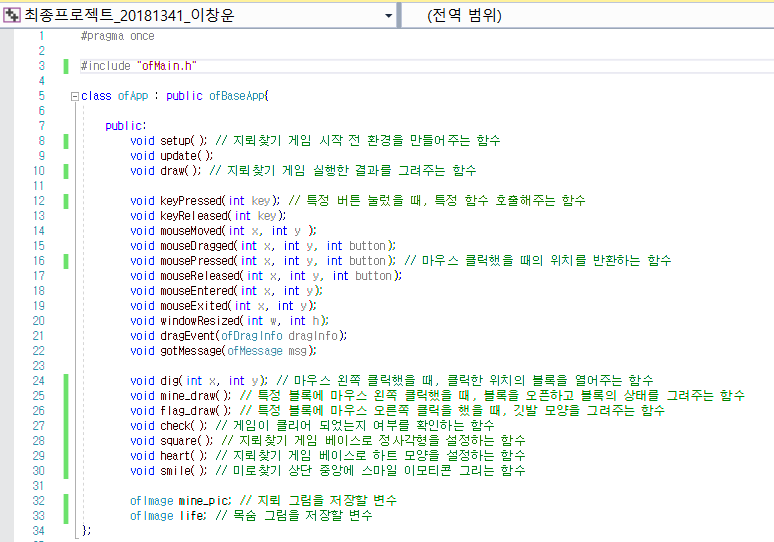
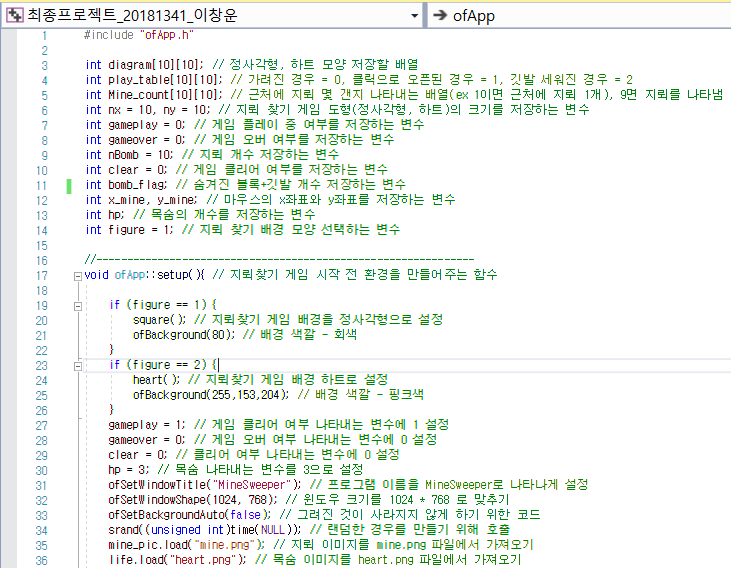
최종 프로젝트 보고서

