CSE3013 최종 프로젝트

20200901 이효주

1. 프로젝트 목표

본 최종프로젝트는 Waterfall 실습을 참고하여 구현하였다.

6~7주차 실습에서는 선반이 벽면에 임의로 놓여있을 때, 물이 떨어져 흐르는 경로를 구현하였으며, 선분과 점을 나타내기 위한 LineSegment와 Dot 구조체를 정의하여, 물을 직선으로 표현한 후 시작점, 선분과 맞닿는 점, 선분 끝 점 등 점들의 좌표를 저장하여 연결하였다.

원 코드는 일련의 섬과 점을 통해 waterfall 시각화를 했다면, 본 프로젝트에서는 사용자가 점을 제어하여 장애물을 피하고 목표에 도달할 수 있는 인터랙티브한 장애물 회피 게임으로 구현하고자 목표를 설정하였다.

따라서, 사용자가 키보드를 이용해 움직이는 점을 조작하여 화면상의 아이템을 수집하고, 선(장애물)에 충돌하지 않고 목적지에 도달해야 한다.

게임에서는 선으로 표현된 장애물을 피해서 목표 지점에 도착하면 게임이 클리어되고, 장애물에 부딪히거나 시간이 지남에 따라 감소하는 점수가 0점이 되면 'Game Over' 메시지가 나타난 후 2초 후에 게임이 종료된다. 이 과정에서 점수 증가, 점수 감소, 속도 증가 등의 효과를 가지는 랜덤으로 생성되는 아이템이 일정한 속도로 움직이며, 일정 시간 후 사라지도록 구현하였다.

2. 실험 환경

운영체제: Windows 10
개발 도구: Visual Studio 2022, openFrameworks 라이브러리
프로그래밍 언어: C++

3. 각 변수에 대한 설명

1. float dot_x;
움직이는 점의 x좌표를 저장하는 변수이다. 사용자가 방향키를 눌러 점을 이동할 때 이 변수의 값은 변경된다.

2. float dot_y;
움직이는 점의 y좌표를 저장하는 변수이다. ‘dot_x’와 마찬가지로 사용자가 방향키를 눌러 점을 이동할 때 이 변수의 값은 변경된다.

3. float** line_in;
선들의 좌표를 저장하는 2차원 배열이다. 각 선의 시작점과 끝점의 x, y 좌표를 저장하여 선의 위치와 방향을 나타낸다.

4. clock_t start, over, collision_time;
게임 시작 시간, 종료 시간, 충돌 시간을 저장하는 변수이다. 게임 진행 시간과 충돌 발생 시간을 측정하는 데 사용하였다.

5. double times;
경과 시간을 저장하는 변수이다. 게임 시작부터 종료까지의 시간을 기록하여 게임 플레이 시간을 측정한다.

6. bool* collision_flags;
각 선에 대한 충돌 여부를 저장하는 배열이다. 특정 선과 충돌이 발생하면 해당 배열의 값이 true로 설정된다.

7. bool collision_detected = false;
충돌이 감지되었는지 여부를 나타내는 변수이다. 충돌이 발생하면 true로 설정되어 충돌 후의 동작을 제어한다.

8. int score;
현재 점수를 저장하는 변수이다. 점수는 시간이 지남에 따라 감소하며, 아이템을 수집하거나 충돌할 때 변경된다.

9. float playerSpeed;
플레이어의 이동 속도를 저장하는 변수이다. 아이템에 따라 속도가 변경될 수 있다.

10. std::vector<Item> items;
아이템들을 저장하는 벡터이다. 각 아이템의 위치, 속도, 유형, 생성 시간 및 생명 시간을 포함하여 게임 내의 아이템들을 관리한다.

4. 각 함수에 대한 설명

1. setup()

초기 설정 함수이며, 프레임 레이트, 배경색, 선 굵기, 점의 초기 좌표 등을 설정하고, 점수를 초기화한다.

2. update()

매 프레임마다 호출되어 게임 상태를 업데이트한다. 충돌 및 점수 변화를 처리한다.

충돌이 감지된 경우, 충돌 후 2초가 경과하면 “Game Over!” 메시지를 출력하고 프로그램을 종료한다.

