CSED451 Assignment #1

2D DRAWING REPORT

Team: GPS

CSE 20202405 문민재 csmmj4594

CSE 20202728 김진수 fusion4268

**Overview of Program**

본 프로그램은 플레이어가 도둑을 추격하며 게임이 진행된다. 플레이어와 도둑은 우측으로 추격전을 이어가고 벽과 충돌하게 된다. 도둑이 벽과 충돌할 때 벽은 도둑의 색과 같은 색으로 변한다. 도둑의 색은 일정한 시간에 따라 무작위로 변하며, 사용되는 색은 빨간색, 녹색, 파란색 그리고 노란색으로 총 4가지 색이다.

도둑이 벽을 통과한 뒤 플레이어가 벽과 충돌하게 되며, 충돌할 때 플레이어는 벽과 같은 색을 띠어야 한다. 플레이어는 방향키를 사용하여 색을 변화시킬 수 있다. 같은 색일 때, 콘솔 창에 Pass를 출력하고 도둑과 플레이어 사이의 거리가 감소한다. 벽 또한 생성 간격이 짧아지고 이동 속도가 증가하게 되며, 그에 따라 카메라도 줌인된다. 플레이어와 벽의 색이 다를 때, 콘솔 창에 Fail을 출력하고 life가 1만큼 감소하게 된다. Life는 총 4개이며, 0이 될 때 콘솔 창에 Lose를 추격하고 게임을 종료한다. Life는 화면 왼쪽 상단에 표시된다. 플레이어와 도둑 사이의 거리가 감소해 서로 충돌하였을 때, 콘솔 창에 Win을 출력하고 게임을 종료한다.

플레이어의 입력과 무관하게 모든 벽을 통과했다고 간주하는 올 패스 기능과, 모든 벽을 통과하지 못했다고 간주하는 올 페일 기능을 구현하였다. “c”를 눌러 올 패스 기능을 키고 끌 수 있으며, “f”를 눌러 올 페일 기능을 키고 끌 수 있다.

**Programming Environment**

Visual Studio 2019

OpenGL – 2.1

Freeglut-MSVC-3.0.0

Glew-2.1.0

Glm-0.9.9.7

**Design and Implementation**

1. 자료구조 설계 (source.h)
2. Coordinates

플레이어가 벽을 통과함에 따라, 카메라를 줌인하기 위해 카메라의 영역을 정의하는 클래스이다. 멤버 변수로는 카메라 상하좌우의 좌표 값이 있다.

1. 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. 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. 플레이어와 도둑이 충돌 했을 때 콘솔 창에 Win을 출력하고 게임을 종료한다.

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
2. Pass Action

collisionCheck에서 wall과 player가 충돌할 때, allPass의 값이 1이거나 wall의 색과 player의 색이 같을 때 pass로 간주한다. 콘솔 창에 pass를 출력하고, wallSpeed를 wallSpeed Increment만큼 증가시킨다. 75로 초기화한 도둑의 색 변화 주기 colorPeriod를 5만큼 감소시킨다. 50프레임 동안 플레이어의 위치가 movingDistance만큼 오른쪽으로 이동하게 하기 위하여 playerNewX에 moving Distance를 더해준다. 이후 매 프레임마다 player의 x좌표가 playerNewX 보다 작은 지 확인하며, 작을 경우 player의 x 좌표를 player.moveright 함수로 distancePerFrame만큼 이동시킨다. 카메라 또한 zoomFrame 동안 coordinatesIncrement만큼 줌인하기 위하여 newWorld을 world 값에 coordinatesIncrement 값을 더한 값으로 한다. 매 프레임마다 world의 left 값이 newWorld의 left 값보다 작은지 확인하며, 작을 경우 world의 상하좌우 좌표에 incrementPerFrame 좌표를 더해준다. 이후, gluOrtho2D에 새로운 world 값을 넣어 바뀐 world 좌표로 줌인되게 된다.