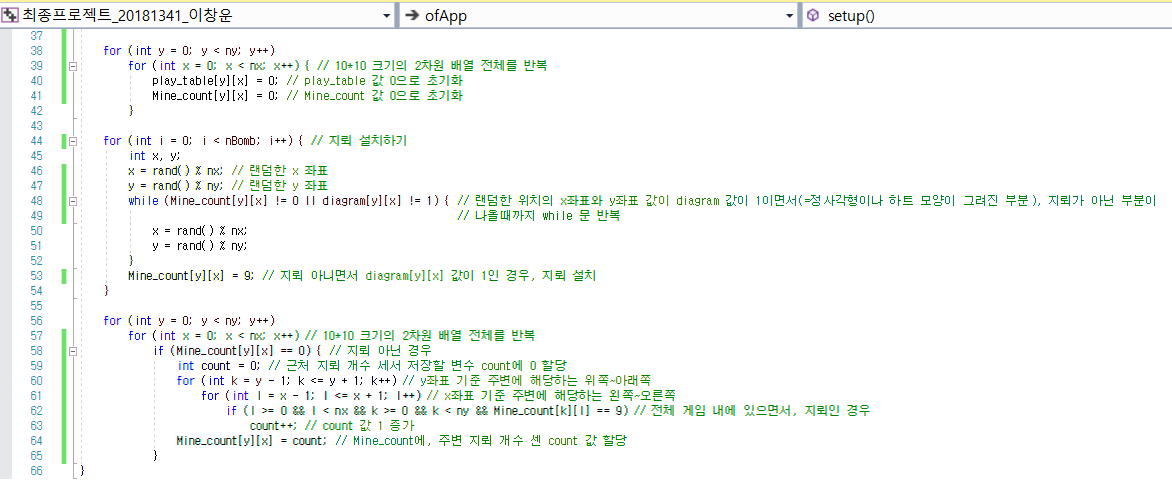
전공: 물리학과 학년: 3학년 학번: 20181341 이름: 이창운

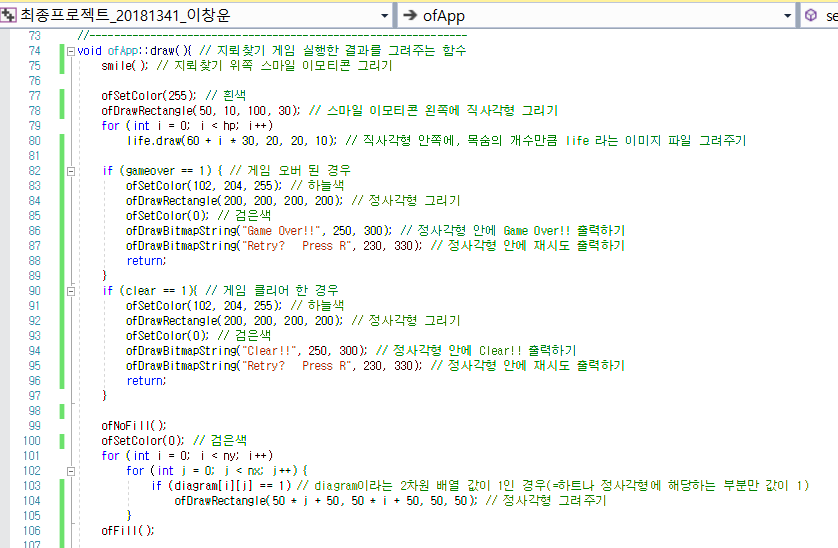
**0. 전체 코드**

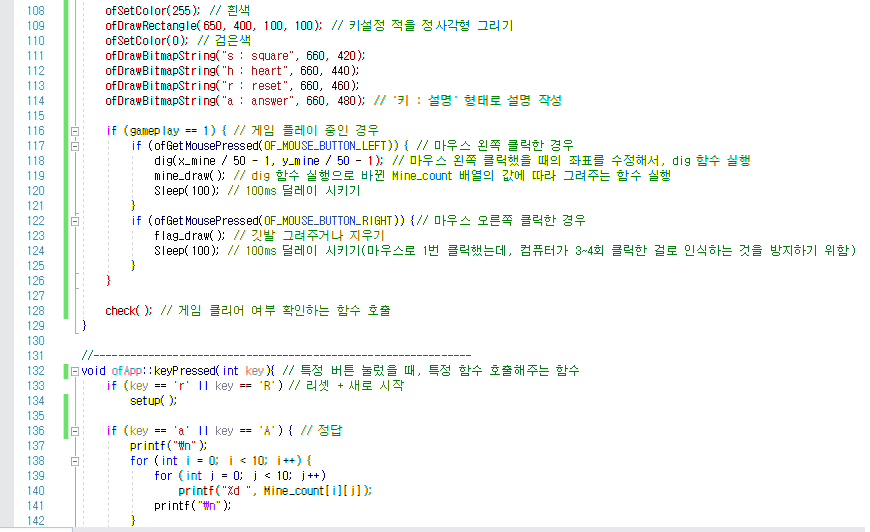


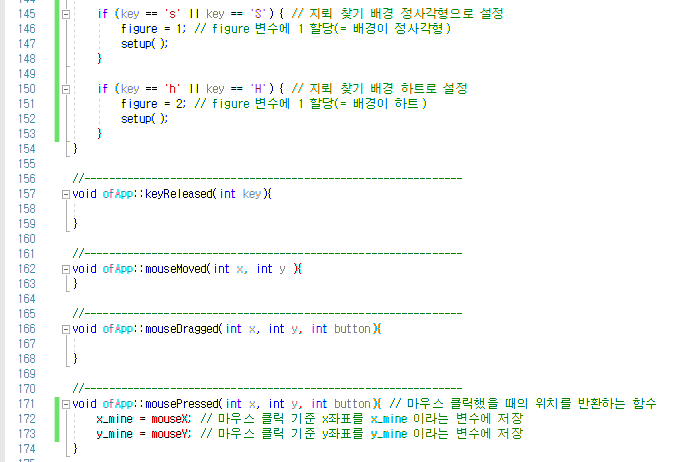
<ofApp.h>

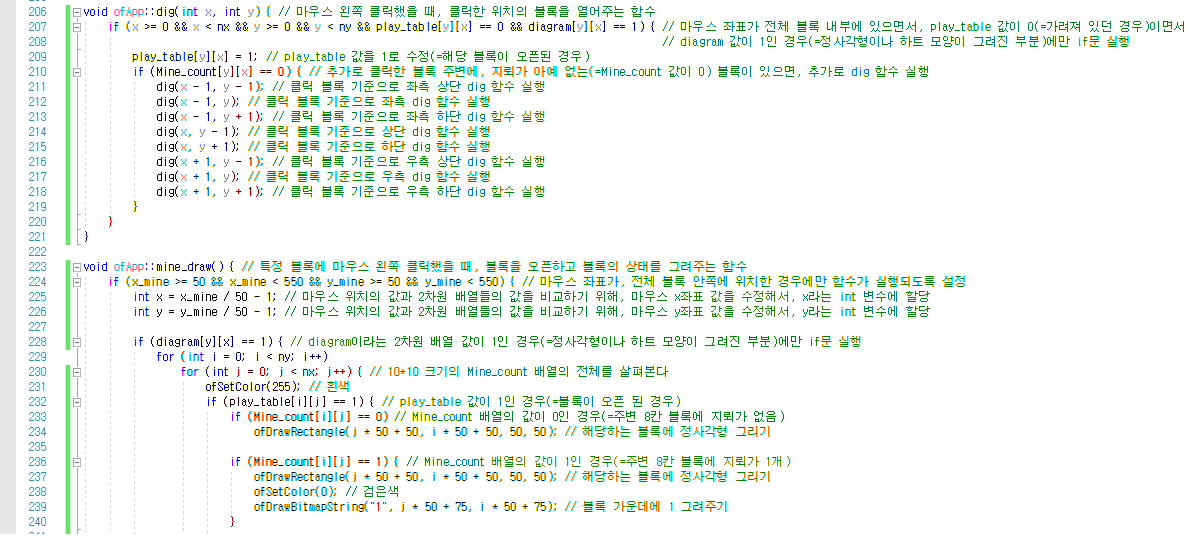






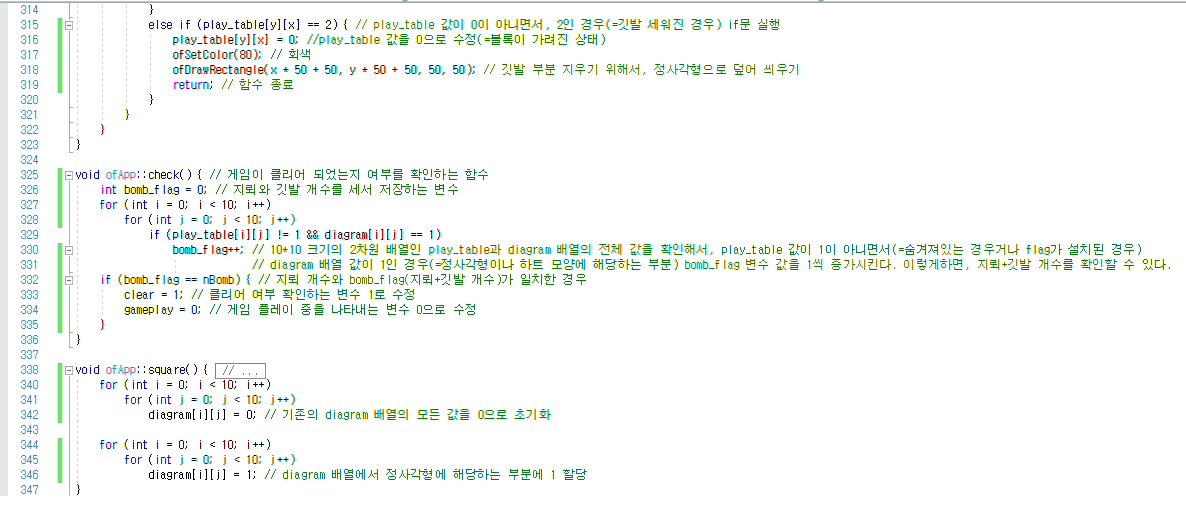


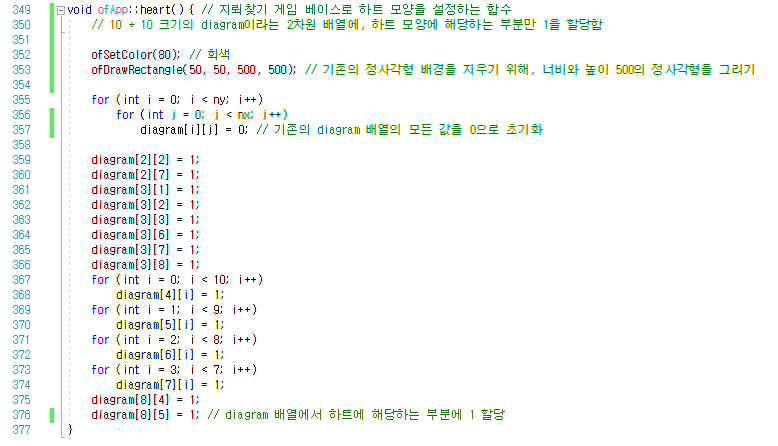


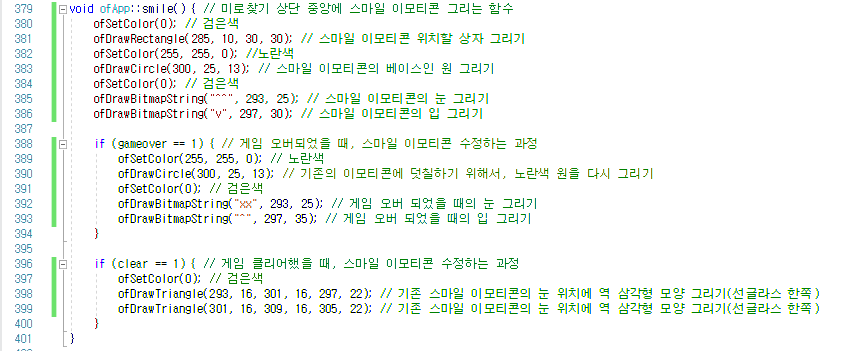












<ofApp.cpp>

**1. 프로젝트 목표와 실험 환경**

프로젝트에서 구현하고자 한 목표는 ‘지뢰 찾기(minesweeper)’ 게임을 OpenFramework를 이용해 구현하는 것이다. 추가로 정사각형 모양의 배경 뿐만 아니라 다른 형태의 도형도 구현하고자 하였다.

사용한 실험 환경은 Visual Studio 2017, Window 11이다.

**2. 각 변수에 대한 설명**

**int\*\* diagram** : 지뢰 찾기 게임의 배경이 되는 도형을 저장하는 배열이다. 전체 배열의 값을 모두 0으로 저장하고, 배경이 되는 도형이 그려진 부분에 한해서만 값을 1로 수정하는 식으로, 배경의 도형을 저장하였다. 예를 들어 2\*2 크기의 정사각형을 diagram 배열에 저장하면

1 1

1 1

과 같이 저장된다. 이번 프로젝트에서는 정사각형과 하트 모양을 저장하도록 하였다.

**int\*\* play\_table** : 지뢰 찾기 게임의 한 칸 한 칸의 박스를, 2차원 배열의 한 칸으로 가정해보았을 때, 그 한 칸의 상태를 저장해주는 2차원 배열이다. 한 칸의 상태는 숨겨져 있는 상태, 열린 상태, 깃발이 세워진 상태로 기본적인 3개의 상태와, 폭탄을 발견해서 그 칸이 이미 열려 있는 상태를 추가해, 총 4개의 상태를 저장하도록 하였다.

**int\*\* Mine\_count :** 마찬가지로 지뢰 찾기 게임의 내용을 2차원 배열로 고려했을 때, 어느 지점에 지뢰가 있는지, 어느 지점 근처에 지뢰가 몇 개 존재하는지를 저장할 2차원 배열이다. 