컴실1 기말 프로젝트

Teleportation maze

20201585 박준하

1. 프로젝트 목표

이번 프로젝트는 기존 실습 때 진행하였던 미로 생성 알고리즘 중 하나인 Eller’s algorithm을 변형하여 teleportation platform이 존재하는 미로 생성 알고리즘을 제작하는 것을 목표로 하였다. 구체적으로, 미로 내에 여러 쌍의 teleportation platform이 존재해 한 platform 위에 올라서면 그에 대응하는 다른 platform으로 순간이동하는 perfect maze를 생성하는 알고리즘이 목표이다.

나아가, Eller’s algorithm과 동일하게 변형한 알고리즘 또한 perfect maze의 성질을 가지기에 DFS로 미로의 경로를 탐색하는 기능을 구현하는 것 또한 추가 목표로 삼았다.

2. 자료구조 및 알고리즘에 대한 설명

프로그램을 실행시키고 메뉴에서 Make를 선택하면 미로가 생성됨과 동시에 화면에 그려진다.

미로가 생성되는 알고리즘은 makeTeleMaze() 함수에서 시작된다. 이 함수는 먼저 미로의 높이와 너비를 입력받은 후 미로 생성과 관련된 다양한 자료구조를 초기화하는 initializeMaze() 함수를 호출한다. 그리고 미로의 마지막 행에 도달하기 전까지 미로의 첫번째 행부터 각 행마다 일련의 과정을 수행한다.

Eller’s algorithm의 경우, 과정은 크게 3단계로 이루어진다.

1. 미로의 번호가 부여되지 않은 방마다 각각 새로운 집합의 번호를 부여한다.

2. 방의 번호가 오른쪽 방의 번호와 다르다면 확률적으로 두 방 사이의 벽을 허문다.

이때, 두 방이 각각 속한 두 집합을 하나로 합친다.

만약 방의 번호가 같을 경우 허물지 않는다.

3. 각 방의 아래쪽 벽을 확률적으로 허문다. 허문 벽의 아래쪽 방은 그 위쪽 방의 집합에 속하도록 한다.

이때, 한 집합에 속한 방이 모두 아래쪽 벽이 존재한다면 그 집합 중 하나의 방을 랜덤하게 선택해 아래쪽 벽을 허문다.

Teleportation platform이 존재하도록 하는, 변형한 알고리즘의 경우 이 과정이 달라진다.

변형된 Eller’s algorithm의 경우, 과정은 크게 5단계로 이루어진다.

1. 미로의 번호가 부여되지 않은 방마다 각각 새로운 집합을 부여한다.

2. 방의 번호가 오른쪽 방의 번호와 다르다면 확률적으로 두 방 사이의 벽을 허문다.

이때, 두 방이 각각 속한 두 집합을 하나로 합치고 두 집합에 속한 텔레포트 발판 후보군 역시 하나의 집합으로 합친다.

만약 방의 번호가 같을 경우 허물지 않는다.

3. 각 방의 아래쪽 벽을 확률적으로 허문다. 허문 벽의 아래쪽 방은 그 위쪽 방의 집합에 속하도록 한다.

이때, 한 집합에 속한 방이 모두 아래쪽 벽이 존재한다면 그 집합 중 하나의 방을 랜덤하게 선택해 아래쪽 벽을 허문다.

4. 만약 방이 3개의 벽으로 둘러싸여 있다면 그 좌표를 텔레포트 발판 후보군 집합에 추가시킨다.

5. 서로 다른 집합에서 두개의 텔레포트 발판 후보를 선택한 뒤, 이를 활성화시킨다.

이때, 두 텔레포트 발판이 속한 두 후보군 집합과 방의 집합을 합친다.

일반적인 Eller’s algorithm과 달라진 점은 2번 과정에서 오른쪽 벽을 없앴을 때 방의 집합이 합쳐짐에 따라 후보군 또한 하나의 집합으로 합쳐지는 과정이 추가되었다는 점, 그리고 텔레포트 발판의 후보를 생성하고 두 후보를 활성화시키는 4, 5번 과정이 추가되었다는 점이다.

이러한 미로 생성 알고리즘을 통해 미로를 생성하고 나면 미로를 DFS를 통해 탐색할 수 있는 기능 또한 존재한다. 이는 DFS과정 중 미로의 방의 상하좌우와 더불어 teleport platform을 통해 갈 수 있는 또 하나의 길을 추가로 고려함으로써 구현하였다.

미로 생성 알고리즘의 대략적인 플로우 차트와 4, 5번 과정의 플로우 차트는 아래와 같다.