카메라가 줌인되더라도 좌측 상단의 lifeText의 위치는 고정되어 있어야 한다. lifeText의 좌표는 다음 식을 따르며, frameMoved는 처음 화면으로부터 몇 프레임만큼 줌인되었는지 확인한다.

lifeX = frameMoved \* incrementPerFrame.getLeft() + LIFE\_X \*

(world.getRight() - world.getLeft()) / WORLD\_X;

lifeY = frameMoved \* incrementPerFrame.getBottom() + LIFE\_Y \*

(world.getTop() - world.getBottom()) / WORLD\_Y;

LIFE\_X, LIFE\_Y는 처음 화면의 life의 좌표이며, WORLD\_X, WORLD\_Y는 처음 화면의 world의 너비와 높이이다.

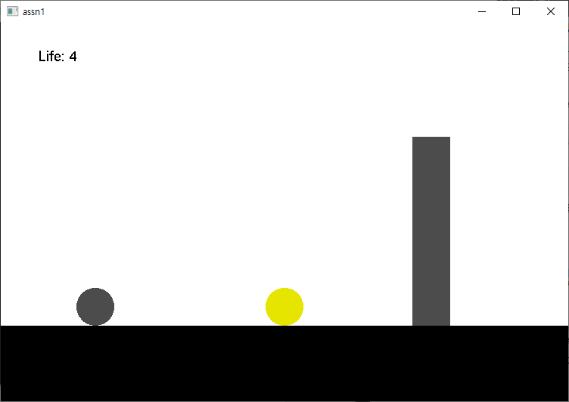
1. Fail Action

collisionCheck에서 wall과 player가 충돌할 때, allFail의 값이 1이거나 wall의 색과 player의 색이 다를 때 fail로 간주한다. 콘솔 창에 Fail을 출력하고 life를 1만큼 감소시킨다. life가 0이 되면 콘솔 창에 Lose를 출력하고 게임을 종료한다.