여기서 말하는 근처는, 한 칸을 기준으로 상하좌우, 대각선까지 포함해 총 8칸을 말한다. 근처에 지뢰가 없으면 0, 지뢰가 1개면 1을 저장하고, 만약 폭탄인 경우에는 9를 저장하도록 하였다.

**int nx, ny :** 지뢰 찾기는 n\*n 크기의 정사각형 안에서, 배경이 되는 도형이 그려져서 진행되는데, 이때의 n 값을 저장해줄 변수이다.

**int gameplay** : 지뢰 찾기 게임이 플레이 되는 중, 다시 말해 게임 오버 되거나 클리어 된 상태가 아닌 상태를 저장하기 위한 변수이다. 게임 오버나 클리어 된 경우, gameplay 변수의 값이 0이 되고, 그 외의 경우에는 1로 저장되어있다.

**int gameover :** 지뢰 찾기 게임은 지뢰를 클릭하면 게임이 종료된다. 이 변수는, 게임 종료 여부를 저장해주는 변수로, 지뢰를 클릭해 게임이 종료되면 1이, 평소에는 0이 저장되어있다.

**int nBomb :** 지뢰 찾기 게임에서, 몇 개의 지뢰를 설치할지를 저장해주는 변수이다.

**int clear :** 지뢰 찾기 게임은 ‘깃발 개수 + 숨겨져 있는 타일 개수’가 지뢰의 개수와 일치하면 게임이 클리어 된다. 이 변수는 클리어 했는지 여부를 저장해주는 변수로, 평소에는 0이, 클리어 했을 시에는 1이 저장되도록 하였다.

**int bomb\_flag :** 지뢰 찾기 게임 화면 내에서, 총 몇 개의 타일이 숨겨져 있는 상태로 존재하는지, 깃발은 몇 개가 세워져 있는지, 이들의 총 개수를 더해서 저장해주는 변수이다.

**int x\_mine, y\_mine :** 마우스를 클릭했을 때, 마우스가 어디에 위치해 있는지를 저장해주는 변수들이다.

**int hp** : 이번 프로젝트의 경우, 지뢰를 3번 클릭한 경우 게임 오버 되도록 설정하였다. 그래서 지뢰를 몇 번 클릭했는지 여부를 저장하기 위해, hp 라는 변수를 사용하였다.

**int figure** : 지뢰 찾기 게임의 배경을 선택해주는 변수이다. 1인 경우 square 함수가 호출되어서 정사각형 모양이 배경이 되도록 하였고, 2인 경우 heart 함수가 호출되도록 해서 하트 모양이 배경이 되도록 하였다.

