최종 프로젝트보고서

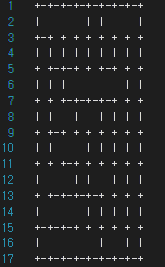
전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 프로젝트 목표 및 실험환경**

이 프로젝트의 목표는 기존 실습시간에 오픈프레임워크로 구현했던 미로를 이용해, 픽셀 게임을 제작함으로서 오픈프레임워크에 대해 더 잘 이해하는 것이다. 실험환경은 실습때와 같이 오픈프레임워크와 visual studio 2017로 진행하였다.

**2. 변수**

우선, 텍스트 파일의 미로를 입력받기 위해 다음과 2차원 포인터 변수를 선언했다. 변수명은 input이다. 이 포인터배열은 미로의 너비와 높이만큼(실제 미로의 높이와 너비와는 다르다. 텍스트파일로 할당 받을 경우 실제 미로의 너비와 높이보다 2배 +1 만큼이다.) 동적할당을 한 후, 텍스트 파일의 각 줄을 읽고 변수에 값을 할당한다.



8 \* 8 미로이지만, 동적할당 해야할 배열의 크기는 17 \* 17이다.

이 프로젝트에서는 DFS만 사용하며, 이때 사용할 배열이 필요하다. 변수명은 visit이며, 2차원 포인터배열로 할당 받는 크기는 input 변수와 동일하다. 같은 배열에 초기 값을 설정해주어야 한다. 우선 보통, 미로탐색을 배열로 표현할 때, 움직일 수 있는 공간은 1로 벽의 경우 0으로 설정해준다. 하지만, 이렇게 했을 경우 방문했던 길을 표현하기 어려웠다. 그래서, 할당 된 값을 보고 visit배열의 colum과 row를 유츄할 수 있는 값을 주기로 했다.

for (int cnt = 0; cnt < HEIGHT; cnt++) {

for (int j = 0; j < WIDTH; j++) {

if (input[cnt][j] == '+' || input[cnt][j] == '-' || input[cnt][j] == '|') visited[cnt][j] = 0;

else {

visited[cnt][j] = 10000 \* cnt + j;

}

}

}

포문을 통해서 visit 배열을 초기화 해준 후, 미로에서 움직일 방향에 대한 배열도 설정한다. 미로에서는 상,하,좌,우로 움직일 수 있다. 이때, 미로의 시작점과 도착점이 고정되어있고, 위치 상 오른쪽 아래 위 왼쪽으로 탐색하는 것이 조금 더 빠를 것이라고 생각했다.

int x[4] = { 1, 0, 0, -1 };

int y[4] = { 0, 1 , -1, 0 };

이후, 목표 지점을 만나면 반복문을 탈출해야 한다. 목표 지점은 n\*m미로에서 n\*m지점으로 설정했다. DFS에서 사용할 stack의 경우에는 stack 라이브러리를 사용했으며, int형 변수 path로 정의했다. 이후, 첫 시작점의 visit 값을 psuh 해주고, visit 배열의 값을 1로 바꾸어준다.

픽셀 게임을 구현하기 위해서, 인터넷에 무료로 배포되어있는 픽셀 이미지들을 사용해야 했다. 오픈프레임워크에서 이미지를 불러오기 위해 사용하는 변수의 데이터타입은 ofImage 이다. ofImage의 데이터 타입을 가진 변수들은 모두 이미지에 관한 변수들이다.

각 이미지들을 화면에 그리기 위해서는, 좌측 상단의 x좌표와 y좌표를 이용해야 한다. 그래서 각 이미지들의 변수명을 딴 int 형 변수들을 여러 개 선언했다. 예시로 다음은 기사 이미지에 관한 변수들이다. 또한, 각 flag라는 이름을 가진 변수들은, 대부분 이미지의 앞글자와 함께 변수명을 선언하였고 이들은 이미지를 화면에 그릴지 말지 판단하는 변수이거나, 각 상황판단을 하기 용이하게 해주는 변수들이다.

ofImage knight;

int knight\_x = 10, knight\_y = 10;

int k\_flag = 0;

int enter\_flag;

이 변수는 게임이 종료되었을 경우, 누를 수 있는 키를 제한하기 위해서 선언하였다.

int f\_flag = 0;

이 변수는 파일이 열려있는지 아닌지 확인해주는 변수이다.