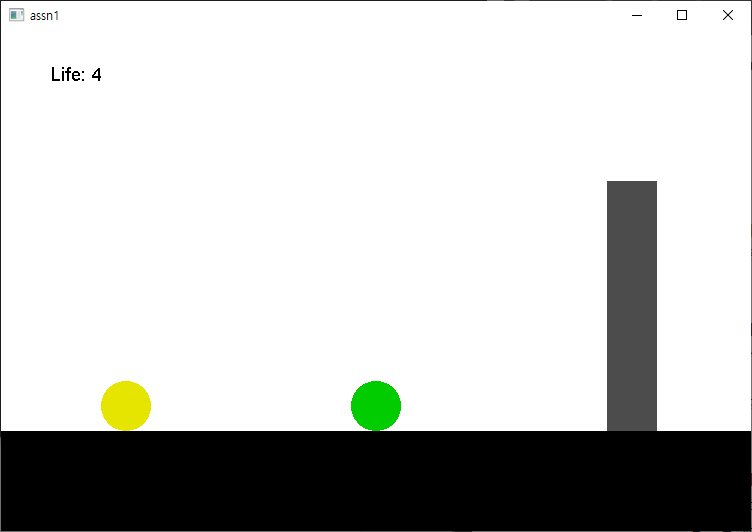
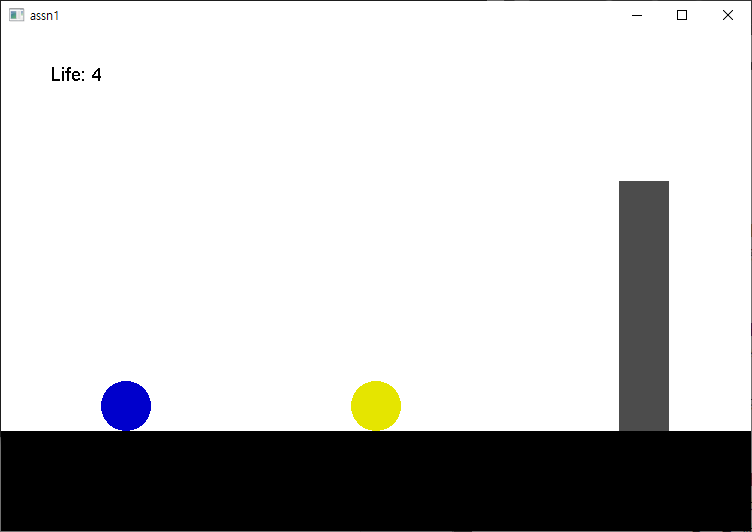
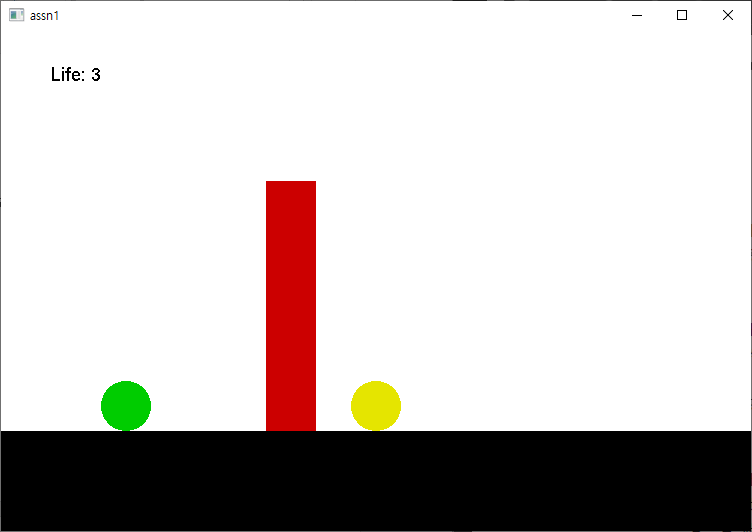
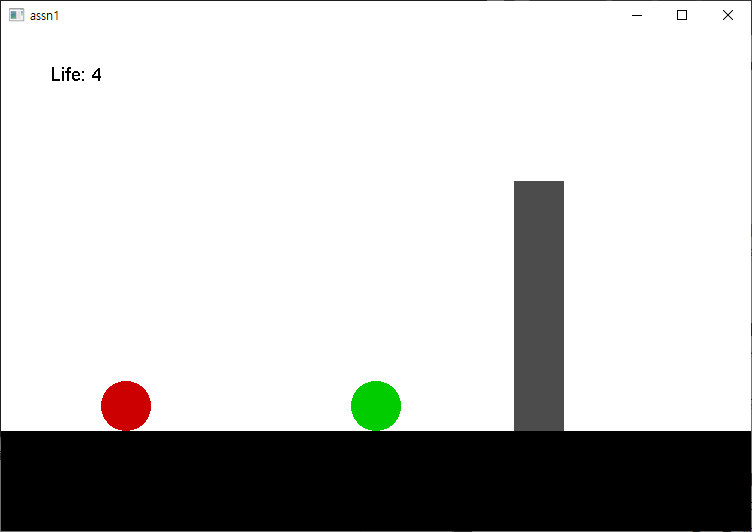
**How to Run**