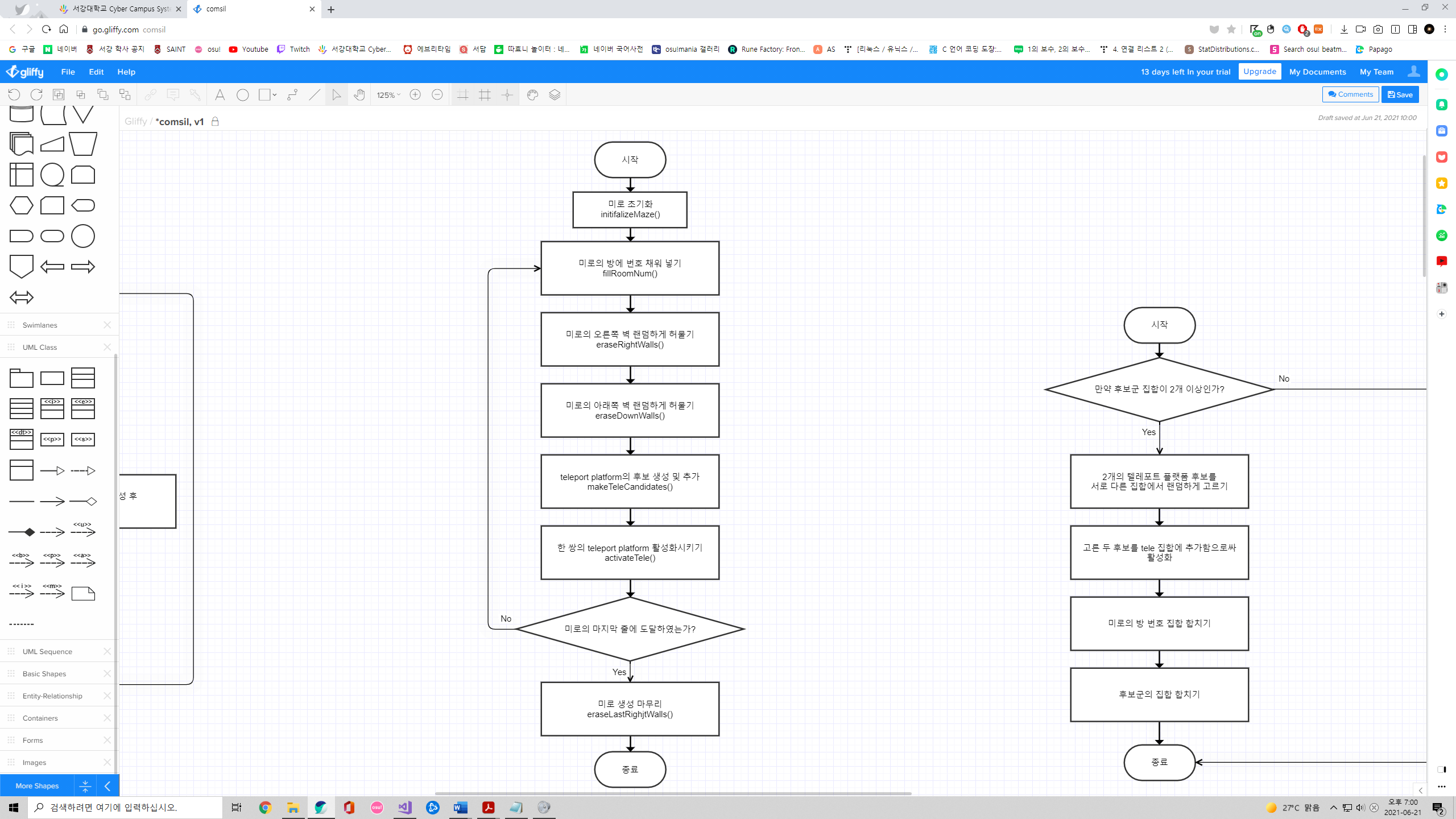


그림 1 – 대략적인 flow chart

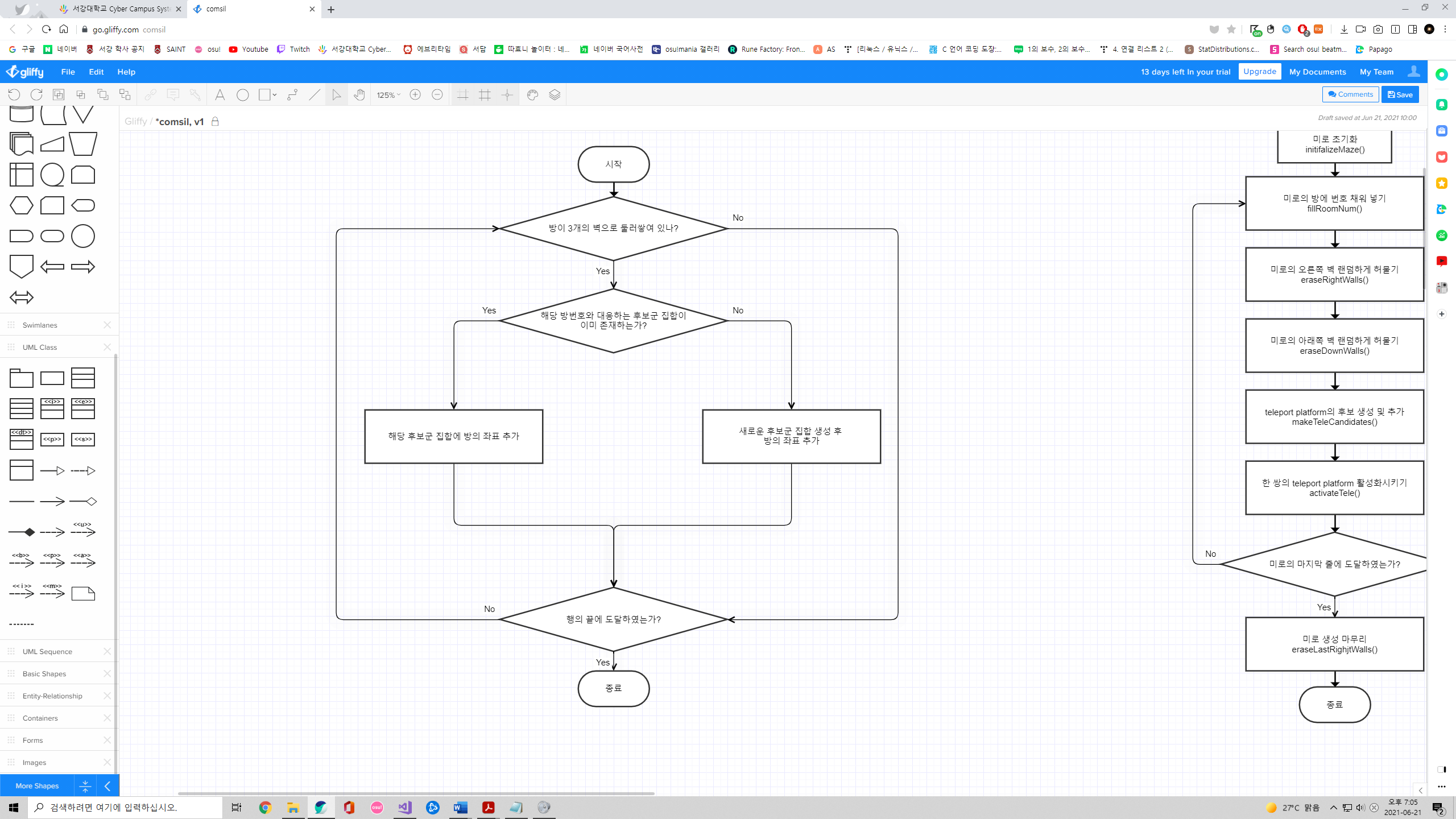


그림 2 – makeTeleCandidates() (4번 과정)의 flow chart

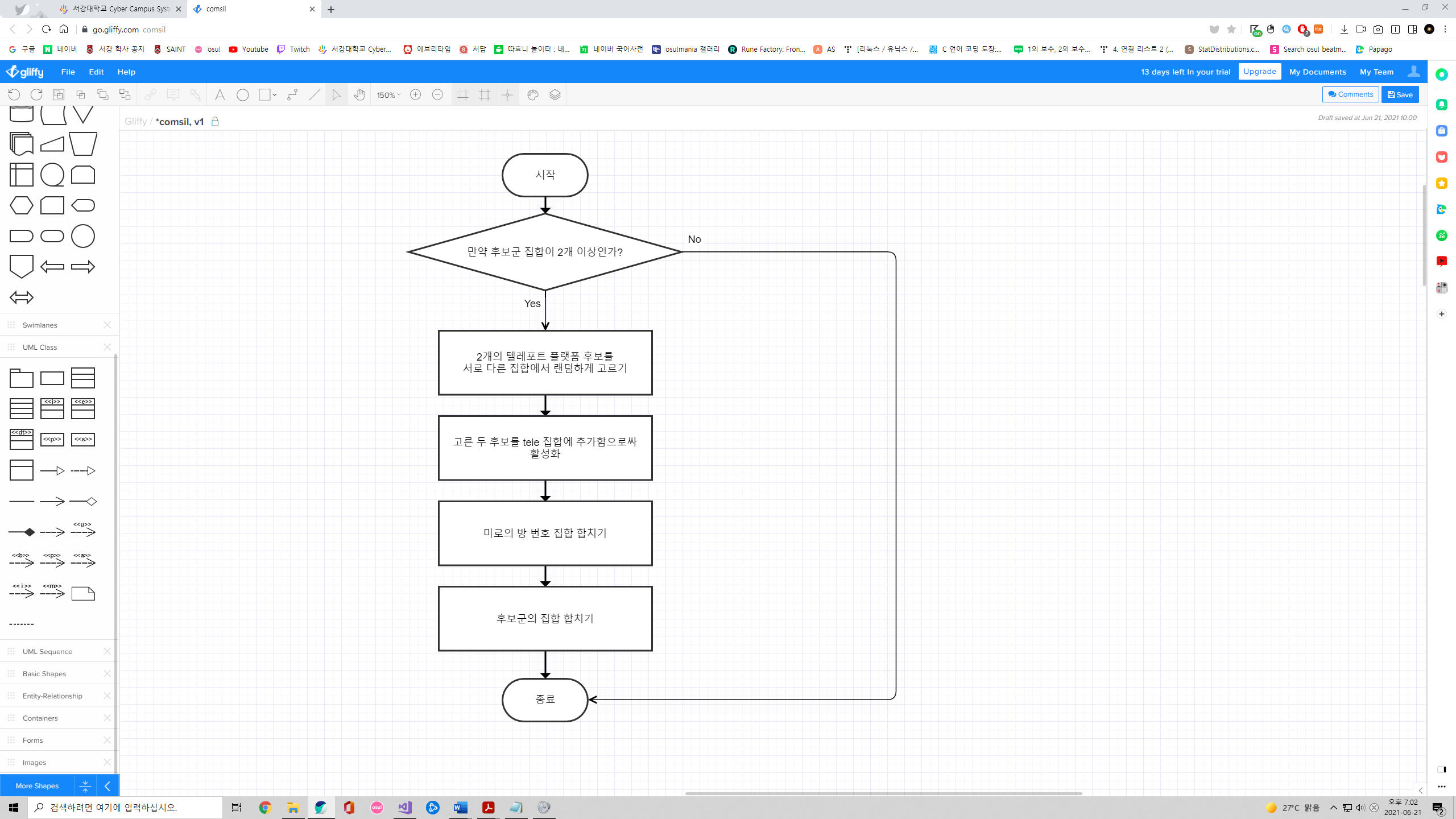


그림 3 – activateTele() (5번 과정)의 flow chart

이러한 알고리즘을 구현하기 위해 사용한 자료구조는 다음과 같다.

|  |  |
| --- | --- |
| MazeRoom \*\*maze | 미로의 벽 정보와 setNumber를 저장하기 위한 2차원 구조체 배열 |
| TeleRoom \*\*tele | 미로에 존재하는 teleportation platform의 정보를 저장하기 위한 2차원 구조체 배열 |
| Vector <TeleCandidateSets> candidateSets | teleportation platform의 후보들을 setNumber에 따라 저장하기 위한 구조체 백터 |
| Vector <Coordiate> s | Iterative DFS를 위한 스택 |

위 표에서 언급된 구조체는 다음과 같이 정의되어 있다.

typedef struct \_Coordinate {

int y; //방의 y좌표

int x; //방의 x좌표

} Coordinate;

typedef struct \_MazeRoom {

int roomNum; //방이 속한 집합의 번호

bool visited; //DFS 중 방문하였는가?

bool hasRightWall; //오른쪽 벽이 있는가?

bool hasDownWall; //아래 벽이 있는가?

} MazeRoom;

typedef struct \_TeleRoom {

bool isTele; //teleport platform인가?

int teleNum; //teleport platform의 번호

int gotoX; //teleport 되는 x좌표

int gotoY; //teleport 되는 y좌표

} TeleRoom;

typedef struct \_TeleCandidateSets {

int setNum; //후보군이 속한 집합의 번호

vector <pair<int, int>> coordinates; // 해당 집합에 속한 teleport platform의 후보군

} TeleCandidateSets;

3. 핵심 변수에 대한 설명

|  |  |
| --- | --- |
| MazeRoom \*\*maze | 미로의 벽 정보와 setNumber를 저장하기 위한 2차원 구조체 배열 |
| TeleRoom \*\*tele | 미로에 존재하는 teleportation platform의 정보를 저장하기 위한 2차원 구조체 배열 |