게임은 애니메이션처럼 동작한다. 이를 위해서 각 이미지들은 독립적으로 움직여야한다. 아이디어로, 시간에 관한 변수를 이용해 이미지가 일정시간이 지나면 사라지게 하고 싶었다. 이를 위해 다음 두 변수를 선언했다. 단순 int형 변수이긴 하지만, time.h 라이브러리와 함께 사용하여, 시간차이를 이용할 수 있었다.

int endTime;

int startTime;

제작한 게임 프로젝트 자체가 복잡한 알고리즘이나 자료구조를 사용해서 구현한 것은 아니기 때문에 동작원리는 간단하며, 변수들도 간단명료하다.

**3. 함수**

함수는 오픈프레임워크에서 기본적으로 주어지는 함수들을 많이 사용했다. 우선, setup() 함수에서 프로그램을 시작하기에 앞서 필요한 설정들을 해주었다. 본 프로젝트에서 중요한 부분은, ofImage로 선언한 변수들을 load 해주는 것이다. 이때, load하기 위해서는 이미지의 위치가 bin/data/images 로 되어있어야 하는 점이다. 이미지로드를 하지 않고 이미지를 사용하면, 에러가 발생한다. 이 외에도, 사용할 폰트나 윈도우 창의 크기, 제목, 메뉴 설정 등이 setup 함수에서 일어난다.

draw() 함수에서는 이미지를 불러와 원하는 곳에 직접 그린다. 다음과 같은 함수들을 호출한다. 함수를 여러 개로 나누어놨을 뿐이며, 각 함수들은 각 상황에 맞게 필요한 이미지들을 그리고 지운다.

draw\_maze(); // 미로 생성

draw\_knight(); // 기사 그리기

draw\_star(); // 별과 관련된 이미지 그리기

draw\_bomb(); // 함정과 관련된 이미지 그리기

draw\_princess(); // 구출해야하는 공주 이미지 그리기

draw\_skill(); // 사용할 수 있는 스킬 이미지 그리기

draw\_attack(); // 공격키 이미지 그리기

draw\_door(); // 이미지는 아니지만, 공주가 갇혀있음을 그려주는 함수

draw\_zombie(); // 좌우로만 움직이는 좀비 하나를 그려준다.

draw\_box(); // 열쇠로 잠겨있는 상자 그리기

draw\_key(); // 상자를 열수있는 열쇠 그리기

draw\_lever(); // 공주를 구출 할 수 있는 레버 생성

draw\_end(); // 플레이가 종료되는 조건에 따른 이미지 그리기

keyPressed() 함수는 키보드의 키가 눌렸을 때 들어온 입력을 받고, 각 입력에 맞는 명령을 실행한다. 이 프로젝트에서는 사용하는 키들은, 4개의 방향키와 파일을 여는 ‘o’, 프로그램을 종료하는 ‘q’, 윈도우 창을 최대로 만드는 ‘f’ , 그리고 게임에서 사용할 스킬인 ‘r’ 과 마지막으로 공격 ‘a’가 있다.

readFile() 함수에서는 미로를 그려놓은 텍스트파일을 읽는 함수이다. 또한, 각 파일을 불러올 때 마다 이전에 사용한 변수들을 초기화 해주어야하며, 동적할당 했던 메모리들은 해제시켜주어야 하기 때문에 readFile()함수에서 그 역할을 한다. 앞서 말한 input 과 visit변수가 여기에서 값을 할당 받는다. 또한, DFS()함수도 여기서 호출되는데, 호출되는 이유는 단순히 visit배열에로부터 도착 경로를 얻어내기 위함이다.

f\_flag = 1; // 파일오픈

s\_flag[0] = 1; s\_flag[1] = 1; s\_flag[2] = 1; //3개의 별에대해 1이면 그림

b\_flag[0] = 1; b\_flag[1] = 1; b\_flag[2] = 1; //3개의 함정에대해 1이면 그림

p\_flag = 1; // 1이면 공주를 그림

H\_flag = 0; // 1이면 game clear 2면 gameover

z\_flag = 1; // 1이면 좀비생성

skill\_num = '3'; // ‘r’를 눌러 사용할 수 있는 스킬의 개수

enter\_flag = 0; // 게임이 끝나면 1로바뀌며 누를 수 있는 키제한

lever\_flag = 0; // 1이면 레버 생성

lever\_shift\_flag = 0; // 1이면 움직인 레버 그리기

box\_flag = 0; // 1이면 상자그리기

key\_flag = 0; // 1이면 열쇠 그리기

right = 1; // 좀비 이동방향

left = 0; // 좀비 이동방향

// 좌표변수 초기화

knight\_x = 10, knight\_y = 10;

princess\_x = 10, princess\_y = 10;

이렇게 변수들을 초기화 해주고 나면, 게임에 사용할 변수들 역시 값을 새로 할당해주어야 한다. 먼저, 박스가 생성될 위치를 시작위치로 지정해준다. 혹은, 취향에 맞게 랜덤위치로 지정해도 된다. 이후, 3개의 별의 위치를 앞서 구한 visited의 배열이 1보다 큰 값으로 놓고 랜덤으로 지정해준다. 같은 방법으로 3개의 함정은 visited의 배열값이 1일 때 랜덤으로 지정해준다.

