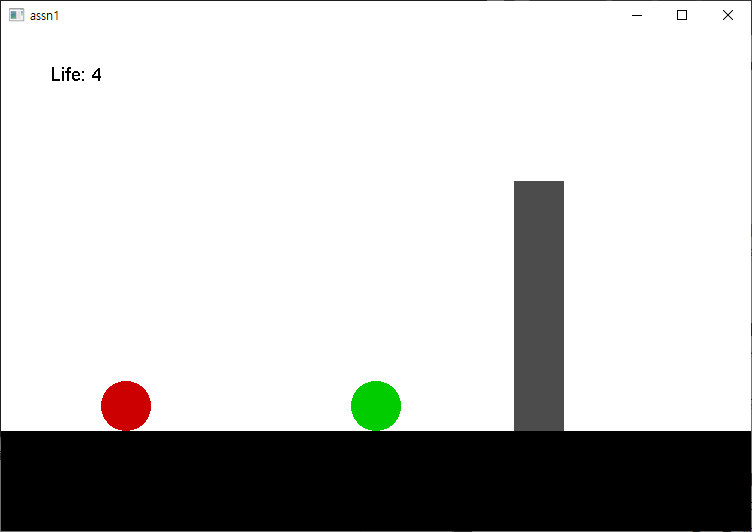
**How to Run**

이번 과제는 별도의 메뉴, data file이 존재하지 않아 특별한 프로그램 실행 방법은 없다.

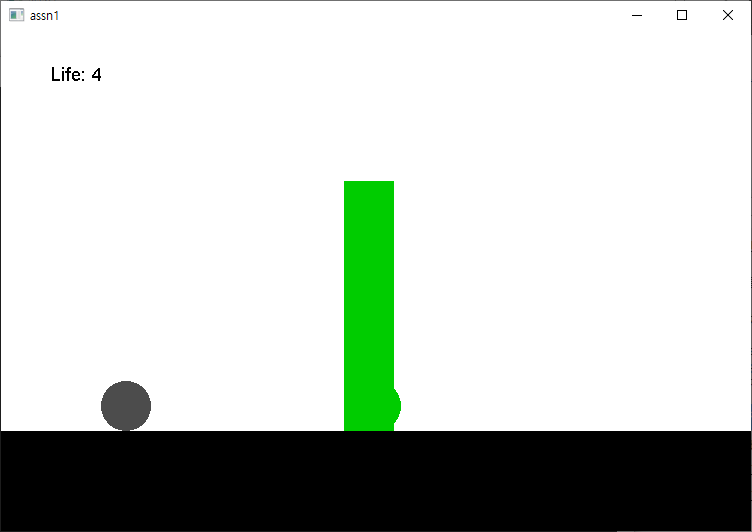
**Example**



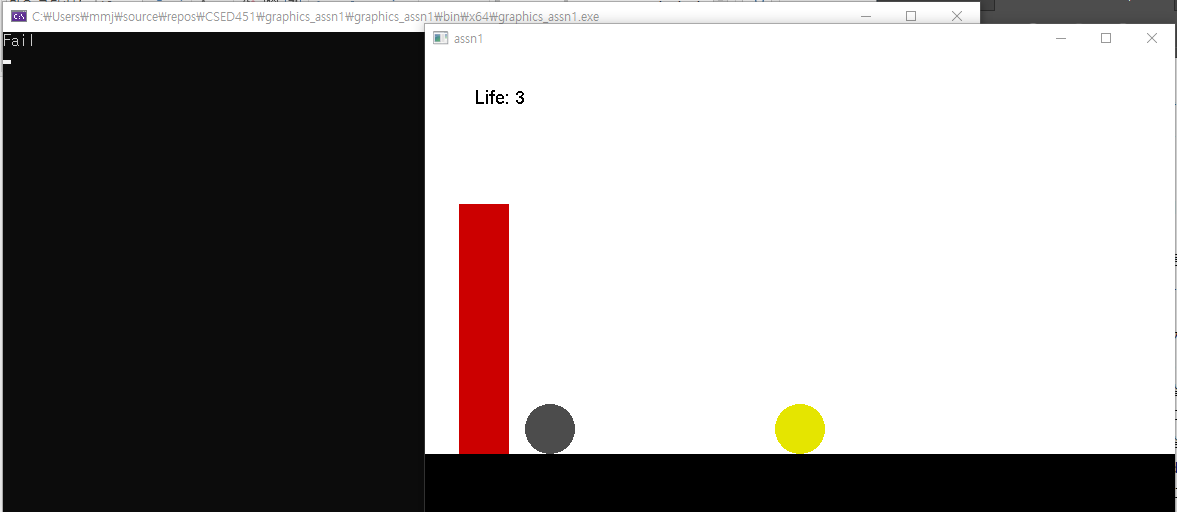
프로그램 실행 시 화면으로, 좌측 상단에 현재 life가 표시되고, 플레이어는 화면 왼쪽, 도둑은 화면 중앙에 위치한다. 벽은 화면 오른쪽에서부터 생성되어 왼쪽으로 이동한다.