| Vector <TeleCandidateSets> candidateSets | teleportation platform의 후보들을 setNumber에 따라 저장하기 위한 구조체 백터 |
| Vector <Coordiate> s | Iterative DFS를 위한 스택 |
| Int height | 미로의 높이 |
| Int width | 미로의 너비 |
| Int lastRoomN | Maze 배열의 cell에서 가장 최근에 추가한 room의 번호 |
| Int lastTeleN | Tele 배열의 cell에서 가장 최근에 추가한 teleport platform의 번호 |
| Int isOpen | 미로가 생성되었는지 판단하는 변수 |
| Int isDFS | DFS함수를 실행시켰는지 판단하는 변수 |
| Int isFirst | 처음으로 미로를 생성한 것인지 판단하는 변수 |

4. 각 함수에 대한 설명

|  |  |
| --- | --- |
| Void setup(); | 프로그램이 시작된 후 전역 변수 초기화 및  윈도우 설정 |
| Void draw(); | 미로가 생성되었을 경우, 미로를 그린다. |
| Void freeMemory(); | 미로 생성 알고리즘 중 사용되었던 메모리를 해제시킨다. |
| Bool DFS(); | 미로가 생성되었을 경우, DFS를 진행한다. |
| Void DFSdraw(); | DFS가 성공적으로 이루어졌을 경우, DFS 과정과 결과를 화면에 그린다. |
| Void printMaze(); | 미로를 출력하고 maze.maz 파일로 내보낸다. |

|  |  |
| --- | --- |
| Void makeTeleMaze(); | 미로 생성 알고리즘을 시작하는 함수. 아래 함수들을 여러 번 호출한다. |
| Void initializeMaze(); | 알고리즘과 관련된 변수들을 초기화한다. |
| void fillRoomNum(int rowN); | rowN번째 행의 초기화되어있지 않은 room들을 초기화시킨다. |
| Void eraseRightWalls(int rowN); | rowN번째 행의 오른쪽 벽을 확률에 따라 허물고 이에 따라 room과 teleport platform 후보의 집합을 갱신한다. |
| void eraseDownWalls(int rowN); | rowN번째 행의 아래 벽을 확률에 따라 허문다. |
| void makeTeleCandidates(int rowN); | rowN번째 행의 teleport platform 후보들을 탐색하고 이를 집합에 추가한다. |
| void activateTele(int rowN); | 한 쌍의 Teleport platform을 활성화시키고 이에 따라 room과 teleport platform 후보의 집합을 갱신한다. |
| void eraseLastRightWalls(); | 미로의 마지막 행에서 오른쪽 벽을 규칙에 따라 허문다. |