**3. 각 함수에 대한 설명**

- **setup**

지뢰 찾기 게임 시작 전 환경을 구성해주는 함수이다. setup 함수가 호출되면, 변수들인 gameover 값은 1로, clear는 0, hp는 3, gameplay는 1로, 2차원 배열들인 play\_table, Mine\_count는 모두 0으로 초기화 해준다. 추가로, Mine\_count 배열에 랜덤한 위치에 폭탄을 설치, 각 배열의 값마다, 근처에 폭탄이 몇 개 존재하는지 찾고 그 값을 Mine\_count에 저장해준다.

- **draw**

지뢰 찾기 게임의 틀(스마일 이모티콘, 목숨 개수, 정사각형이나 하트 틀)을 그려주고, 마우스 왼쪽 클릭, 오른쪽 클릭의 결과를 그려주는 함수이다. 추가로, 게임 클리어 했을 때와 게임 오버 했을 때의 결과도 그려준다.

마우스 왼쪽 클릭이 발생하면, dig 함수로 클릭한 위치의 play\_table 배열의 값을 수정해주고, 이를 토대로 mine\_draw 함수로 바뀐 결과들을 그려준다. 마우스 오른쪽 클릭이 발생하면, flag\_draw 함수를 호출해 깃발을 그려주거나 지워준다. 추가로, 마우스 오른쪽 클릭의 경우, 오른쪽 클릭을 1회 했다 생각했는데, 컴퓨터는 3~4회 클릭한 것으로 인식하는 경우가 종종 발생해, 이를 방지하고자 Sleep 함수로 딜레이를 일부로 주었다.

- **keyPressed**

키보드에서 특정 버튼(이번 프로젝트에서는 r, a, s, h)을 눌렀을 때, 특정 함수를 호출하는 함수이다. r 또는 R 키는 ‘reset’ 버튼으로, setup() 함수를 호출해, 지뢰 위치도 다 바꿔서 게임을 처음부터 시작하도록 하는 버튼이다. a 또는 A 키는 ‘answer’ 버튼으로, 누르면 cmd 창에 Mine\_count 배열의 전체를 출력해줘서, 어디에 지뢰가 위치해 있는지 알려준다. s 또는 S 키는 ‘square’ 버튼으로, square 함수를 호출해, 지뢰 찾기 게임을 정사각형 틀에서 진행하도록 해준다. h 또는 H 키는 ‘heart’ 버튼으로, heart 함수를 호출해, 지뢰 찾기 게임을 하트 틀에서 진행하도록 해준다.

- **mousePressed**

마우스가 클릭했을 때의 위치를 x\_mine, y\_mine 변수로 저장해서 내보내는 함수이다.

- **dig(int x, int y)**

지뢰 찾기 게임의 메인 함수 부분으로, 마우스 왼쪽을 클릭했을 때, 클릭한 부분의 play\_table 2차원 배열의 값을 Open(=1)으로 바꿔주는 부분이다. 추가로, 지뢰 찾기 게임을 진행하다 보면 클릭한 블록 기준으로, 근처 블록들의 주변에 지뢰가 없는(=Mine\_count 배열의 값이 0) 경우가 있는데, 이 경우, 그 근처의 블록들도 dig 함수가 실행되도록 재귀 함수를 구현하였다.

- **mine\_draw()**

마우스 왼쪽 클릭으로 블록이 Open 되었을 때, Open된 블록을 그려주는 함수이다. 마우스 클릭이 발생했을 때 마우스의 x좌표와 y좌표를 수정해서, Mine\_count[0][1], Mine\_count[2][3]과 같이 Mine\_count라는 2차원 배열의 값을 선택할 수 있게, 마우스 클릭했을 때의 좌표를 수정해서 x와 y라는 int 변수에 저장해주었다. 이후, 선택된 Mine\_count 값이 0이면, 해당 위치에 흰색 정사각형을 그려주었고, Mine\_count의 값이 1~8이면 흰색 정사각형 바탕에, 정사각형 중앙에 해당하는 숫자가 나오게 해주었다. 추가로, Mine\_count 값이 9이면 지뢰 이미지를 가져와서 그려주었고, 목숨의 개수를 나타내는 hp 변수 값을 1 감소시켜주었다. 이후 같은 과정을 반복해, hp 변수가 0이 되면, 게임 오버되도록, 게임 오버를 나타내는 변수와 게임 플레이 도중을 나타내는 변수의 값들을 수정해주었다.

- **flag\_draw()**

마우스 오른쪽 클릭을 했을 때, 블록이 가려져 있었으면(=play\_table 값이 0), play\_table 값을 2로 수정하고, 깃발을 그려주고, 블록에 깃발이 그려져 있었으면(=play\_table 값이 2), play\_table 값을 0으로 수정하고, 깃발을 지워주는 함수이다. mine\_draw 함수와 마찬가지로, 마우스 클릭했을 때의 좌표가, diagram이나 play\_table이라는 2차원 배열을 가리키도록, 좌표를 수정해서 x와 y라는 int 변수에 저장해서 사용하였다.

- **check()**

지뢰 찾기 게임이 클리어 되었는지 여부를 확인해주는 함수이다. check 함수가 호출될 때마다, play\_table 배열과 diagram 배열의 전체를 확인해서, 도형이 그려진 부분이면서(diagram 배열 값이 1인 경우), ‘숨겨져 있는 블록 개수 + 깃발이 세워진 블록 개수’가 지뢰 개수와 일치한지 확인하고, 일치하면 clear 변수를 1로 수정해주었다.

- **square()**

지뢰 찾기를 플레이하는 배경으로 정사각형을 선택해주는 함수이다. 함수가 호출되면, diagram이라는 10\*10 크기의 2차원 배열을 모두 0으로 초기화 해준 뒤, 정사각형에 해당하는 부분만 모두 1로 수정해준다.