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

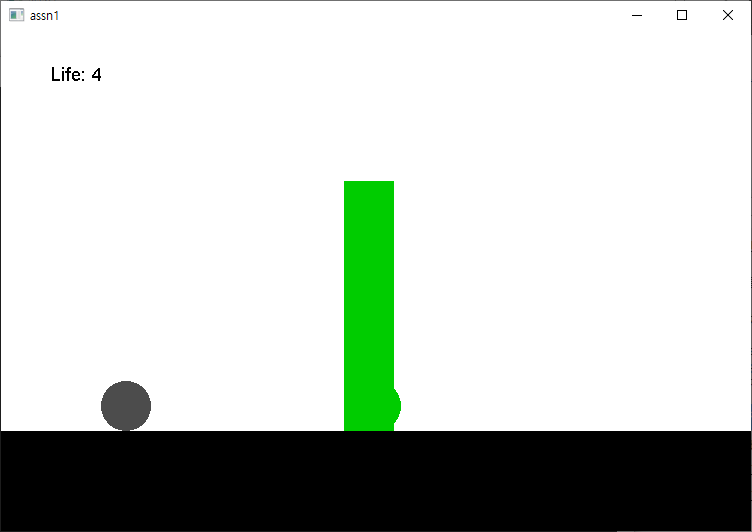
**Example**



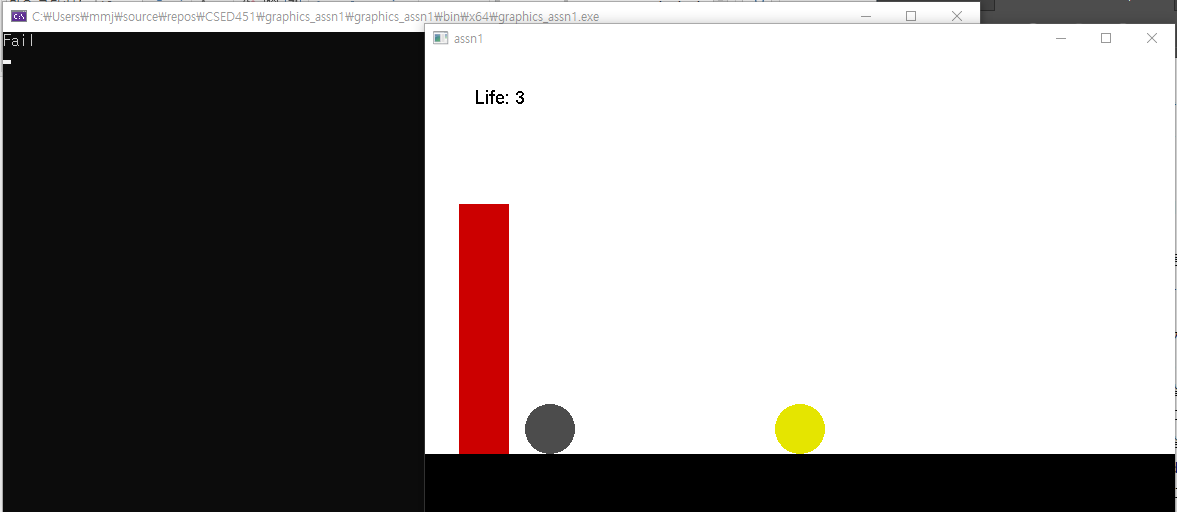
프로그램 실행 시 화면으로, 좌측 상단에 현재 life가 표시되고, 플레이어는 화면 왼쪽, 도둑은 화면 중앙에 위치한다. 벽은 화면 오른쪽에서부터 생성되어 왼쪽으로 이동한다.



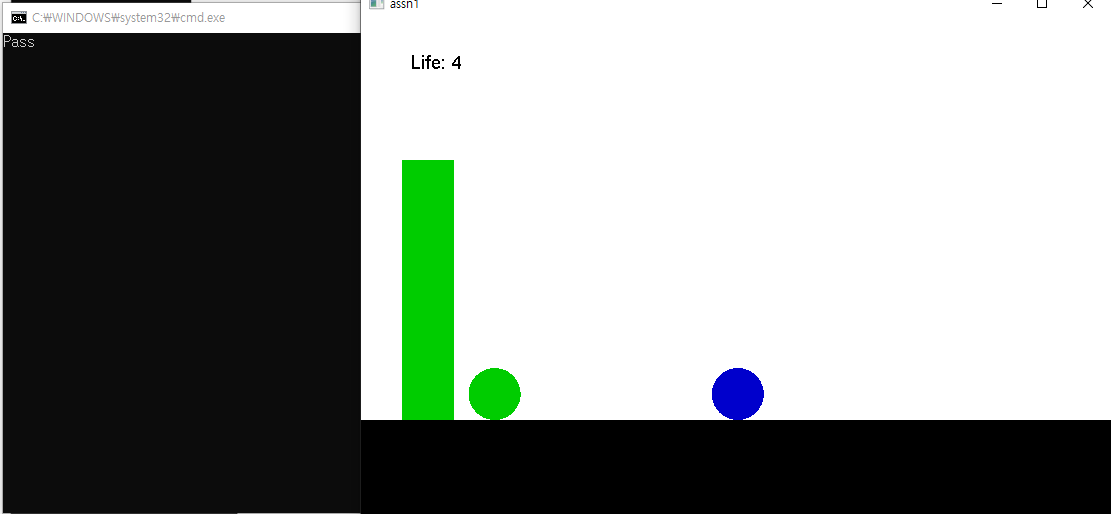
위(적), 아래(녹), 왼쪽(청), 오른쪽(황) 방향키를 눌러 플레이어의 색을 변경할 수 있다.



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.