5. 프로젝트 실행 화면

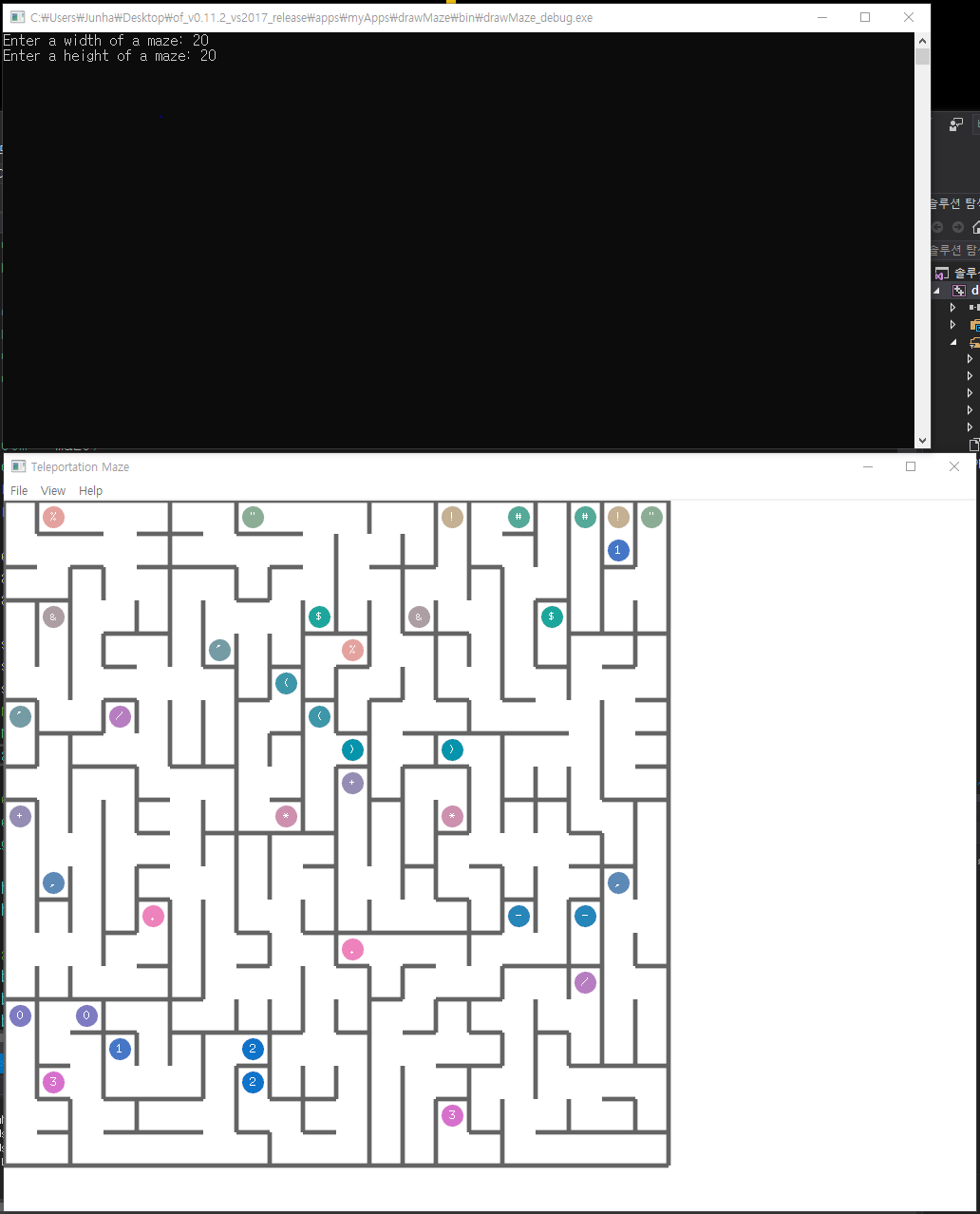


그림 4 – 20x20 미로를 생성한 모습

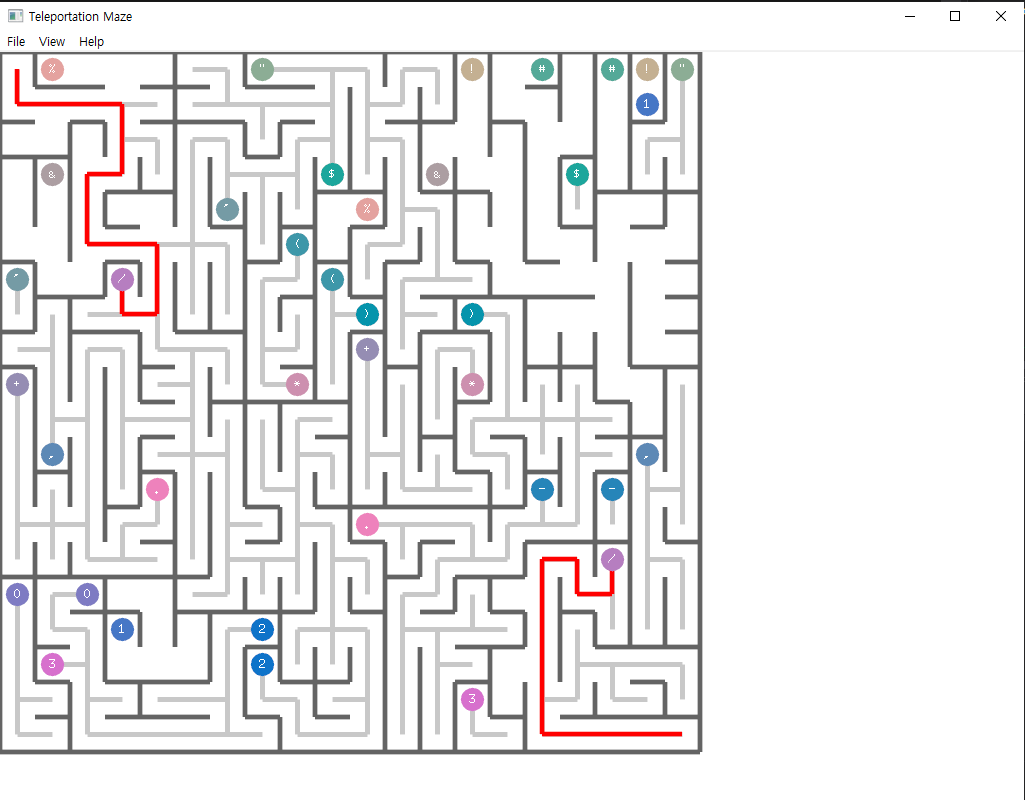


그림 5 – 생성한 미로에서 DFS를 진행한 모습

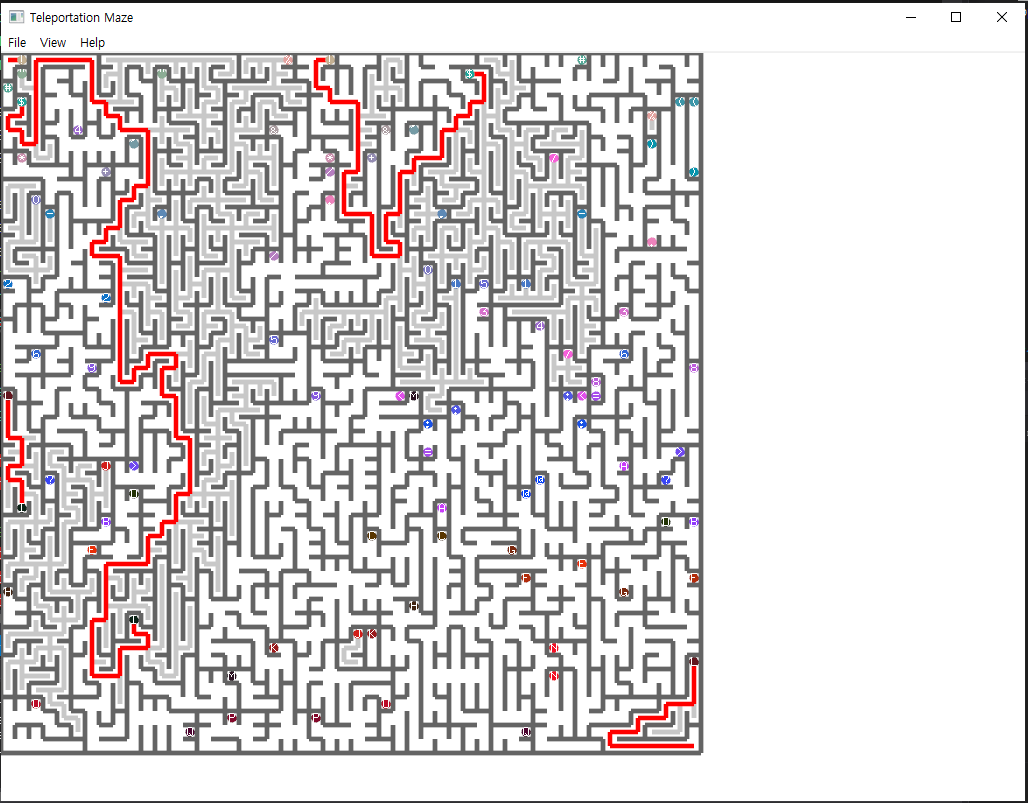


그림 6 – 50x50 미로 생성 후, DFS를 진행한 모습

6. 느낀 점 및 개선사항

미로를 여러 개의 set으로 나누어 생성하는 eller’s algorithm의 특성을 활용하여 teleport platform이 존재하는 미로 생성 알고리즘을 만드는 과정에서 eller’s algoirthm의 유용성을 실감할 수 있었다. 또한 일반적인 과제와 다른 이번 프로젝트를 통해 주도적인 프로그래밍 습관을 기를 수 있게 되어 유익했다.

프로그램의 개선사항으로는 BFS 기능의 추가, 미로의 set별로 색을 다르게 표현해 알고리즘의 기능을 눈으로 볼 수 있게 하는 기능의 추가가 있다.

7. 전체 소스코드

Main.cpp

#include "ofMain.h"

#include "ofApp.h"

// for default console

//========================================================================

int main() {

//

// for window without console

//========================================================================

// int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd) {

ofSetupOpenGL(1024, 768, OF\_WINDOW);// <-------- setup the GL context

// this kicks off the running of my app

// can be OF\_WINDOW or OF\_FULLSCREEN

// pass in width and height too:

ofRunApp( new ofApp());

}

ofApp.h

#pragma once

#include "ofMain.h"

#include "ofxWinMenu.h" // Menu addon

class ofApp : public ofBaseApp {

public:

void setup();

void update();

void draw();

void keyPressed(int key); // Traps escape key if exit disabled

void keyReleased(int key);

void mouseMoved(int x, int y);

void mouseDragged(int x, int y, int button);

void mousePressed(int x, int y, int button);

void mouseReleased(int x, int y, int button);

void windowResized(int w, int h);

void dragEvent(ofDragInfo dragInfo);

void gotMessage(ofMessage msg);

void freeMemory();

bool DFS();

void DFSdraw();

void makeTeleMaze();