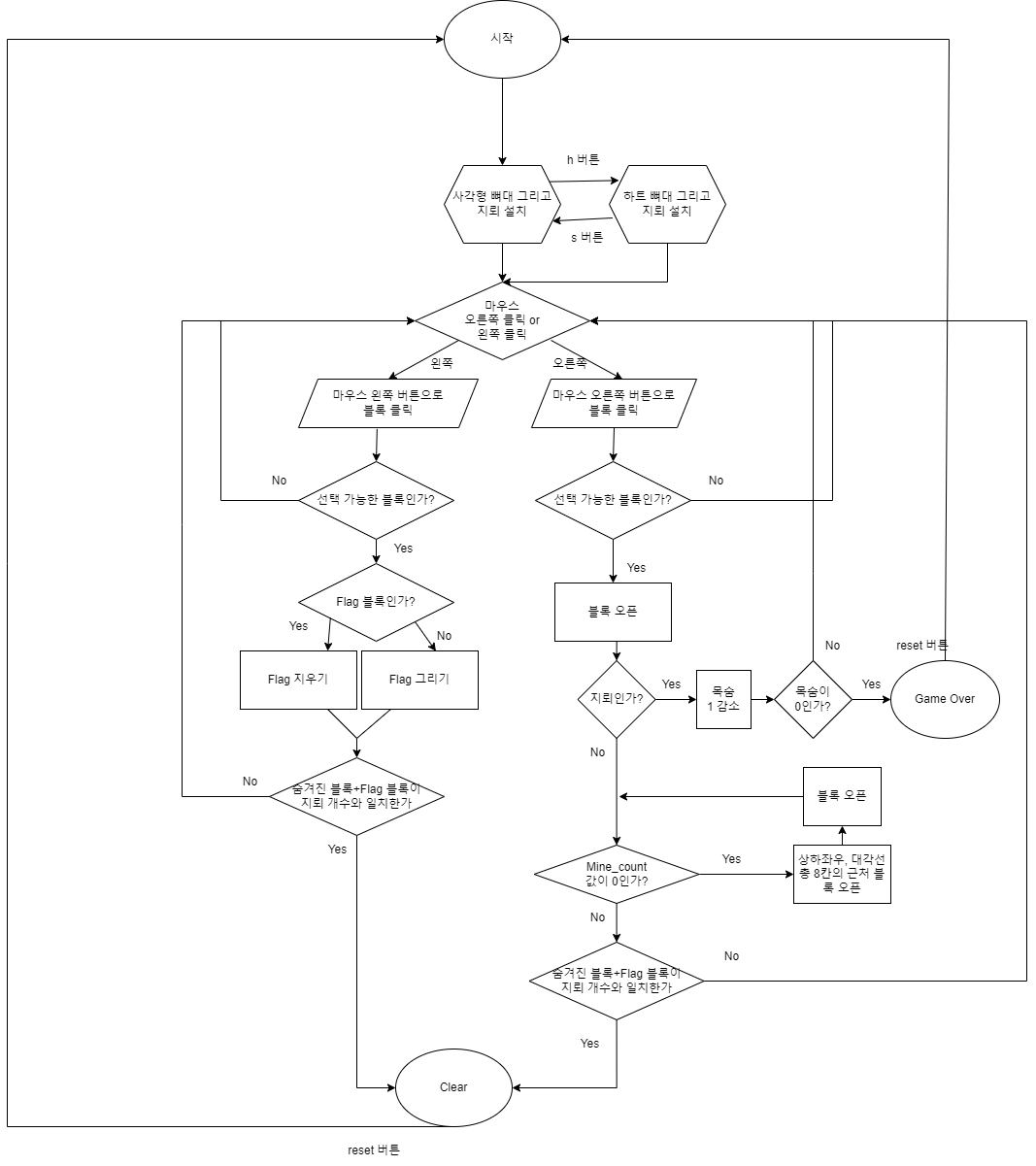
- **heart()**

지뢰 찾기를 플레이하는 배경으로 하트 모양을 선택해주는 함수이다. 함수가 호출되면, diagram이라는 10\*10 크기의 2차원 배열을 모두 0으로 초기화 해준 뒤, 하트에 해당하는 부분만 모두 1로 수정해준다.

- **smile()**

지뢰 찾기 게임 상단 중앙에, 스마일 이모티콘을 그려주는 함수이다. 평소에는 노란 원을 베이스로, ^v^ 와 같은 표정을 그려주었고, 게임이 클리어 되었을 경우에는 눈 부분에 역삼각형을 그려주어서, 선글라스를 쓰게 해주었고, 게임 오버 되었을 경우에는 x­­^x 와 같은 표정을 그려주었다.

**4. 프로젝트 전체 플로우차트, 자료구조, 알고리즘, 시간/공간 복잡도**



<**Fig 1.** 전체 프로젝트 flow chart>

이번 프로젝트에서 사용한 자료구조는 2차원 배열들이다. 배경이 되는 도형을 그려줄 diagram 배열, 지뢰의 위치와 근처에 몇 개의 지뢰가 존재하는지를 저장한 Mine\_count 배열, 지뢰 찾기 게임 진행 정도를 저장한 play\_table 배열들이다.

smile() 함수의 경우 gameover, clear 변수만 호출해서 사용하였으므로 시간/공간복잡도가 모두 O(1)이다.

heart(), square() 함수의 경우 diagram이라는 10\*10 크기의 2차원 배열을 저장해서 사용하였으므로 공간복잡도는 O(100)이고, 두 함수 모두 for 반복문을 10\*10 회 반복해 사용하였으므로 시간복잡도 역시 O(100)이다.

check(), flag\_draw(), mine\_draw() 함수의 경우 for 반복문을 100회 반복하였고, play\_table과 Mine\_count와 같은 2차원 배열을 저장해서 사용하였으므로 시간/공간복잡도가 모두 O(100)이다.

지뢰 찾기 게임의 경우, 특정 블록을 선택했을 때, 그 블록이 지뢰가 아니면서 근처에 지뢰도 없는, 즉 Mine\_count 값이 0인 블록인 경우, 그 블록의 근처의 블록들을 오픈해주는 알고리즘을 가지고 있다. 이를 dig() 함수로 구현하였다. 클릭한 블록의 play\_table 값을 1로 바꿔서 블록을 오픈한 상태로 바꿔준다. 이후, 오픈한 블록의 Mine\_count 값이 0인 경우, 오픈한 블록의 상하좌우, 대각선을 포함해, 주위 8칸의 좌표를 이용해, dig 함수를 호출한다. 오픈한 블록의 왼쪽 블록의 Mine\_count 값이 1인 경우, dig() 함수가 호출되었으므로 play\_table 값이 1로 바뀌어 블록이 오픈되지만, Mine\_count 값이 0이 아니므로 근처 블록들의 좌표를 dig() 함수에 넣지 않는다. 반대로, 오픈한 블록의 오른쪽 블록의 Mine\_count 값이 0인 경우, dig() 함수를 호출해 play\_table 값을 1로 바꿀 뿐만 아니라, 또다시 오픈한 블록 기준으로 상하좌우, 대각선의 좌표들을 dig() 함수에 집어넣는다. 이와 같은 방식으로 연결되어 있는, 주변에 지뢰가 없는 블록들의 주변 8칸의 블록들을 모두 열어준다.