마지막으로, 앞서 잠깐 설명했던 시간차이를 이용해 이미지를 그리기 위해서 변수 endTime 에 (unsigned)time(NULL)을 할당한다. 이후, endTime += 3을 통해서 3초의 차이를 둔다. 좀비의 위치역시 설정하는데, 미로의 좌측 가장 하단으로 고정해놓았다.

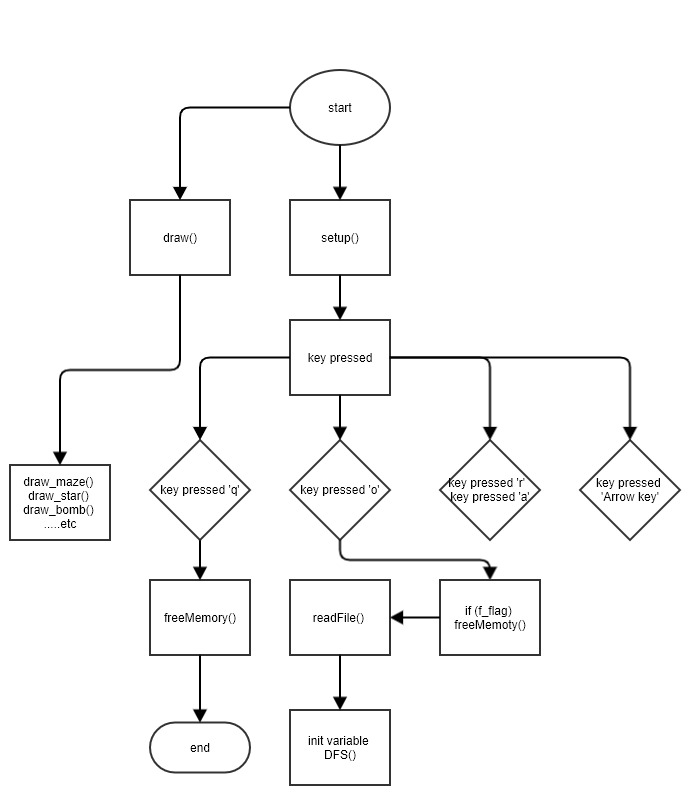
freeMemory()함수는 프로그램이 종료되기 직전과, 이미 파일을 한번이상 열었을 때 또 다른 파일을 여려할 때 호출되어 메모리를 해제시켜준다. 프로젝트 전체로 봤을 때, 동적메모리를 사용한 경우는 input과 visit 배열밖에 없다.

DFS() 함수는 실습때와 마찬가지로 stack과 iterative를 이용해 목적지까지의 경로를 visited배열에 저장한다.

draw\_zombie() 함수는 애니메이션 처럼 좀비가 걸어다니도록 구현하였다. 이는 오픈프레임워크의 특징을 이용한 것인데, 오픈프레임워크는 프레임레이트를 정할 수 있다. 흔히 말하는 FPS인데, 초당 화면 갱신을 몇번하는 지 의미하며 당연하게도 높을수록 연속된 동작으로 보이며, 부드러워진다. 이를 이용해, draw가 호출될 때 마다, 좀비의 x좌표는 0.1씩 움직인다. 또한, 벽에 부딪히면 다시 되돌아오도록 설정하여 애니메이션처럼 좀비가 돌아다니게 표현이 가능했다.

**4. 순서도 및 시간,공간 복잡도**

이 프로젝트의 대략적인 순서도는 다음과 같다.

****

보통 인접 리스트로 구현한 DFS의 경우 O(V+E)로 나타낸다. 하지만, 이 프로젝트의 DFS()함수 구현시 사용했던 미로에서 통로 역시 방처럼 생각했기 때문에 정점의 개수가 빈방의 개수가 아닌 그 정점의 개수 그 자체로 V+E가 되었다. 더불어, 늘어난 정점마다 다시 간선들이 존재하는 것 처럼 이동할 수 있는지 여부를 확인해주었다.