void initializeMaze();

void eraseRightWalls(int rowN);

void eraseDownWalls(int rowN);

void makeTeleCandidates(int rowN);

void activateTele(int rowN);

void fillRoomNum(int rowN);

void eraseLastRightWalls();

void printMaze();

typedef struct \_Coordinate {

int y; //방의 y좌표

int x; //방의 x좌표

} Coordinate;

typedef struct \_Node {

int data;

struct \_Node\* next;

} Node;

typedef struct \_MazeRoom {

int roomNum; //방이 속한 집합의 번호

bool visited; //DFS 중 방문하였는가?

bool hasRightWall; //오른쪽 벽이 있는가?

bool hasDownWall; //아래 벽이 있는가?

} MazeRoom;

typedef struct \_TeleRoom {

bool isTele; //teleport platform인가?

int teleNum; //teleport platform의 번호

int gotoX; //teleport 되는 x좌표

int gotoY; //teleport 되는 y좌표

} TeleRoom;

typedef struct \_RoomNumSet {

int roomNum;

int elementN;

unsigned char connected;

struct \_RoomNumSet \*nextSet;

Node \*index;

} RoomNumSet;

typedef struct \_TeleCandidateSets {

int setNum; //후보군이 속한 집합의 번호

vector <pair<int, int>> coordinates; // 해당 집합에 속한 teleport platform의 후보군

} TeleCandidateSets;

MazeRoom \*\*maze;

TeleRoom \*\*tele;

vector <TeleCandidateSets> candidateSets;

vector <Coordinate> s;

int height, width;

int lastRoomN;

int lastTeleN;

int isOpen; //미로가 생성되었는지 판단하는 변수. 0이면 생성되지 않았고 1이면 생성되었다.

int isDFS; //DFS함수를 실행시켰는지 판단하는 변수. 0이면 실행 안했고 1이면 실행했다.

int isFirst;

// Menu

ofxWinMenu \* menu; // Menu object

void appMenuFunction(string title, bool bChecked); // Menu return function

// Used by example app

ofTrueTypeFont myFont;

ofImage myImage;

float windowWidth, windowHeight;

HWND hWnd; // Application window

HWND hWndForeground; // current foreground window

// Example menu variables

bool bShowInfo;

bool bFullscreen;

bool bTopmost;