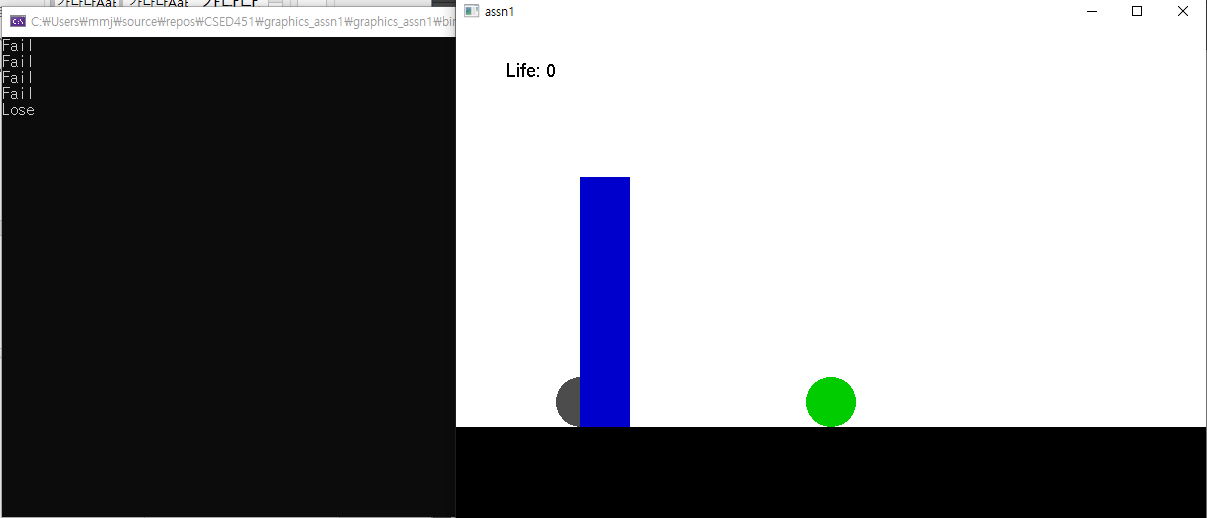


벽이 플레이어를 통과할 때 벽의 색과 플레이어의 색이 다를 경우, Life가 1 줄어들고, 콘솔에는 Fail이 출력된다.

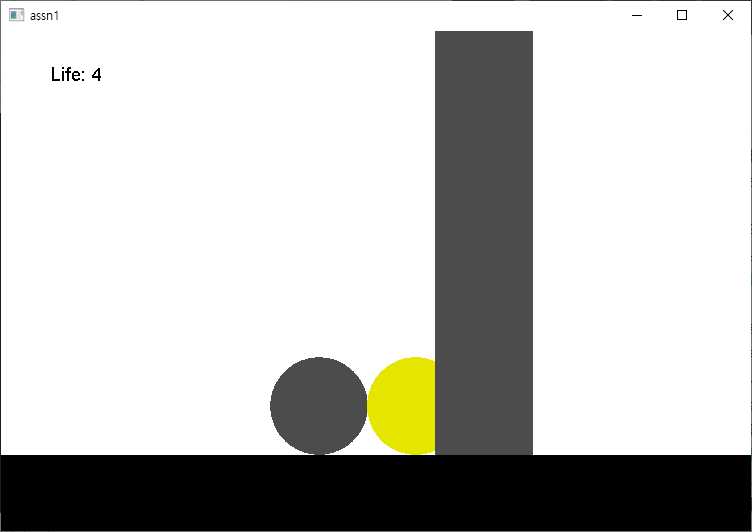


벽이 플레이어를 통과할 때 벽의 색과 플레이어의 색이 같을 경우, 콘솔에 ‘Pass’가 출력되고, 플레이어와 도둑의 거리가 가까워짐과 동시에 화면이 줌인되어 더 좁은 영역을 그린다.