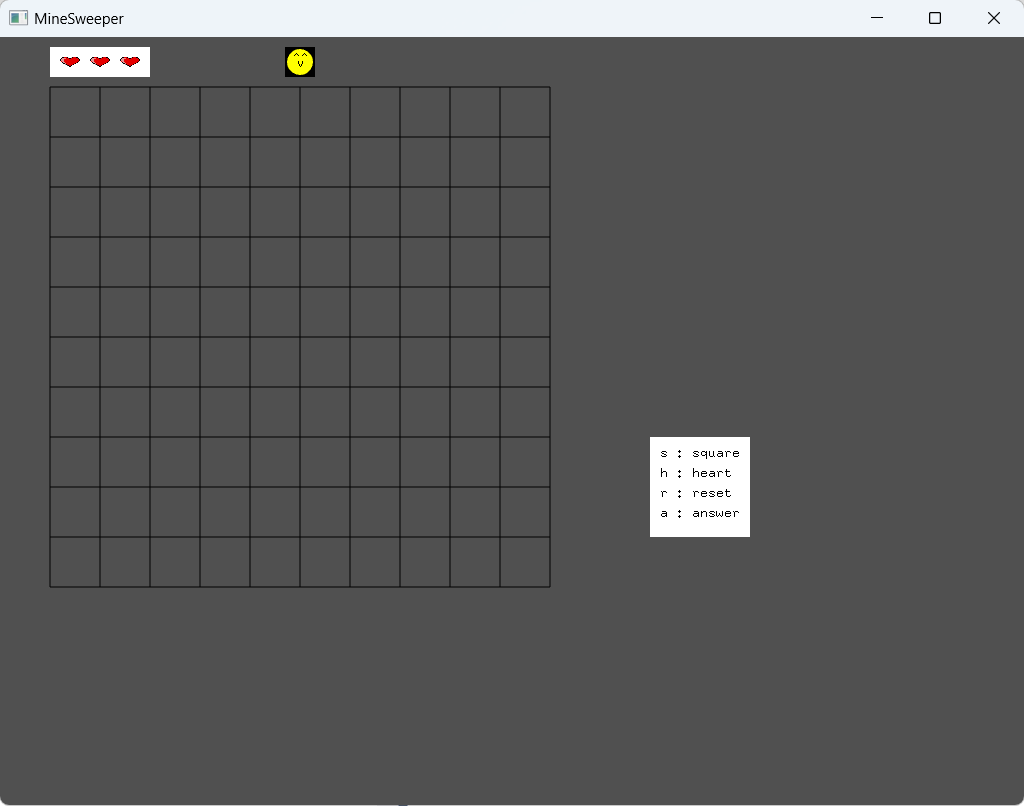
dig() 함수의 경우 재귀함수가 존재한다. 재귀 함수가 없다 생각했을 때 함수의 시간복잡도는 O(1)이다. 최악의 경우 10\*10 크기의 전체 배열을 모두 확인하므로, O(1 \* 100)인 O(100)이 시간복잡도이다. 공간복잡도는 Mine\_count와 play\_table, diagram 2차원 배열을 저장해서 사용하므로 O(100)이다.

**5. 어떻게 창의적 구현을 하였는지**

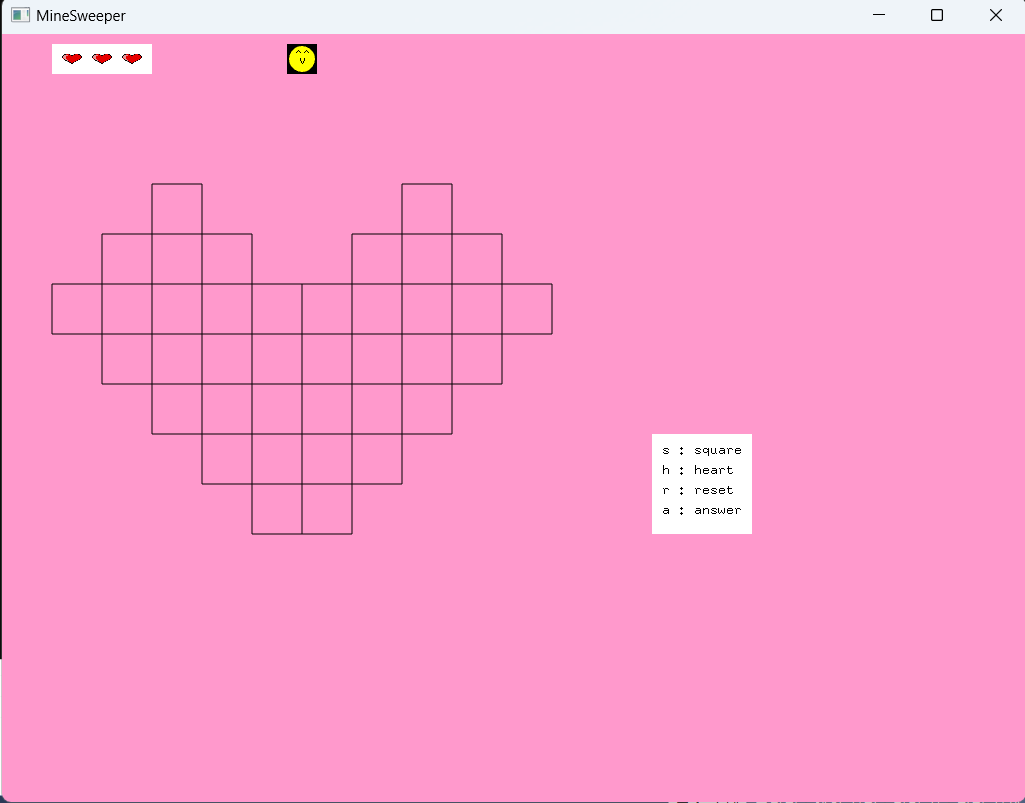
기존에 시도해 보았던 코드들이나, 실습에서 사용한 코드들은 대부분 키보드를 사용하였고, 마우스는 파일이나 기능을 선택하는 정도로만 사용되었다. 하지만 우리가 플레이하는 게임들은 보통 마우스+키보드를 동시에 사용하는 식으로, 마우스가 많이 사용된다. 그래서 마우스를 이용하는 프로젝트를 만들고자 하였다. 마우스를 이용하면서, openframework를 이용해 구현할 수 있는 게임을 생각해보았을 때, 지뢰 찾기가 딱 맞다는 생각에 지뢰 찾기 게임을 구현하였다.