// Example functions

void doFullScreen(bool bFull);

void doTopmost(bool bTop);

};

ofApp.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "ofApp.h"

#include <iostream>

using namespace std;

//--------------------------------------------------------------

void ofApp::setup() {

ofSetWindowTitle("Teleportation Maze"); // Set the app name on the title bar

ofSetFrameRate(15);

ofBackground(255, 255, 255);

ofSetBackgroundAuto(TRUE);

// Get the window size for image loading

windowWidth = ofGetWidth();

windowHeight = ofGetHeight();

isDFS = 0;

isOpen = 0;

isFirst = 1;

// Centre on the screen

ofSetWindowPosition((ofGetScreenWidth()-windowWidth)/2, (ofGetScreenHeight()-windowHeight)/2);

// Load an image for the example

//myImage.loadImage("lighthouse.jpg");

// Window handle used for topmost function

hWnd = WindowFromDC(wglGetCurrentDC());

// Disable escape key exit so we can exit fullscreen with Escape (see keyPressed)

ofSetEscapeQuitsApp(false);

//

// Create a menu using ofxWinMenu

//

// A new menu object with a pointer to this class

menu = new ofxWinMenu(this, hWnd);

// Register an ofApp function that is called when a menu item is selected.

// The function can be called anything but must exist.

// See the example "appMenuFunction".

menu->CreateMenuFunction(&ofApp::appMenuFunction);

// Create a window menu

HMENU hMenu = menu->CreateWindowMenu();

//

// Create a "File" popup menu

//

HMENU hPopup = menu->AddPopupMenu(hMenu, "File");

//

// Add popup items to the File menu

//

// Open an maze file

menu->AddPopupItem(hPopup, "Make", false, false); // Not checked and not auto-checked

menu->AddPopupItem(hPopup, "Print and export", false, false);

// Final File popup menu item is "Exit" - add a separator before it

menu->AddPopupSeparator(hPopup);

menu->AddPopupItem(hPopup, "Exit", false, false);

//

// View popup menu

//

hPopup = menu->AddPopupMenu(hMenu, "View");

bShowInfo = true; // screen info display on

menu->AddPopupItem(hPopup, "Show DFS",false,false); // Checked

bTopmost = false; // app is topmost

bFullscreen = false; // not fullscreen yet

menu->AddPopupItem(hPopup, "Full screen", false, false); // Not checked and not auto-check

//

// Help popup menu

//

hPopup = menu->AddPopupMenu(hMenu, "Help");

menu->AddPopupItem(hPopup, "About", false, false); // No auto check

// Set the menu to the window

menu->SetWindowMenu();

} // end Setup

//

// Menu function

//

// This function is called by ofxWinMenu when an item is selected.

// The the title and state can be checked for required action.

//

void ofApp::appMenuFunction(string title, bool bChecked) {

ofFileDialogResult result;

string filePath;

//

// File menu

//

if(title == "Make") {

if (!isFirst) {

freeMemory();

}

else isFirst = 0;

makeTeleMaze();

isOpen = 1;

}

if (title == "Print and export") {

if (isOpen) {

printMaze();

}

else cout << "You must make a maze first" << endl;

}

if(title == "Exit") {

freeMemory();

ofExit(); // Quit the application

}

//

// Window menu

//

if(title == "Show DFS") {

if (isOpen)

{

if(DFS()) isDFS = TRUE;

else printf("DFS failed\n");

}

else

cout << "You must make a maze first" << endl;

}

if(title == "Full screen") {

bFullscreen = !bFullscreen; // Not auto-checked and also used in the keyPressed function

doFullScreen(bFullscreen); // But als take action immediately

}

//

// Help menu

//

if(title == "About") {

ofSystemAlertDialog("ofxWinMenu\nbasic example\n\nhttp://spout.zeal.co");

}

} // end appMenuFunction

//--------------------------------------------------------------

void ofApp::update() {

}

//--------------------------------------------------------------

void ofApp::draw() {

int i, j;

float y, x;

// 저장된 자료구조를 이용해 미로를 그린다.

if (isDFS)

{

ofSetColor(200);

ofSetLineWidth(5);

if (isOpen) {

DFSdraw();

}

else {

cout << "You must make a maze first." << endl;

isDFS = FALSE;

}

}

ofSetColor(100);

ofSetLineWidth(5);

if (isOpen)

{

float roomSize = (width > height) ? 700 / (float)width : 700 / (float)height;

ofDrawLine(0, 0, roomSize \* width, 0);

ofDrawLine(0, 0, 0, roomSize \* height);

for (i = 0, y = 0; i < height; i++, y += roomSize) {

for (j = 0, x = 0; j < width; j++, x += roomSize) {

if (maze[i][j].hasDownWall == TRUE) ofDrawLine(x, y + roomSize, x + roomSize, y + roomSize);

if (maze[i][j].hasRightWall == TRUE) ofDrawLine(x + roomSize, y, x + roomSize, y + roomSize);

if (tele[i][j].isTele == 1) {

int colorN = tele[i][j].teleNum;

string teleString = "";

teleString.push\_back(colorN);

srand(colorN);

ofSetColor(rand() % 256, rand() % 256, rand() % 256);

ofDrawCircle(x + roomSize / 2, y + roomSize / 2, roomSize / 3);

ofSetColor(255);

ofDrawBitmapString(teleString, x + roomSize / 2 - 4, y + roomSize / 2 + 4);

ofSetColor(100);

}

}

}

}

} // end Draw

void ofApp::doFullScreen(bool bFull)

{

// Enter full screen

if(bFull) {

// Remove the menu but don't destroy it

menu->RemoveWindowMenu();

// hide the cursor

ofHideCursor();

// Set full screen

ofSetFullscreen(true);

}

else {

// return from full screen

ofSetFullscreen(false);

// Restore the menu

menu->SetWindowMenu();

// Restore the window size allowing for the menu

ofSetWindowShape(windowWidth, windowHeight + GetSystemMetrics(SM\_CYMENU));

// Centre on the screen

ofSetWindowPosition((ofGetScreenWidth()-ofGetWidth())/2, (ofGetScreenHeight()-ofGetHeight())/2);

// Show the cursor again

ofShowCursor();

// Restore topmost state

if(bTopmost) doTopmost(true);

}

} // end doFullScreen

void ofApp::doTopmost(bool bTop)

{

if(bTop) {

// get the current top window for return

hWndForeground = GetForegroundWindow();

// Set this window topmost

SetWindowPos(hWnd, HWND\_TOPMOST, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

ShowWindow(hWnd, SW\_SHOW);

}

else {

SetWindowPos(hWnd, HWND\_NOTOPMOST, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

ShowWindow(hWnd, SW\_SHOW);

// Reset the window that was topmost before

if(GetWindowLong(hWndForeground, GWL\_EXSTYLE) & WS\_EX\_TOPMOST)