하지만, 시간복잡도는 매 시행마다 다르게 되는데 이는 목표지점을 만나는 경우 일반 DFS와는 다르게 바로 탈출을 하기 때문이다. 물론 최악의 경우에는 늘어난 정점과 늘어난 간선의 개수의 합이 되고 O(V+E)의 형태가 된다.

공간복잡도의 경우에는 기본적으로 visit배열의 크기인 O(HEIGHT\*WIDTH)와 stack에서 길의 경로를 저장한 만큼이 되게된다. Stack의 크기는 매 시행마다 다르고 예측할 수 없지만, 가장 작을 경우 O(HEIGHT + WIDTH) 만큼이 된다.

DFS함수를 제외하면 나머지 함수들은 대부분 2중 for문으로 되어있어 O(HEIGHT \*WIDTH)의 시간복잡도를 가지거나, 다중 if문으로 되어있다. 간간히, s\_flag나 b\_flag 검사를 위한 포문이 존재하지만, 사실상 3번만 반복하므로, O(1)이라고 생각할 수 있다.

readFile()에서는 while문 안에 2중포문이 존재하지만, 이 역시 대부분이 2중 루프를 돌면서 while문의 조건을 만족하게 되므로 2중포문으로 볼 수 있다.

최종적으로 이 프로젝트의 시간복잡도는 DFS의 시간복잡도에 따르거나, 최대 2중포문을 따르게 되어 O(HEIGHT\*WIDTH) 가 된다. 그러나, 당연하게도 DFS안에 이미 2중포문이 존재하므로 DFS의 시간복잡도를 따른다고 해도 무방하다. 늘어난 정점과 간선의 개수의 합인O(V+E)가 되며, 공간복잡도의 경우 마찬가지로 input과 visit 배열을 제외하면, 대부분이 일반 변수이며 데이터타입이 integer이며 크기가 3인 배열을 총 6개, 크기가 4인 배열을 총 2개, 그리고 visit과 input 각각의 배열의 크기를 고려했을 때 O(2\*HEIGHT\*WIDTH + 24) 정도가 된다.

**5. 창의적 구현**

창의적으로 구현했다고 생각한 부분은 time 라이브러리의 시간함수를 이용해 시간차를 두고 이미지가 사라지는 애니메이션을 구상한 것이다. 이 차이를 이용해, 함정의 위치를 3초간 보여주고 사라지게함으로서 게임의 재미를 조금 더 증가시켰다.

endTime = (unsigned)time(NULL);

endTime += 3;

void ofApp::draw\_bomb()

{

startTime = (unsigned)time(NULL);

for (int i = 0; i < 3; i++) {

if (b\_flag[i]) {

ofSetColor(255, 255, 255);

bomb.draw(bomb\_x[i] - 10, bomb\_y[i] - 10, MAX, MAX);

}

}

if (endTime - startTime == 0) {

b\_flag[0] = 0;

b\_flag[1] = 0;

b\_flag[2] = 0;