다만 지뢰 찾기 게임을 그대로 구현했을 때 창의적인 부분이 없는 것 같아, 지뢰 찾기 게임을 하면서 흔히 봐온 정사각형 뿐만 아니라, 하트를 추가로 구현하였다. 그런데 하트 모양의 경우, 정사각형 모양과 달리 주어진 정보로는 지뢰를 구분할 수 없는 부분이 존재해서, 목숨이라는 개념을 도입해 사용하였다.

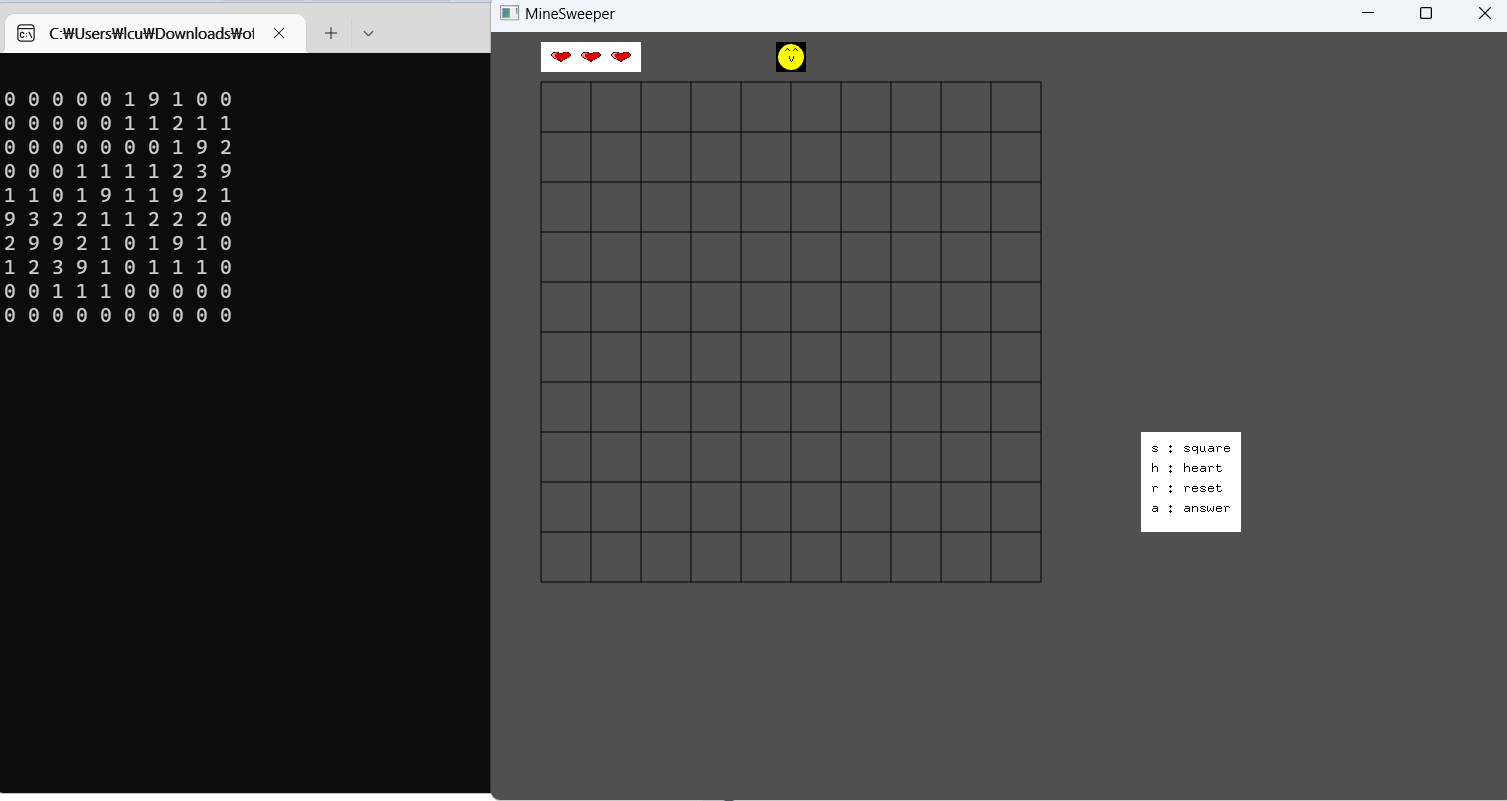
**6. 프로젝트 실행 결과**



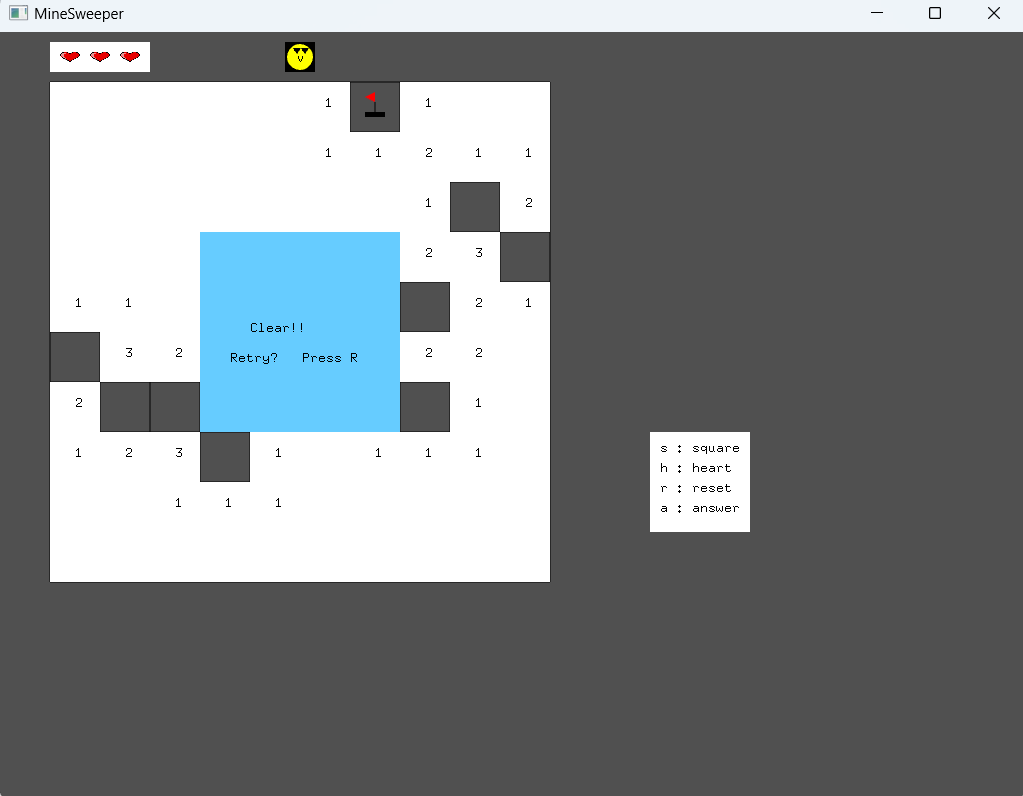
<**Fig 2.** 기본 실행 했을 때 화면>



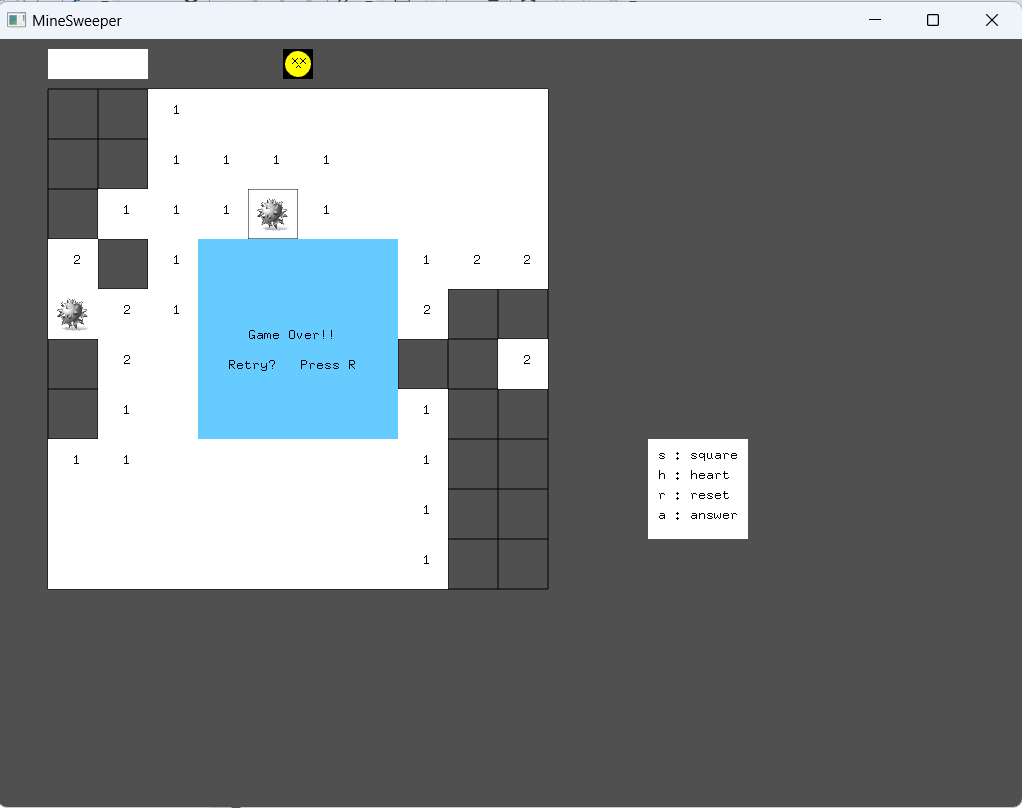
<**Fig 3.** h 키 눌렀을 때 화면>



<**Fig 4.** a 키 눌렀을 때 화면>



<**Fig 5.** clear 했을 때 화면>



<**Fig 6.** gameover 했을 때 화면>

**7. 느낀 점 및 개선 사항**

우선, 아주 간단한 게임이지만 그래도 하나의 게임을 만들었다는 사실이 뿌듯했던 것 같다. 특히 OpenFramework를 사용해서 마우스를 이용해 코드를 짠 부분이, 익숙지 않아서 해매기도 했지만, 결국에는 프로그램을 완성해서 잘 작동하는 것이 되게 신기하였다.