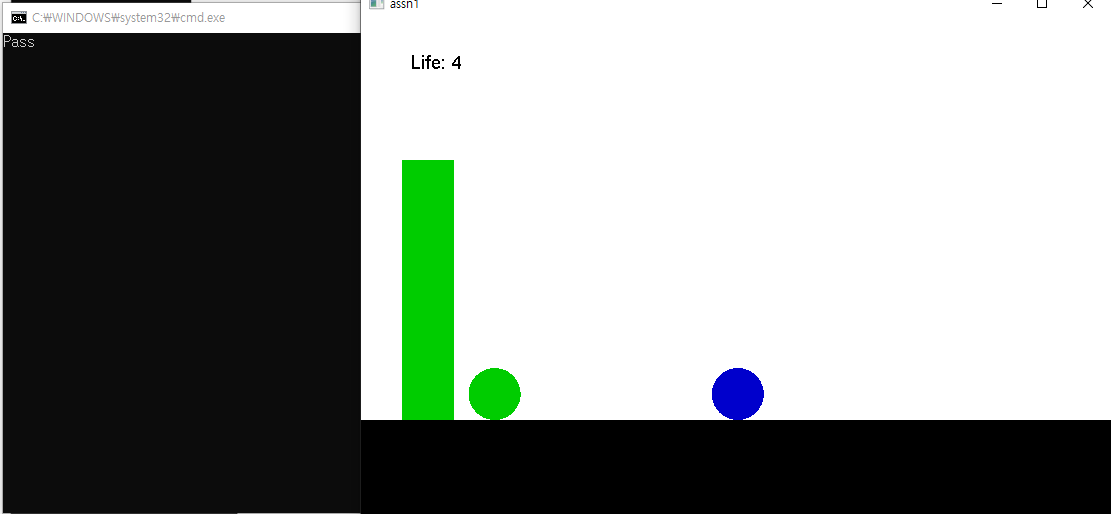
위(적), 아래(녹), 왼쪽(청), 오른쪽(황) 방향키를 눌러 플레이어의 색을 변경할 수 있다.



도둑의 색은 일정시간마다 랜덤하게 변경되며, 도둑 색의 변경 간격은 플레이어가 벽을 통과하여 도둑과 가까워질수록 줄어든다.  
도둑과 벽이 충돌할 경우 벽의 색이 도둑의 색으로 변경되어 플레이어에게 다가온다.