SetWindowPos(hWndForeground, HWND\_TOPMOST, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

else

SetWindowPos(hWndForeground, HWND\_TOP, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);

}

} // end doTopmost

//--------------------------------------------------------------

void ofApp::keyPressed(int key) {

// Escape key exit has been disabled but it can be checked here

if(key == VK\_ESCAPE) {

// Disable fullscreen set, otherwise quit the application as usual

if(bFullscreen) {

bFullscreen = false;

doFullScreen(false);

}

else {

ofExit();

}

}

// Remove or show screen info

if(key == ' ') {

bShowInfo = !bShowInfo;

// Update the menu check mark because the item state has been changed here

menu->SetPopupItem("Show DFS", bShowInfo);

}

if(key == 'f') {

bFullscreen = !bFullscreen;

doFullScreen(bFullscreen);

// Do not check this menu item

// If there is no menu when you call the SetPopupItem function it will crash

}

} // end keyPressed

//--------------------------------------------------------------

void ofApp::keyReleased(int key){

}

//--------------------------------------------------------------

void ofApp::mouseMoved(int x, int y){

}

//--------------------------------------------------------------

void ofApp::mouseDragged(int x, int y, int button){

}

//--------------------------------------------------------------

void ofApp::mousePressed(int x, int y, int button){

}

//--------------------------------------------------------------

void ofApp::mouseReleased(int x, int y, int button){

}

//--------------------------------------------------------------

void ofApp::windowResized(int w, int h){

}

//--------------------------------------------------------------

void ofApp::gotMessage(ofMessage msg){

}

//--------------------------------------------------------------

void ofApp::dragEvent(ofDragInfo dragInfo){

}

void ofApp::freeMemory() {

// malloc한 memory를 free해주는 함수

int i;

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

free(maze[i]);

free(tele[i]);

}

free(maze);

free(tele);

for (i = 0; i < candidateSets.size(); i++) {

candidateSets[i].coordinates.clear();

vector<pair<int, int>>().swap(candidateSets[i].coordinates);

}

candidateSets.clear();

vector<TeleCandidateSets>().swap(candidateSets);

s.clear();

vector<Coordinate>().swap(s);

}

//DFS탐색을 하는 함수

bool ofApp::DFS()

{

int i, j;

Coordinate target = { height - 1, width - 1 };

Coordinate start = { 0, 0 };

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

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

maze[i][j].visited = FALSE;

}

}

s.clear();

s.push\_back(start);

maze[s.back().y][s.back().x].visited = TRUE;

while (!s.empty()) {

if (s.back().x == target.x && s.back().y == target.y) return true;

if (s.back().x != width-1 && maze[s.back().y][s.back().x + 1].visited == FALSE && maze[s.back().y][s.back().x].hasRightWall == FALSE) {

maze[s.back().y][s.back().x + 1].visited = TRUE;

Coordinate temp = { s.back().y , s.back().x + 1 };

s.push\_back(temp);

}

else if (s.back().y != height - 1 && maze[s.back().y + 1][s.back().x].visited == FALSE && maze[s.back().y][s.back().x].hasDownWall == FALSE) {

maze[s.back().y + 1][s.back().x].visited = TRUE;

Coordinate temp = { s.back().y + 1, s.back().x };

s.push\_back(temp);

}

else if (s.back().x != 0 && maze[s.back().y][s.back().x - 1].visited == FALSE && maze[s.back().y][s.back().x - 1].hasRightWall == FALSE) {

maze[s.back().y][s.back().x - 1].visited = TRUE;

Coordinate temp = { s.back().y , s.back().x - 1 };

s.push\_back(temp);

}

else if (s.back().y != 0 && maze[s.back().y - 1][s.back().x].visited == FALSE && maze[s.back().y - 1][s.back().x].hasDownWall == FALSE) {

maze[s.back().y - 1][s.back().x].visited = TRUE;

Coordinate temp = { s.back().y - 1, s.back().x };

s.push\_back(temp);

}

else if (tele[s.back().y][s.back().x].isTele == 1 && maze[tele[s.back().y][s.back().x].gotoY][tele[s.back().y][s.back().x].gotoX].visited == FALSE) {

maze[tele[s.back().y][s.back().x].gotoY][tele[s.back().y][s.back().x].gotoX].visited = TRUE;

Coordinate temp = { tele[s.back().y][s.back().x].gotoY, tele[s.back().y][s.back().x].gotoX };

s.push\_back(temp);

}

else {

s.pop\_back();

}

}

return false;

}

void ofApp::DFSdraw()

{

//DFS를 수행한 결과를 그린다.

int i, j;

float roomSize = (width > height) ? 700 / (float)width : 700 / (float)height;

float x\_coord = roomSize / 2, y\_coord = roomSize / 2;

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

x\_coord = roomSize / 2;

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

if (maze[i][j].visited) {

if (j + 1 < width && maze[i][j + 1].visited && !maze[i][j].hasRightWall) {

ofDrawLine(x\_coord, y\_coord, x\_coord + roomSize, y\_coord);

}

if (i + 1 < height && maze[i + 1][j].visited && !maze[i][j].hasDownWall) {

ofDrawLine(x\_coord, y\_coord, x\_coord, y\_coord + roomSize);

}

}

x\_coord += roomSize;

}

y\_coord += roomSize;

}

ofSetColor(255, 0, 0);

i = s.size() - 1;

while (i >= 0) {

int y1, x1, y2, x2;

y1 = s[i].y;

x1 = s[i].x;

i--;

if (i < 0) break;

y2 = s[i].y;

x2 = s[i].x;

if (!(tele[y1][x1].isTele == 1 && tele[y2][x2].isTele == 1)) {

x1 = roomSize / 2 + x1 \* roomSize;

y1 = roomSize / 2 + y1 \* roomSize;

x2 = roomSize / 2 + x2 \* roomSize;

y2 = roomSize / 2 + y2 \* roomSize;

ofDrawLine(x1, y1, x2, y2);

}

}

}

void ofApp::makeTeleMaze() {

int i;

srand(time(NULL));

while (1) {

printf("Enter a width of a maze: ");

i = scanf("%d", &width);

if (i != 1) {

printf("Input a positive number.\n");

}

else if (width > 50) printf("Input is too large. - max width : 50\n");

else if (width <= 0) printf("Input a positive number.\n");

else break;

while (getchar() != '\n');

}

while (getchar() != '\n');

while (1) {

printf("Enter a height of a maze: ");

i = scanf("%d", &height);

if (i != 1) {

printf("Input a positive number.\n");

}

else if (height > 50) printf("Input is too large. - max height : 50\n");

else if (height <= 0) printf("Input a positive number.\n");

else break;

while (getchar() != '\n');

