12주차 예비보고서

전공 : 컴퓨터공학과 학년 : 2학년 학번 : 20211558 이름 : 윤준서

**1. 함수들의 Pseudo Code**

**- readFile() -**

freeMemory()

WIDTH = 0, HEIGHT = 0

ofFileDialogResult openFileResult = ofSystemLoadDialog("Select a maze file")

if(file open is true)

location = openFileResult.getPath()

ifstream input\_file(location)

*//파일이 성공적으로 열린 경우*

if (input\_file.is\_open()) {

*//각 줄을 읽어들여 미로의 너비와 높이를 계산한다.*

string line;

while (getline(input\_file, line)) {

WIDTH = max(WIDTH, static\_cast<int>(line.length()))

++HEIGHT }

*// 미로 데이터를 저장할 벡터 생성*

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

*// 파일을 다시 열어서 미로 데이터를 벡터에 저장한다.*

input\_file.clear()

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

getline(input\_file, line)

for (int j = 0; j < line.length(); ++j)

maze[i][j] = line[j];}

*// 라인 객체를 생성하여 저장한다.*

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

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

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

LINE.push\_back({j - 0.5, i + 0.5, j + 1.5, i + 0.5});

} else if (maze[i][j] == '|') LINE.push\_back({j + 0.5, i - 0.5, j + 0.5, i + 1.5}); }}

*// 미로 읽기 성공 시*

cout << "Maze successfully loaded. Width: " << WIDTH << ", Height: " << HEIGHT << endl;

return true; }

else { *// 파일 열기 실패 시*

cout << "Failed to open the file." << endl;

draw\_flag = 0;

return false; }

else { *// 사용자가 파일 선택을 취소한 경우*

cout << "File selection canceled." << endl;

return false; }

**- freeMemory() -**

maze, LINE, visit, maze\_block clear()

m\_block, m\_path, a\_path .pop()

draw\_flag, dfs\_flag, bfs\_flag = 0

path, all\_path .clear()

*//미로와 선, 방문 유무, 블록, 길, 그림 여부, 탐색 여부 등의 모든 저장된 변수들을 해제한다.*

**- draw () -**

if(this->draw\_flag == 1) *//파일을 불러온 경우 진행*

color 설정

for (int i = 0; i < this->LINE.size(); i++) maze\_line(this->LINE[i]) *//미로 그리기*

if(this->dfs\_flag == 1) { *//"d" 키를 누른 경우*

color 설정

size\_all = a\_path.size();

temp\_stack = a\_path;

while(!temp\_stack.empty()) {

all\_path.addVertex(20 \* temp\_stack.top().x + 10, 20 \* temp\_stack.top().y + 10);

temp\_stack.pop() }

all\_path.draw(); *//(1,1)이 시작, (HEIGHT, WIDTH) 이 끝인 길을 그린다.*

int size = m\_block.size();

temp\_stack = m\_block;

while(!temp\_stack.empty()) {

path.addVertex(20 \* temp\_stack.top().x + 10, 20 \* temp\_stack.top().y + 10);

temp\_stack.pop() }

path.draw(); *//연결한 길을 최단 거리로 추가로 그린다.*

}