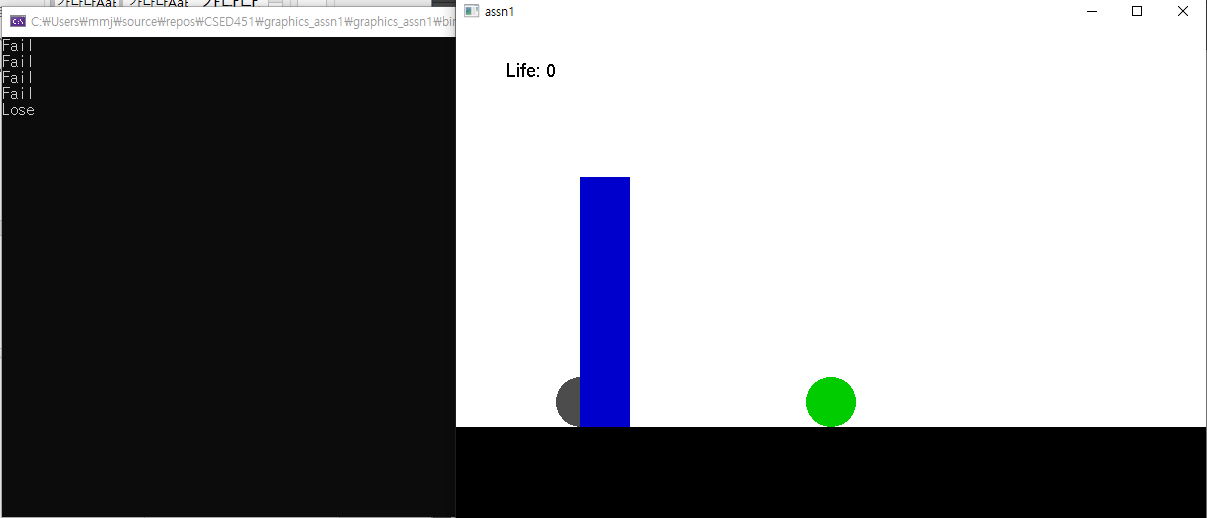


벽이 플레이어를 통과할 때 벽의 색과 플레이어의 색이 다를 경우, Life가 1 줄어들고, 콘솔에는 Fail이 출력된다.

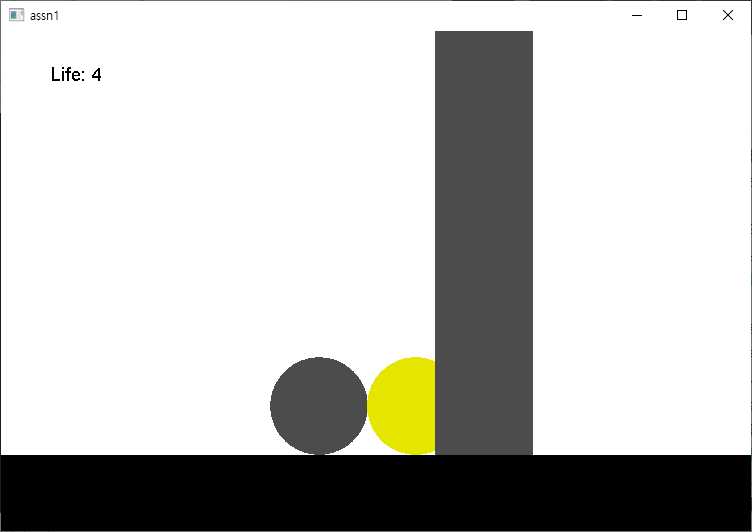


벽이 플레이어를 통과할 때 벽의 색과 플레이어의 색이 같을 경우, 콘솔에 ‘Pass’가 출력되고, 플레이어와 도둑의 거리가 가까워짐과 동시에 화면이 줌인되어 더 좁은 영역을 그린다.

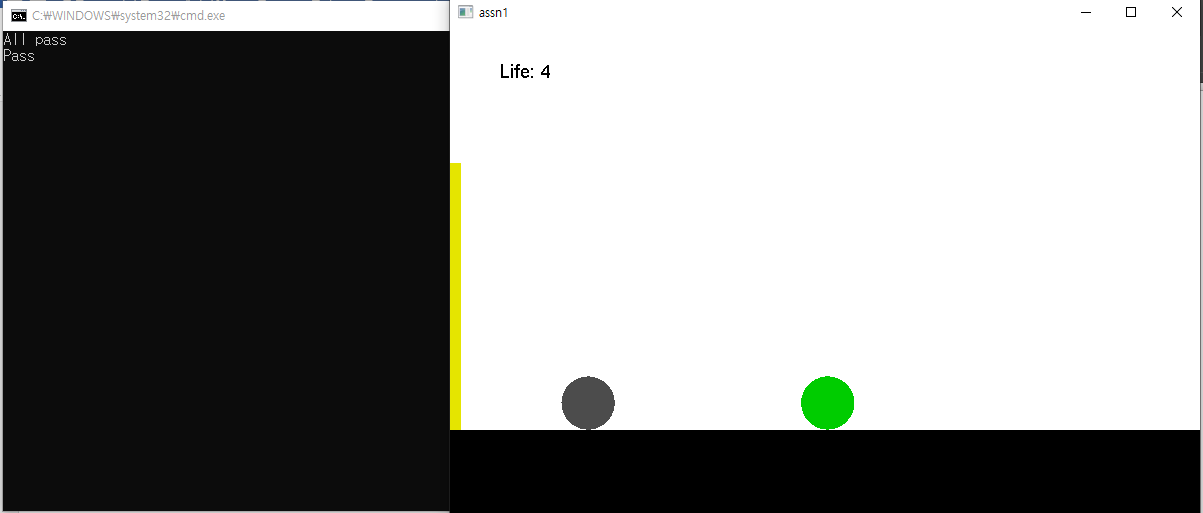
그리고 벽의 속도와 재생성 간격이 빨라진다.