다만 아쉬웠던 점으로는, OpenFramework를 아직 잘 다루지 못해서, 생각만 하고 구현하지 못한 기능들이 많았던 점이다. 테트리스 실습에서 구현한 랭킹 시스템이나, 지뢰를 클릭했을 때 터지는 효과가 나오거나, 게임 클리어 했을 때 메시지 출력 뿐만 아니라 뒤로 폭죽이 터지는 식으로 이펙트가 나오는 등의 것들을, 어떻게 구현할 지 몰라서 구현하지 못하였다. 특히 이펙트의 경우, 직접 구현해보거나 이미지 파일을 가져온 것처럼 비디오 파일을 가져와서 사용하면 될 것 같았는데, 둘 다 잘 되지 않았다. 또한, 하트 모양을 diagram이라는 2차원 배열에

00000

01010

과 같은 식으로 구현해 놓았기에, 지뢰 찾기 게임의 가로 세로 크기를 키울 수 없었던 점 역시 아쉬웠던 것 같다.

개선 사항으로는, 프로그램이 실행되면 update->draw가 계속 반복되고, 이로 인해 draw 함수에서 사용한 mine\_draw 함수나 check 함수에서 사용되는 이중 for 반복문이 지속적으로 호출되는 부분이다. 이번 프로젝트에서 구현한 지뢰 찾기 게임의 크기가 작아서 이번에는 문제가 되지 않았던 것 같은데, 조금 더 크게 구현하였으면, 프로그램의 진행 속도가 굉장히 느려졌을 것 같다. 또한 아쉬운 점에서 언급했던, 2차원 배열에 하트 모양을 크기에 맞춰서 구현하는 방법을 찾는다면, 지뢰 찾기 게임의 크기가 늘어나 더 재미있게 플레이 할 수 있을 것 같다.