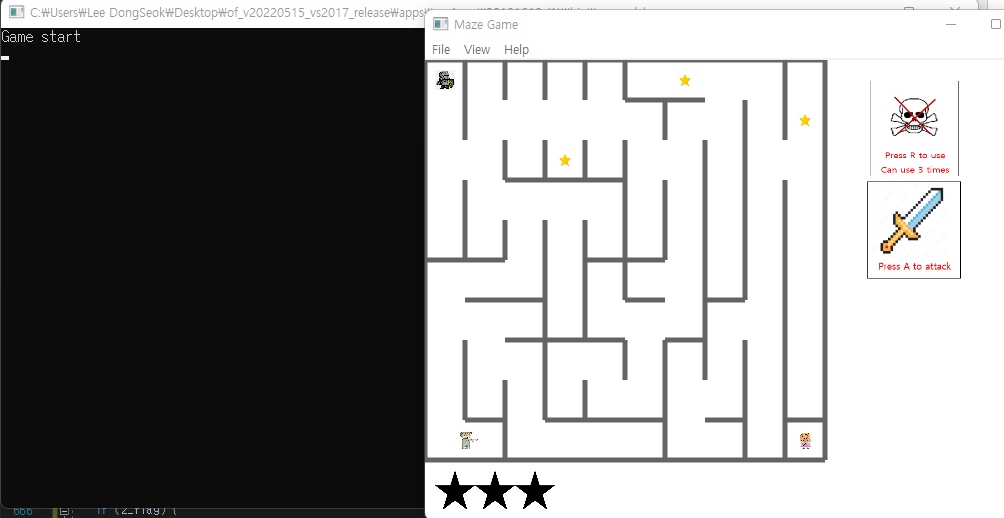
}

}

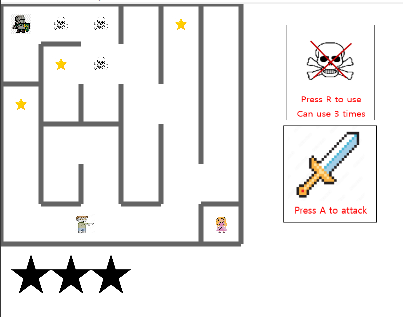
두번째는, 앞서 말했듯이 좀비 캐릭터의 애니메이션 움직임이다. 오픈프레임워크에서는 프로그램이 실행되면 끊임없이 draw 함수를 프레임에 맞게 호출하게 된다. 이때, 자동적으로 x값이 변하면서 움직이는 것의 상대적 속도차이를 이용하면, 좀비는 좀더 천천히 움직이고 결국 시간에 따라 애니메이션처럼 모션을 취하는 것으로 보인다. 변화를 주는 x의 값이 너무 크면, 좀비의 움직임이 너무 빠르게 변하여 우리의 눈에는 순간이동을 한 것처럼 변하기 때문에, 애니메이션 움직임을 위해서 x변화량을 작게해야한다.

마지막은, 게임의 진행 방식이다. 게임은 기본적으로 3개의 함정이 게임 시작 후 보였다가 3초 후 사라지게 된다. 이때, 플레이어는 함정 바로앞에서 ‘r’키를 통해 제거 할 수 있다. 하지만, 스킬의 사용은 3번으로 제한되므로 순간 기억력이 요구된다. 이후, 플레이어가 3개의 별을 먹게 되면, 숨겨져있던 상자의 위치가 나오게 된다. 상자는 좀비를 ‘a’키로 죽였을 때 나오는 열쇠를 이용해 열 수 있다. 상자를 열게 되면 역시 숨겨져있던 레버가 나오게 되고 레버의 바로 앞으로 가게 되면, 레버가 움직이고 공주를 가두고 있던 벽 하나가 사라진다. 당연하게도, 함정을 밟거나, 좀비와 몸이 닿게되면 게임은 종료된다.

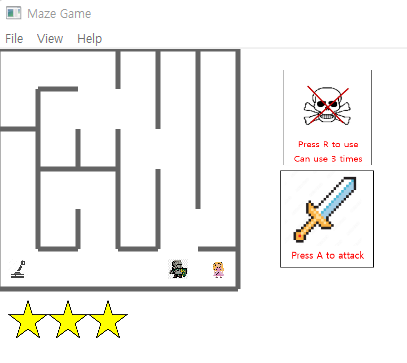
**6. 프로젝트 실행 결과**

****

시간이 조금 지나 함정이 사라진 모습이다.

****

함정의 모습

****

클리어 직전

**7. 느낀 점 및 개선 사항**

본 프로젝트를 진행하면서, 획기적인 알고리즘이나 자료구조를 막 사용하지는 않았다. 하지만, 오픈프레임워크라는 툴을 이용하여 픽셀 게임을 만들어보면서 재미있었고 뿌듯했다. 물론, 아직 실력이 많이 부족해서 난잡하고, 어리숙하다. 하지만, 오늘 해본 경험은 다음에 있을 프로젝트에 있어서 도움이 될 것이라는건 믿어 의심치 않는다. 특히나, 오픈프레임워크에서 이미지를 불러 그리고 애니메이션처럼 움직이게 하는 것이 신기했다. 오픈프레임 워크는 gif 파일도 그릴 수 있는 것으로 아는데, 모션을 이용한 도트게임을 만들 수 있을 것 같다.

진행하면서 느낀 아쉬운점은 무료로 풀려 있는 도트이미지들은 매우 적어서 마음대로 사용하지 못한다는 점이었다. 또한, 아직 오픈프레임워크의 다른 기능들을 사용해보지 않았기에 조금 더 나은 프로젝트를 만들 수 있지 않았을까 하는 아쉬움이 있다.

개선해야할 사항들은 사실 너무나도 많다. 우선, 옅은 자료구조 지식으로 조금 더 나은 DFS를 구현할 수 있지 않았을까 하는 아쉬움이 있다. 게다가, 도트 이미지의 단적인 움직임밖에 표현하지 못해서 너무 단조로웠다. 또한, 조금 더 나아가 좀비가 플레이러의 위치를 따라오게 만들거나, 좀비를 더 만든다던지 등 유동적으로 해야하는 부분에 있어서 너무 수동적인 부분이 너무 많았다고 생각한다.