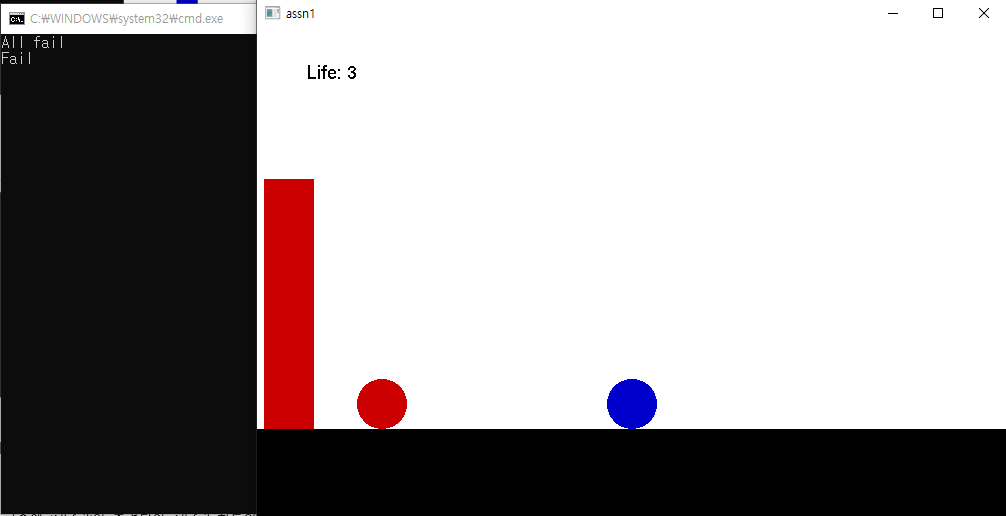
Life를 모두 잃어 0이 되면, 콘솔에는 Lose가 출력되고, 화면이 멈추어 추격이 종료된다.



벽을 계속 성공적으로 통과하여 도둑과 가까워지다가, 플레이어와 도둑이 서로 맞닿게 되면 콘솔에 ‘Win’이 출력되어 플레이어가 승리했음을 알리고, 추격이 종료된다.



C를 누르면 콘솔에 ‘All pass’가 출력되어 All pass 치트가 활성화되었음을 알리고, 플레이어의 색과 무관하게 벽을 성공적으로 통과한 것으로 간주한다.



F를 누르면 콘솔에 ‘All fail’이 출력되어 All fail 치트의 활성화를 알리고, 플레이어의 색과 무관하게 벽을 통과하지 못한 것으로 간주한다.

**Discussion**

1. 자료구조 설계 및 기본 구동 구현
2. 벽의 이동

과제의 지시사항에 따르면 벽은 화면의 오른쪽에서 반복적으로 생성되어 화면의 왼쪽으로 이동하며, 벽의 생성 패턴은 자유인 대신 플레이어가 벽을 통과할 때마다 벽의 생성 간격은 짧아지고 이동 속도는 빨라져야 한다.