충돌이 감지되지 않은 경우, 점수를 1씩 감소시키며, 점수가 0이되면 2초가 경과한 후 ““Game Over!” 메시지를 출력하고 프로그램을 종료한다.

아이템의 위치와 상태를 업데이트하고, 점이 아이템을 수집하면 해당 아이템을 제거하고 효과를 적용하도록 한다.

3. draw()

화면에 게임 요소들을 그리고, 점, 선, 아이템 등을 그리며, 게임 오버 시 메시지를 표시한다.

상단과 하단에 장식 요소를 추가했으며, draw_flag가 설정된 경우 각 선을 그린다. 충돌한 선은 빨간색으로 표시하고, 그렇지 않은 선은 검정색으로 표시한다.

움직이는 점을 파스텔 핑크색으로 그리고, 목적지 점을 노란색 점으로 효과를 넣어 그렸다.

현재 점수와 경과 시간을 화면에 검정색으로 표시하고, 아이템을 그렸으며, 충돌

이 감지되었거나 점수가 0이 된 경우 화면 중앙에 빨간색 "GAME OVER" 메시지를 표시한다.

4. keyPressed(int key)

키 입력을 처리하는 함수로, 사용자가 방향키나 'q', 'd' 키를 눌렀을 때의 동작을 정의한다.

'q' 키를 누르면 게임을 종료하며, 이때 동적으로 할당된 메모리를 해제한다.
'd' 키를 누르면 draw_flag를 1로 설정하고 generateItems()를 호출하여 아이템을 생성한다.

방향키 입력에 따라 점의 위치를 변경하고, 점의 이동 경로가 유효한지 check_road()를 호출하여 확인한다. 점의 새로운 좌표를 출력하고, gameover()를 호출하여 게임 종료 조건을 확인한다.

5. keyReleased(int key)

키가 놓였을 때의 동작을 정의하며, 'o' 키를 누르면 파일 다이얼로그를 열어 선 좌표 데이터를 로드한다. 사용자가 선택한 파일의 유효성을 검사하고, 유효한 경우 processOpenFileSelection 함수를 호출하며, load_flag를 1으로 설정한다.

6. processOpenFileSelection(ofFileDialogResult openFileResult)

파일을 열고 선 좌표 데이터를 읽어오는 함수이다.

파일 다이얼로그에서 선택한 파일이 존재하지 않으면 에러 메시지를 출력하고, 파일이 존재하면 파일 내용을 읽어와 선 좌표 데이터를 파싱하여 line_in 배열에 저장한다.

선의 개수를 확인하고, 잘못된 개수일 경우 에러 메시지를 출력하며, 각 선의 시작점과 끝점의 좌표를 읽어와 line_in 배열에 저장함과 동시에 충돌 여부를 저장할 collision_flags 배열을 초기화한다.

7. gameover()

게임 종료 조건을 확인하고, 종료 시 메시지를 출력하는 함수이다.

점이 목적지에 도달하면 "게임 클리어!!" 메시지를 출력하고, 게임 시작부터 종료까지의 시간을 기록하여 출력한다. 게임이 종료된 이후 프로그램을 종료한다.

8. check_road(int dot_x, int dot_y)

점의 이동 경로를 확인하여 충돌 여부를 판단하는 함수이다.

점의 새로운 위치가 선과 겹치는지 확인하고, 각 선에 대해 점의 위치와 선의 시작점, 끝점의 거리를 계산한다. 점이 선과 동일한 위치에 있을 경우 충돌로 판단하고, collision_flags와 collision_detected를 업데이트한다.

충돌이 발생하면 collision_time을 기록하고 -1을 반환하며, 충돌이 발생하지 않으면 0을 반환한다.

9. generateItems()

랜덤한 위치에 아이템을 생성하는 함수이다.

기존 아이템을 초기화하고, 5 개의 새로운 아이템을 생성한다. 각 아이템의 좌표를 화면 내의 랜덤한 위치로 설정하고, 각 아이템의 -0.5에서 0.5 사이의 속도와 유형 (0: 점수 2배, 1: 점수 절반 감소, 2: 속도 증가)을 랜덤하게 설정한다.