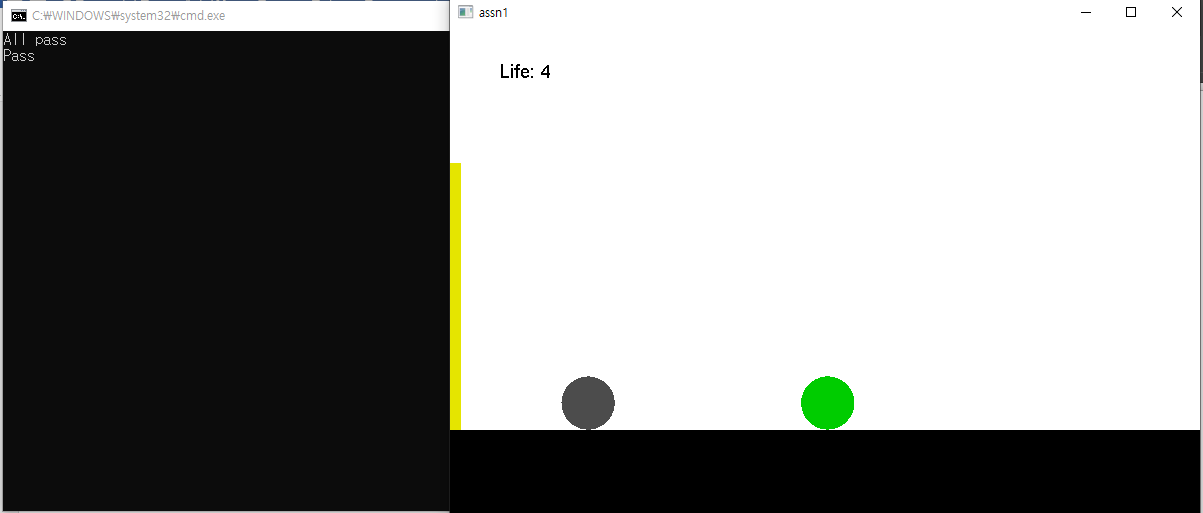
그리고 벽의 속도와 재생성 간격이 빨라진다.



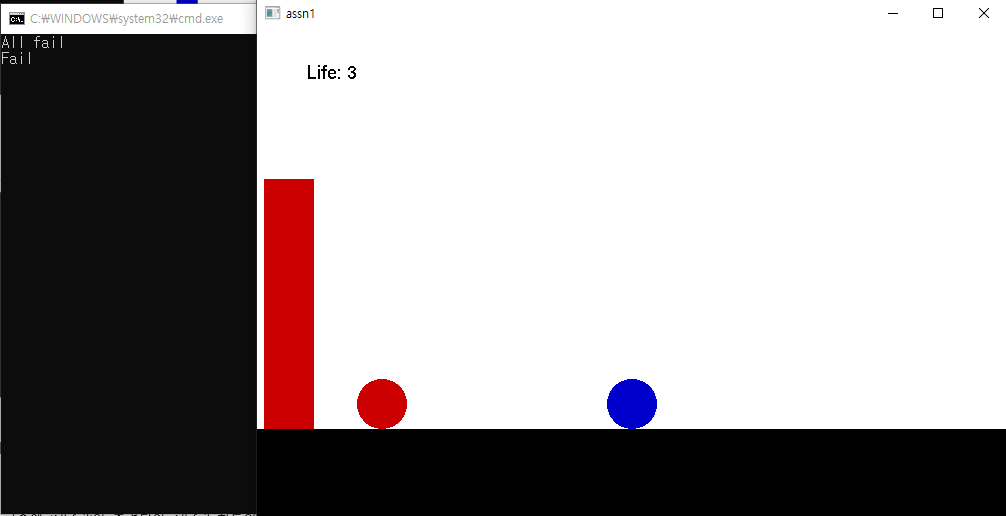
Life를 모두 잃어 0이 되면, 콘솔에는 Lose가 출력되고, 화면이 멈추어 추격이 종료된다.



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
2. Continuous Action

플레이어가 벽과 같은 색일 때 벽을 통과해 Pass 판정이 발생되었을 때, 플레이어의 위치를 오른쪽으로 움직이고 카메라를 줌인해야한다. 단순히 플레이어의 위치를 옮기고자하는 위치만큼 증가시키고, 카메라를 원하는 만큼 줌인 할 수 있지만 게임의 몰입감을 방해하고 화면이 끊긴 것처럼 보일 것으로 생각되었다. 플레이어가 실제로 오른쪽으로 움직이는 것처럼, 카메라가 실제로 줌인되고있는 것처럼 보여지기 위해 목표 값을 지정하고 특정 프레임 동안 목표 값만큼 좌표가 움직이도록 구현하였다. 그 결과 플레이어의 위치와 카메라의 연속적인 action을 구현할 수 있었다.

