12주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

프로그램 구현을 위한 자료구조로 2차원 vector를 사용했다. maze는 전역 변수로 ofApp.h에 선언되어 있다. 자료구조를 화면에 그리기 위해 우선 readFile 함수에서 .maz 파일의 data를 maze 변수에 저장하고, 다음으로 draw 함수를 호출하여 index로 maze 각 원소에 접근해 미로를 그린다. 마지막으로 freeMemory 함수에서는 maze 변수에 할당된 메모리를 해제한다.

*(구현 코드)*

vector<vector<char>> maze;

1) readFile 함수

이 함수에서는 주어진 .maz 파일을 읽어와서 자료구조에 저장하는 부분을 구현했다. 미로 파일의 내용을 읽고, 가로와 세로의 크기를 구한 후, 그 크기에 맞게 자료구조를 초기화하고 미로 데이터를 저장하는 순서를 거친다.

먼저, ifstream 클래스를 사용하여 .maz 파일을 연다. fp.open(filePath)에서 filePath는 읽어올 .maz 파일의 경로이다. 파일에서 한 줄씩 읽어와서 미로의 가로 크기(WIDTH)와 세로 크기(HEIGHT)를 구한다. getline(fp, line) 함수를 사용하여 파일에서 한 줄을 읽어온다. 읽어온 줄의 길이를 line.length()로 구하여 가로의 크기를 업데이트하고, HEIGHT 변수를 증가시킨다. 이 과정을 파일의 끝까지 반복하면, 가로와 세로의 크기가 결정된다. 다음으로, 파일을 다시 열고, 미로 데이터를 저장할 벡터의 크기를 HEIGHT와 WIDTH로 설정하여 초기화한다. 파일을 읽으면서 미로 데이터를 input과 maze에 저장한다. getline(fp, line) 함수를 사용하여 파일에서 한 줄을 읽어온다. 그리고 for 루프를 통해 maze[idx][i]에 line[i] 값을 저장한다. 이렇게 파일의 모든 줄을 읽으면서 미로 데이터를 저장하는 과정을 반복한다. 마지막으로, 파일을 닫고 draw 함수를 호출한다.

*(TODO 구현 부분)*

bool ofApp::readFile() {

ifstream fp;

fp.open(filePath);

string line;

while (getline(fp, line)) {

WIDTH = line.length();

HEIGHT++;

}

fp.close();

fp.open(filePath);

maze.resize(HEIGHT, vector<char>(WIDTH));

while (getline(fp, line)) {

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

maze[idx][i] = line[i];

}

idx++;

}

fp.close();

this->draw();

}

2) draw 함수

이 함수에서는 주어진 자료구조를 기반으로 미로를 그리는 부분을 작성했다.

변수 및 상수 초기화:

* rectLong은 사각형의 가로 길이를 나타내는 변수이다.
* rectShort는 사각형의 세로 길이를 나타내는 변수이다.
* x와 y는 현재 사각형을 그릴 좌표를 나타내는 변수이다.

미로 그리기:

* 두 개의 반복문을 사용하여 미로의 각 위치를 순회한다.
* 외부 반복문은 HEIGHT만큼 반복하며, 내부 반복문은 WIDTH만큼 반복한다.
* 각 위치에 해당하는 좌표 x와 y를 계산한다.
* maze의 해당 위치에 저장된 값을 확인하여 적절한 사각형을 그린다.

maze[i][j]가 '+'인 경우, 현재 위치에 정사각형을 그린다.

(ofDrawRectangle(x, y, rectShort, rectShort))

maze[i][j]가 '-'인 경우, 현재 위치에 가로 방향 사각형을 그린다.

(ofDrawRectangle(x, y, rectLong, rectShort))

maze[i][j]가 '|'인 경우, 현재 위치에 세로 방향 사각형을 그린다.