아이템의 생성 시간을 현재 시간으로 설정한 후, 아이템의 생명 시간을 10초에서 20초 사이의 랜덤 값으로 설정하여 랜덤한 시간에 아이템이 사라지도록 한다. 생성된 아이템을 items 벡터에 추가한다.

10. updateItems()

아이템의 위치와 생명 시간을 업데이트하는 함수이다.

각 아이템의 위치를 속도에 따라 업데이트한다. 아이템이 경계를 벗어나면 속도의 방향을 반대로 변경하여 반사 효과를 준다. 아이템의 생성 시간과 현재 시간을 비교하여 생명 시간이 다한 아이템을 제거한다.

11. applyItemEffect(int type)

아이템의 종류에 따라 점수나 속도를 변경하는 함수이다.

점수 2배 아이템(type 0)을 수집하면 현재 점수를 2배로 증가시킨다.
점수 절반 감소 아이템(type 1)을 수집하면 현재 점수를 절반으로 감소시킨다.
속도 증가 아이템(type 2)을 수집하면 플레이어의 이동 속도를 2배로 증가시키며, 속도 증가 효과는 3초 동안 지속되며, 이후 원 상태로 되돌아온다.

5. 자료구조 및 알고리즘

<float>
dot_x, dot_y: 움직이는 점의 좌표를 저장한다.
line_in: 선들의 좌표를 저장하는 2차원 배열로, 각 선은 [x1, y1, x2, y2] 형식으로 시작점(x1, y1)과 끝점(x2, y2)의 좌표를 저장한다.

<clock_t>
start, over, collision_time: 게임 시작 시간, 종료 시간, 충돌 시간을 저장한다.

<double>
times: 경과 시간을 저장한다.

<bool>
collision_flags: 각 선에 대한 충돌 여부를 저장하는 배열이다.
collision_detected: 충돌이 감지되었는지 여부를 저장하는 변수이다.

<int>
score: 현재 점수를 저장한다.

<float>
playerSpeed: 플레이어의 속도를 저장한다.

std::vector<Item>
items: 아이템들을 저장하는 벡터이며, 각 아이템은 구조체로, 위치(x, y), 속도(speedX, speedY), 유형(type), 생성 시간(spawnTime), 생명 시간(lifespan)을 포함한다.

1) 충돌 감지 알고리즘

위 프로젝트에서 충돌 감지 알고리즘을 작성하는 데 가장 많은 시간이 들었다. 플레이어가 점을 움직일 때, 선들과 충돌하는지 확인하는 알고리즘으로, 점과 선의 좌표를 비교하여, 점이 선과 일정 범위 내에 있는 경우 충돌로 간주하여 진행을 막는다.

점(dot)의 좌표를 (dot_x, dot_y)라고 하고, 선(line)의 양 끝점을 (x1, y1)와 (x2, y2)라고 하면, 점과 선 사이의 최소 거리를 계산하여 충돌 여부를 판단한다.

선분의 양 끝점 (x1, y1)와 (x2, y2)를 연결하는 직선의 방정식을 구한 후, 점 (dot_x, dot_y)에서 이 직선까지의 수직 거리를 계산하도록 한다.

d는 |A * x + B * y + C| / sqrt(A^2 + B^2)로 계산된다.

점 (dot_x, dot_y)에서 직선까지의 수직 거리 d를 계산한 후, 거리 d가 일정 임계값 이하이면 충돌로 간주한다. 선분의 양 끝점 (x1, y1)와 (x2, y2) 사이에 점 (dot_x, dot_y)이 위치하는지 추가로 확인하기도 한다.

2) 아이템 이동
updateItems 함수에서 각 아이템의 위치를 속도에 따라 업데이트한다. 아이템이 화면 경계를 벗어나면 속도의 방향을 반대로 변경하여 반사 효과를 준다,

3) 아이템 생명 시간 관리
아이템 생성 시 spawnTime과 lifespan을 설정한다. updateItems 함수에서 현재 시간과 spawnTime을 비교하여 생명 시간이 다한 아이템을 제거한다.