이러한 벽을 구현하기 위한 방법으로 두 가지 방안을 고려하였는데, 첫째는 world 상에 벽들을 필요한 최대 개수만큼 미리 생성하여 늘어놓고 플레이어와 도둑, 그리고 시점을 이동시키는 방법이었고, 둘째는 플레이어와 도둑, 시점을 고정시킨 채 필요할 때마다 벽을 생성시키고 이동시키는 방식이었다.  
처음에는 첫 번째 방식으로 구현하고자 했으나, 플레이어가 벽의 통과에 성공했을 때에만 벽의 재생성 간격이 짧아지도록 구현하는 것에 어려움을 느껴 두 번째 방식으로 벽의 이동을 구현하기로 결정하였다.

벽의 이동의 구현 방식을 캐릭터와 시점을 고정시키고 벽의 위치만 이동시키기로 결정한 후, 다음으로 고려한 사항은 벽의 생성이었다. 첫 번째로 고려한 방식은 벽이 게임이 종료되기 전까지 화면 밖에서 무한히 생성되고 화면을 벗어나면 파괴시키는 방식이었다. 하지만 이 방식은 벽의 무한한 생성과 파괴의 구현이 상대적으로 어려웠고, 생성된 벽을 파괴시키지 않을 경우 화면에는 나타나지 않지만 화면 밖에 생성된 채로 방치되어 있는 수많은 벽들이 자원의 낭비를 일으킬 것이기에 다른 방안을 고민해보았다.  
그 결과 world 상에 단 하나의 벽만이 존재하는 대신에 벽이 화면의 왼쪽 밖을 벗어나면 다시 오른쪽 끝으로 옮겨주는 방식을 고안해냈고, 이 방식을 활용하면 컴퓨터의 자원을 효율적으로 사용할 수 있음과 더불어 벽이 왼쪽으로 사라지는 즉시 다시 화면상의 오른쪽으로 옮기기 때문에 플레이어가 벽의 통과에 성공함에 따라 벽의 속도를 증가시키면 자연스럽게 벽의 재생성 간격이 늘어난 것처럼 보이게 된다는 구현상의 이점을 찾을 수 있었다. 따라서 world 상에 단 하나의 벽만이 존재하는 대신에 벽이 화면의 왼쪽 밖을 벗어나면 다시 오른쪽 끝으로 옮겨주는 방식으로 벽의 생성을 구현하였다.

1. 충돌 판정

플레이어의 승리 시점은 플레이어 캐릭터와 도둑 캐릭터가 서로 닿는 순간으로 결정하였고, 캐릭터와 벽의 충돌 시점은 벽의 왼쪽 모서리가 캐릭터의 중심을 지나는 순간으로 결정하였다.

캐릭터(플레이어, 도둑)와 벽의 충돌 판정을 구현하는 과정에서, 벽이 캐릭터와 충돌했을 때 충돌 판정이 한 번만 발생하는 것이 아닌, 캐릭터가 벽을 통과하는 동안 계속해서 발생하는 문제가 있었다. 충돌 판정이 계속해서 발생했기에, 도둑이 벽을 통과하는 동안 도둑의 포즈(색)가 변경되면 벽의 포즈(색)도 변경되고, 플레이어가 벽을 통과하는 도중 포즈를 변경하면 성공/실패가 뒤집히는 현상이 발생하였다.

이러한 문제를 해결하기 위하여 캐릭터 클래스 내에 isCollided라는 변수를 추가하여 isCollided가 true일 경우 충돌 판정을 발생시키지 않아 단 한 번의 충돌 판정만 발생시키게끔 구현하여 이러한 문제를 해결하였다.

1. Pass/Fail Action

**Conclusion**

1. 자료구조 설계 및 기본 구동 구현

이번 assn1을 수행하면서 기본적인 openGl 사용법을 익힐 수 있었다. 또한, LMS에 올라온 질의응답과 기본적인 기능의 구동을 구현을 통해서 실제로 무한한 world 또는 무한 개수의 오브젝트를 전부 구현하는 것보다는 화면에 나타나는 영역이 한정된 사실을 이용하여 화면에 나타나는 부분들만 구현하거나 화면 밖으로 벗어난 리소스를 적절히 재사용하는 것이 컴퓨터 자원 관리와 실제 프로그래밍의 편리함 부분에서 이점이 있음을 알게 되었다.

1. Pass/Fail Action

**Direction of Improvement**