(ofDrawRectangle(x, y, rectShort, rectLong))

*(TODO 구현 부분)*

void ofApp::draw() {

char str[256];

ofSetColor(100);

ofSetLineWidth(5);

int i, j;

double x = 0, y = 0;

int rectLong = 30;

int rectShort = 3;

for (i = 0; i < HEIGHT; i++) {

y = rectLong \* (double)(i / 2) + rectShort \* (double)(i - (i / 2));

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

x = rectLong \* (double)(j / 2) + rectShort \* (double)(j - (j / 2));

if (maze[i][j] == '+') {

ofDrawRectangle(x, y, rectShort, rectShort);

}

else if (maze[i][j] == '|') {

ofDrawRectangle(x, y, rectShort, rectLong);

}

else if (maze[i][j] == '-') {

ofDrawRectangle(x, y, rectLong, rectShort);

}

}

}

}

3) freeMemory 함수

이 함수에서는 vector 변수에 할당된 메모리를 해제하는 코드를 작성했다.

*(TODO 구현 부분)*

void ofApp::freeMemory() {

maze.clear();

}

4) 시간 및 공간 복잡도

4-1) Readfile 함수

시간 복잡도: 파일을 한 줄씩 읽어오는 과정에서 HEIGHT를 계산하고, 각 줄의 길이를 확인하여 WIDTH를 계산하는 데 O(N)의 시간이 소요된다. 여기서 N은 파일의 줄 수이다. 또한, 파일을 다시 한 줄씩 읽어오면서 input 배열과 maze 벡터에 데이터를 저장하는 과정도 O(N)의 시간이 소요된다.

공간 복잡도: maze 벡터는 HEIGHT \* WIDTH 크기의 2차원 벡터를 사용하므로 O(HEIGHT \* WIDTH)이다.

4-2) draw 함수

시간 복잡도: 외부 반복문은 HEIGHT번, 내부 반복문은 WIDTH번 실행되기 때문에 전체적인 시간 복잡도는 O(HEIGHT \* WIDTH)이다.

공간 복잡도: maze 배열의 크기에 비례한다. maze 배열은 HEIGHT행과 WIDTH열로 구성되어 있으므로, 공간 복잡도는 O(HEIGHT \* WIDTH)이다.

4-3) 그래픽 전환 작업의 총 시간 및 공간 복잡도

시간 복잡도: O(HEIGHT \* WIDTH)

공간 복잡도: O(HEIGHT \* WIDTH)

실험 전에는 템플릿 폴더의 ofApp.h 파일에 2차원 character 배열인 input 변수가 있어 이를 이용하려고 했지만, c++ 언어에 관해 여러 정보를 검색 하던 중 vector라는 class가 있다는 것을 알게 되어 이를 이용하게 되었다. vector 변수를 이용하는 것이 member function을 이용한다는 점에서 array보다 다루기가 편리해 이 방법을 택하게 되었다. 또한 주어진 data 파일 이름 형식이 (미로의 가로)\_(미로의 세로).maz인 것을 보고 파일 이름에서 미로의 가로와 세로 크기를 따오려고 했으나 maze0 파일을 뒤늦게 발견해 직접 파일의 크기를 재서 미로의 가로와 세로 크기를 구하는 방법으로 바꾸었다.

**2.**

이전까지 c언어로 코드를 작성하는 것만 익숙했는데 저번 waterfall 프로젝트와 이번 maze 프로젝트를 거치며 점점 c++ 언어로 코드를 구현하는 방식을 배워가고 있는 것 같다. 예를 들면 fgets 함수 대신 getline 함수를 사용한다는 점이나 fopen 함수 대신 (파일 포인터).open(파일 경로) 함수를 사용한다든지 각 클래스 멤버 함수를 사용하는 것에 익숙해지는 중이다. 또한 vector 클래스의 사용 방법과 관련 member function등에 대해서도 알게 되었다. 마지막으로 draw 함수 구현 부분에서 waterfall 프로젝트에서 한 번 배웠던 도형 그리기 함수들을 복습할 수 있어 좋았다.