4) 아이템 효과 적용:
applyItemEffect 함수에서 아이템의 유형에 따라 점수나 속도를 변경한다. 점수 2배, 점수 절반 감소, 속도 증가 효과를 적용하며, 속도 증가 효과는 3초 동안 지속되며, 이후 원래 속도로 복구된다.

5. 시간 및 공간 복잡도

[시간 복잡도]

1) 게임 초기화 (setup)
시간 복잡도: O(1)
설정 작업이 고정된 시간 내에 완료되므로 상수 시간이다. 프레임 레이트, 배경색, 선 굵기, 점의 초기 좌표 설정 등은 모두 상수 시간 내에 완료된다.

2) update()
점수 감소 및 충돌 확인: O(n)
각 프레임마다 모든 선의 좌표와 점의 좌표를 비교하여 충돌 여부를 확인한다. 선의 개수(n)에 비례하여 시간이 소요된다.
아이템 업데이트: O(m)
각 프레임마다 모든 아이템의 위치를 업데이트하고, 생명 시간을 체크하여 제거한다. 아이템의 개수(m)에 비례하여 시간이 소요된다.
아이템 수집 및 효과 적용: O(m)
각 프레임마다 모든 아이템이 점과 충돌하는지 확인하고, 충돌 시 효과를 적용한다. 아이템의 개수(m)에 비례하여 시간이 소요된다.

3) draw()
선 그리기: O(n)
각 프레임마다 모든 선을 그린다. 선의 개수(n)에 비례하여 시간이 소요된다.
아이템 그리기: O(m)
각 프레임마다 모든 아이템을 그린다. 아이템의 개수(m)에 비례하여 시간이 소요된다.
점 그리기 및 기타 UI 요소 그리기: O(1)

4) 키 입력 처리 (keyPressed, keyReleased)
시간 복잡도: O(1)
키 입력 처리 작업은 상수 시간 내에 완료된다.

5) 파일 처리 (processOpenFileSelection)
시간 복잡도: O(n)
파일을 열고 선 좌표 데이터를 읽어오는 데 걸리는 시간은 선의 개수(n)에 비례한다.

6) 게임 종료 조건 확인 (gameover)
시간 복잡도: O(1)
게임 종료 조건을 확인하는 과정은 상수 시간 내에 수행된다.

[공간 복잡도]

1) 게임 초기화 (setup)
공간 복잡도: O(1)
설정 작업에서 사용되는 메모리는 고정되어 있다.

2) update()
점수 및 충돌 여부 확인: O(n)
선의 개수(n)에 비례하여 충돌 여부를 저장하는 배열(collision_flags)을 사용한다.
아이템 업데이트 및 수집 체크: O(m)
아이템의 개수(m)에 비례하여 아이템 정보를 저장하는 벡터(items)를 사용한다.

3) draw()
선 그리기 및 아이템 그리기: O(n + m)
선의 개수(n)와 아이템의 개수(m)에 비례하여 메모리를 사용한다.

4) 키 입력 처리 (keyPressed, keyReleased)
공간 복잡도: O(1)
키 입력 처리 과정에서 사용되는 메모리는 고정되어 있다.

5) 파일 처리 (processOpenFileSelection)
공간 복잡도: O(n)
파일에서 읽어온 선의 좌표 데이터를 저장하는 데 필요한 메모리는 선의 개수(n)에 비례한다.

6) 게임 종료 조건 확인 (gameover)
공간 복잡도: O(1)
게임 종료 조건을 확인하는 과정에서 사용되는 메모리는 고정되어 있다.