1. Camera Zoom In

카메라의 줌인은 glMatrixMode(GL\_PROJECTION)와 gluOrtho2D()를 이용하여 구현하였다. 윈도우에 보여지는 화면의 너비와 높이를 축소시킴으로써 대상을 확대하는 듯한 느낌을 줄 수 있었다. 고려해야할 사항으로는 초기 화면비와 축소시키는 화면비가 같아야 한다는 점이었다. 초기 화면비와 축소시키는 화면비가 다를 때, 화면의 대상이 왜곡되는 현상이 발생하였다.

1. LifeText Position

LifeText의 위치는 본 프로그램을 구현하는 데 가장 많은 시간을 소요한 부분이었다. 카메라가 줌인 될 때, 텍스트는 전체 윈도우에서 같은 자리에 위치하고 있어야하며 이는 텍스트의 position이 이동하면서 화면 비율만큼 x와 y좌표를 축소시켜야 함을 의미했다. 그림을 그려 관계식을 계산해 텍스트가 카메라가 줌인 될 때 마다 가져야할 좌표값을 얻어낼 수 있었고, 대상은 확대하지만 텍스트의 위치는 고정되어 있는 화면을 구현할 수 있었다.

1. 추격 종료

Win이나 Lose가 발생해 게임이 종료되고 추격을 멈출 때, 어떤 방법으로 게임을 끝낼 수 있을지 고민하였다. 단순히 벽의 위치가 움직이지 않도록 고정시킬 수 있었으나, 프로그램 내에서는 매 프레임 마다 여러 함수들이 여전히 실행되므로 비효율적이라고 생각하였다. 이를 해결하기위해 finishGame 함수에 프레임마다 반복적으로 실행되고 있는 함수들에 NULL을 전달함으로써 불필요한 계산을 수행하지 않도록 하였다.

**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

Pass/Fail Action을 구현하면서 가장 많이 배울 수 있던 부분은 glMatrixMode와 관련된 부분이었다. GL\_MODELVIEW에 어떤 값을 전달하는지에 따라서 프로그램을 보여주는 방식을 선택할 수 있었으며, gluOrtho2D를 이용하여 연속적으로 카메라를 줌인하는 부분이 이를 배우는 데 많은 도움을 주었다. lifeText의 위치를 조정하면서 카메라를 줌인할 때 고정하고자 하는 대상에 대해서는 화면의 줌인과 화면비 변화 또한 고려해야한다는 것을 알 수 있었다.

**Direction of Improvement**

현재 구현된 프로그램에서는 흰색 바탕으로 배경이 구현되고, 플레이어와 도둑의 움직임에 따른 특별한 동작이 존재하지 않아 움직인다는 느낌을 주는 것이 어렵다고 보여진다. 게임의 진행방식을 이해하지 못한 사용자가 프로그램을 사용하게 된다면 도둑과 플레이어가 오른쪽으로 움직이는 것이 아닌 벽이 왼쪽으로 이동하는 것으로 생각할 수도 있다는 생각이 들었다. 이러한 문제점은 다음의 두가지 방식으로 해결할 수 있을 것이다.

1. 배경 구체화

배경을 흰색의 단순한 배경이 아닌 나무나 구름과 같은 물체를 포함시켜준다면 플레이어와 도둑이 오른쪽으로 움직이는 듯하게 보여주는 데 도움이 될 것으로 생각한다.

1. 움직이는 모션 구현

현재의 플레이어와 도둑은 단색의 원으로 구현되어 있어 움직이는 듯한 느낌을 제공하기 어렵다. 실제 사람이나 동물의 이미지를 활용하여 달리는 것처럼 구현하여 움직이는 듯한 느낌을 구현할 수 있다. 비교적 단순한 방법으로는 단색의 원이 아닌 여러 색이 혼합된 원을 사용하여 원의 중점을 기준으로 이동 속도에 따라 회전시켜준다면 원이 굴러가는 듯한 시각적 효과를 제공할 수 있을 것으로 보여진다.