[시간/공간 복잡도의 종합]

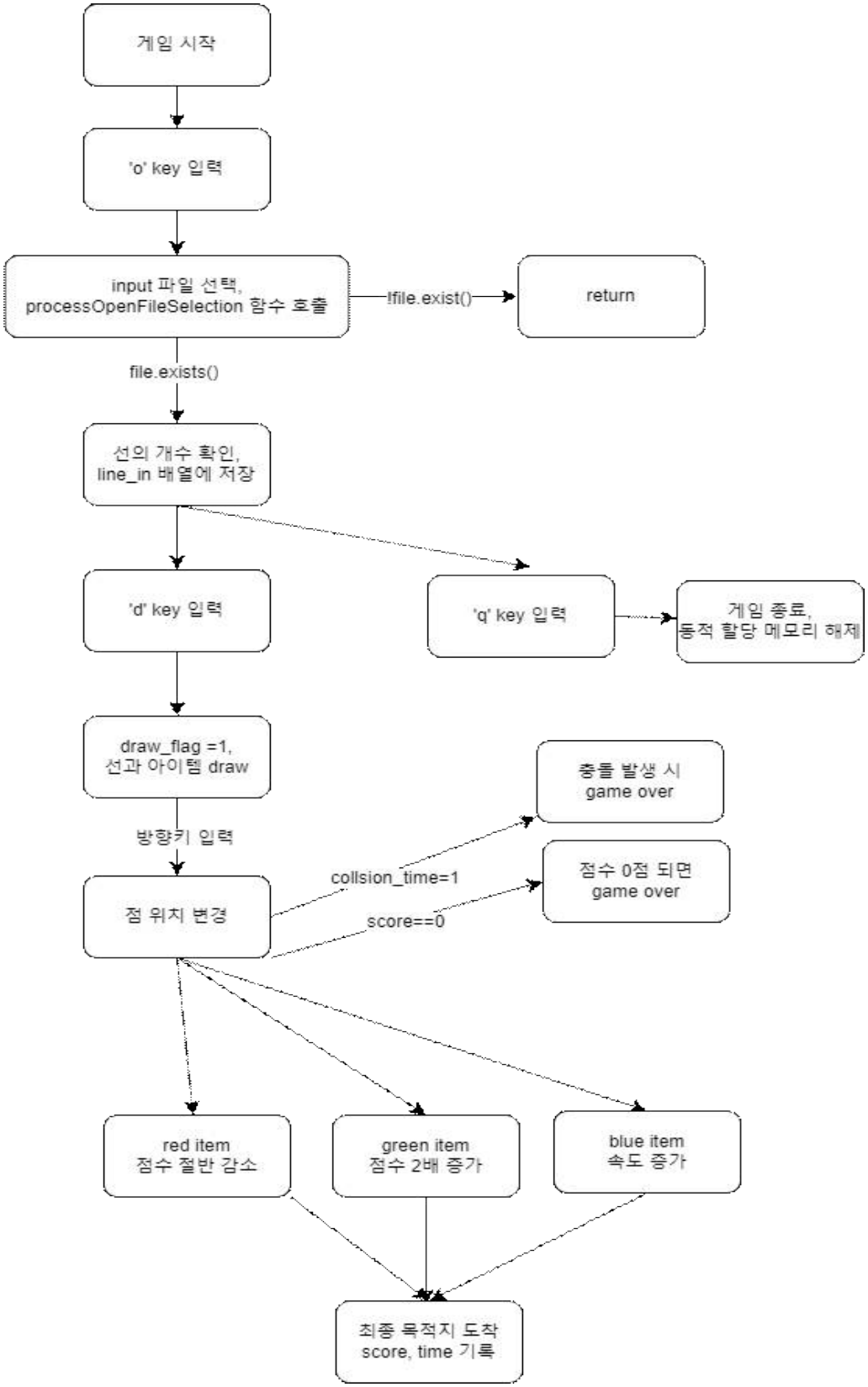
시간 복잡도: O(n + m)

게임의 매 프레임 처리 과정에서 주요 시간 복잡도는 O(n + m)이다. 이는 충돌 감지(n)와 아이템 업데이트(m), 그리기(n + m)에서 비롯된다. 나머지 초기 설정, 키 입력 처리, 게임 종료 조건 확인 등의 과정은 상수 시간 O(1) 내에 수행된다.

공간 복잡도: O(n + m)

선의 좌표 데이터(n)와 아이템 데이터(m)를 저장하는 데 필요한 메모리는 O(n + m)이다. 추가적인 설정과 과정에서 사용되는 메모리는 고정되어 있어 상수 공간 O(1)을 차지한다.

6. 플로우 차트


게임 시작
'o' key 입력
input 파일 선택,
processOpenFileSelection 함수 호출
!file.exist()
return
file.exists()
선의 개수 확인,
line_in 배열에 저장
'd' key 입력
'q' key 입력
게임 종료,
동적 할당 메모리 해제
draw_flag =1,
선과 아이템 draw
충돌 발생 시
game over
방향키 입력
collsion_time=1
점수 0점 되면
game over
점 위치 변경
score==0
red item
점수 절반 감소
green item
점수 2배 증가
blue item
속도 증가
최종 목적지 도착
score, time 기록

7. 창의적 구현 항목

본 프로젝트에서는 다음과 같은 창의적인 요소를 추가하여 게임의 재미와 복잡성을 높였다.

1) 아이템 생성 및 이동
아이템이 랜덤한 위치에 생성되며, 랜덤한 속도로 화면 내를 이동한다.
아이템은 일정 시간이 지나면 사라지며, 아이템의 수명도 랜덤하게 설정된다.
아이템의 종류는 세 가지로, 각각 점수 2배 증가, 점수 절반 감소, 플레이어 속도 2배 증가의 효과를 가진다.

2) 아이템 효과 적용
플레이어가 아이템을 수집하면 즉시 효과가 적용된다.
점수 2배 아이템은 현재 점수를 2배로 증가시키고, 점수 절반 감소 아이템은 점수를 절반으로 감소시킨다. 속도 증가 아이템은 플레이어의 속도를 2배로 증가시키며, 3초 후에 원래 속도로 복구된다.

3) 게임 오버 및 게임 클리어 조건:
플레이어가 장애물(선)에 충돌하거나 점수가 0점이 되면 게임이 종료된다. 게임 오버 시 2초 후에 프로그램이 종료되며, 화면에는 "Game Over" 메시지가 표시된다.

플레이어가 목적지에 도달하면 게임이 클리어되며, 화면에 "Game Clear!!" 메시지가 표시되고 경과 시간이 출력된다.

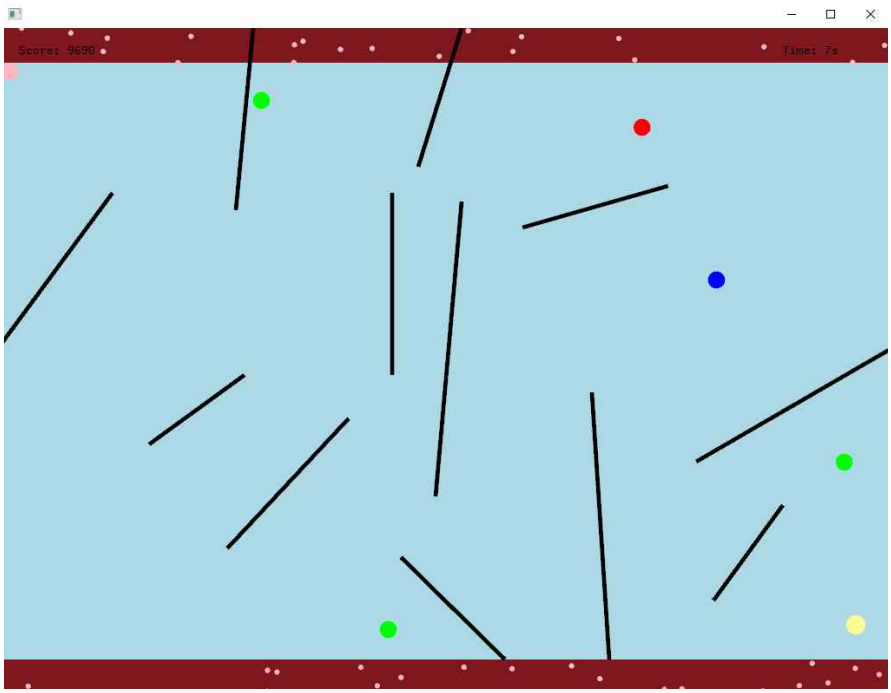
4) UI 요소
현재 점수와 경과 시간이 화면에 실시간으로 표시되며, 게임 오버 시와 게임 클리어 시 각각의 상태를 알리는 메시지가 화면에 표시된다.

8. 프로젝트 실행 결과

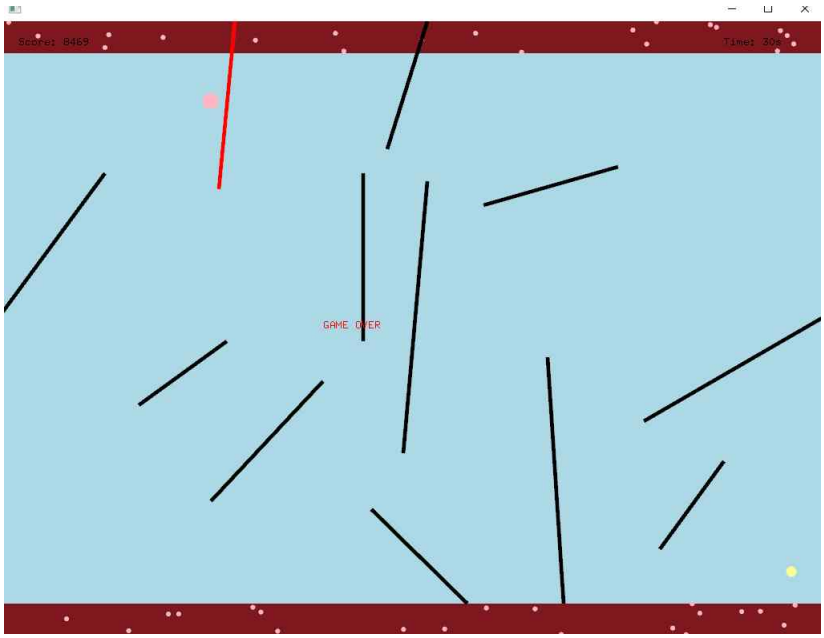
초기 화면

‘o’ key (open)을 누른 이후, ‘d’ key (draw)를 누른 직후
(상단에 score와 time이 표시되어 있다)

빨간색 원 – 점수 감소 아이템
초록색 원 – 점수 2배 아이템
파란색 원 – 속도 증가 아이템



장애물을 만난 경우 - Game over와 함께, 선이 빨간색으로 변한다.



9. 느낀 점 및 개선 사항

본 프로젝트를 통해 C++와 openFrameworks를 활용하여, 다양한 게임 기능을 구현해 볼 수 있었다.

충돌 감지 알고리즘을 구현하며 직선의 방정식을 사용하여 문제를 해결하는 것이 재미있었고, 점과 선 사이의 거리 계산과 수직 거리 개념을 통해 충돌 여부를 판단하며 수학적 지식을 코딩에 활용하는 경험을 할 수 있었다.

또한, 처음에는 단순히 장애물 피하기 게임을 목표로 코드를 구현했으나 구현하다 보니 아이템도 추가하고, 각 아이템이 점수나 속도에 미치는 영향을 구현하며 게임의 복잡도를 높일 수 있었던 것 같다.

그리고 게임 개발 과정에서 발생한 여러 문제를 디버깅하고 해결하는 과정을 통해 코딩 실력이 많이 늘 수 있었다고 생각한다.

다만, 개선하고 싶은 점은 장애물이 있을 때 최적 경로를 찾는 알고리즘을 추가하고 싶다. 프로그래밍 언어 시간에 배운 다익스트라 알고리즘을 활용하여 최적 경로를 찾고자 하였으나, 점이 10*10 크기로 움직이고, 움직이는 모든 경로를 고려해야 하므로 구현하기가 어려웠다. 다익스트라 알고리즘은 그래프의 모든 노드를 탐색하

여 최단 경로를 찾기 때문에, 게임 환경에서 점이 움직이는 모든 가능한 경로를 고려하면 시간 복잡도가 매우 높아질 것이라 생각한다.

따라서, 그리드 기반 경로 탐색을 통한 최적 경로를 찾는 알고리즘을 추가하여, 상용자가 'g'키를 눌렀을 때, 현 위치에서 가장 빠르게 최종점에 도달할 수 있는 경로를 보여주도록 코드를 구현해보고 싶다.