}

while (getchar() != '\n');

printf("\n");

initializeMaze();

for (i = 0; i < height - 1; i++) {

fillRoomNum(i);

eraseRightWalls(i);

eraseDownWalls(i);

makeTeleCandidates(i);

activateTele(i);

}

fillRoomNum(i);

eraseLastRightWalls();

}

void ofApp::initializeMaze() {

int i, j;

maze = (MazeRoom\*\*)malloc(height \* sizeof(MazeRoom\*));

tele = (TeleRoom\*\*)malloc(height \* sizeof(TeleRoom\*));

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

\*(maze + i) = (MazeRoom\*)malloc(width \* sizeof(MazeRoom));

\*(tele + i) = (TeleRoom\*)malloc(width \* sizeof(TeleRoom));

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

maze[i][j].roomNum = -1000;

maze[i][j].visited = 0;

maze[i][j].hasRightWall = 1;

maze[i][j].hasDownWall = 1;

tele[i][j].isTele = 0;

tele[i][j].teleNum = 0;

tele[i][j].gotoX = 0;

tele[i][j].gotoY = 0;

}

}

lastRoomN = 0;

lastTeleN = ' ';

return;

}

void ofApp::eraseRightWalls(int rowN) {

int j, k;

for (j = 0; j < width - 1; j++) {

if (maze[rowN][j].roomNum != maze[rowN][j + 1].roomNum) {

if (rand() % 2 == 0) {

maze[rowN][j].hasRightWall = 0;

int compareWith = maze[rowN][j + 1].roomNum;

for (k = 0; k < width; k++) {

if (maze[rowN][k].roomNum == compareWith) {

maze[rowN][k].roomNum = maze[rowN][j].roomNum;

}

}

int selectedSet1, selectedSet2;

int found1 = 0, found2 = 0;

for (k = 0; k < candidateSets.size(); k++) {

if (candidateSets[k].setNum == maze[rowN][j].roomNum) {

selectedSet1 = k;

found1 = 1;

}

else if (candidateSets[k].setNum == compareWith) {

selectedSet2 = k;

found2 = 1;

}

}

if (found1 && found2) {

candidateSets[selectedSet1].coordinates.reserve(candidateSets[selectedSet1].coordinates.size() + candidateSets[selectedSet2].coordinates.size());

candidateSets[selectedSet1].coordinates.insert(candidateSets[selectedSet1].coordinates.end(), candidateSets[selectedSet2].coordinates.begin(), candidateSets[selectedSet2].coordinates.end());

candidateSets.erase(candidateSets.begin() + selectedSet2);

}

else if (found2) {

candidateSets[selectedSet2].setNum = maze[rowN][j].roomNum;

}

}

}

}

return;

}

void ofApp::eraseDownWalls(int rowN) {

int i, j;

int setN = 0;

unsigned char findFlag = 0;

RoomNumSet \*headSet = (RoomNumSet\*)malloc(1 \* sizeof(RoomNumSet));

headSet->nextSet = NULL;

RoomNumSet \*setPtr;

Node \*indexPtr;

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

if (rand() % 2 == 0) {

maze[rowN][j].hasDownWall = 0;

maze[rowN + 1][j].roomNum = maze[rowN][j].roomNum;

}

}

// 집합 중 최소 하나의 벽은 지워져야함.

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

findFlag = 0;

setPtr = headSet;

while (setPtr->nextSet != NULL) {

setPtr = setPtr->nextSet;

if (setPtr->roomNum == maze[rowN][j].roomNum) {

(setPtr->elementN)++;

if (maze[rowN][j].hasDownWall == 0) setPtr->connected = 1;

for (indexPtr = setPtr->index; indexPtr->next != NULL; indexPtr = indexPtr->next);

indexPtr->next = (Node\*)malloc(1 \* sizeof(Node));

indexPtr->next->data = j;

indexPtr->next->next = NULL;

findFlag = 1;

break;

}

}

if (findFlag == 0) {

setPtr->nextSet = (RoomNumSet\*)malloc(1 \* sizeof(RoomNumSet));

setPtr->nextSet->nextSet = NULL;

setPtr->nextSet->index = (Node\*)malloc(1 \* sizeof(Node));

setPtr->nextSet->roomNum = maze[rowN][j].roomNum;

setPtr->nextSet->elementN = 1;

if (maze[rowN][j].hasDownWall == 0) setPtr->nextSet->connected = 1;

else setPtr->nextSet->connected = 0;

setPtr->nextSet->index->data = j;

setPtr->nextSet->index->next = NULL;

}

}

RoomNumSet\* preSetPtr = headSet;

for (setPtr = headSet->nextSet; setPtr != NULL; setPtr = setPtr->nextSet) {

Node\* preIndexPtr;

if (setPtr->connected == 0) {

int randIndex = rand() % setPtr->elementN;

indexPtr = setPtr->index;

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

indexPtr = indexPtr->next;

}

maze[rowN][indexPtr->data].hasDownWall = 0;

maze[rowN + 1][indexPtr->data].roomNum = maze[rowN][indexPtr->data].roomNum;

setPtr->connected = 1;

}

preIndexPtr = setPtr->index;

for (indexPtr = setPtr->index->next; indexPtr != NULL; indexPtr = indexPtr->next) {

free(preIndexPtr);

preIndexPtr = indexPtr;

}

free(preIndexPtr);

free(preSetPtr);

preSetPtr = setPtr;

}

free(preSetPtr);

return;

}

void ofApp::makeTeleCandidates(int rowN) {

int j, k;

int blocked = 0; //둘러싸인 벽의 수

int pushed = 0; // coordinate 벡터에 push\_back 되었는가?

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

if (rowN == 0 && j == 0) j++;

blocked = 0;

pushed = 0;

//둘러싸인 벽의 수를 센다.

if (maze[rowN][j].hasDownWall == 1) {

blocked++;

}

if (maze[rowN][j].hasRightWall == 1) {

blocked++;

}

if (rowN == 0 || maze[rowN - 1][j].hasDownWall == 1) {

blocked++;

}

if (j == 0 || maze[rowN][j - 1].hasRightWall == 1) {

blocked++;

}

//만약 미로의 막다른 길이라면 텔레포트 플랫폼의 후보군에 추가한다.

if (blocked >= 3) {

//텔레포트 플랫폼의 set number와 일치하는 set number가 후보군에 있다면 그 set에 좌표를 추가한다.

for (k = 0; k < candidateSets.size(); k++) {

if (candidateSets[k].setNum == maze[rowN][j].roomNum) {

pair<int, int> temp(rowN, j);

candidateSets[k].coordinates.push\_back(temp);

pair<int, int>().swap(temp);

pushed = 1;

break;

}

}

//없다면 새로운 set을 만들고 그 set에 좌표를 추가한다.

if (!pushed) {

TeleCandidateSets temp1;

pair<int, int> temp2(rowN, j);

temp1.setNum = maze[rowN][j].roomNum;

temp1.coordinates.push\_back(temp2);

candidateSets.push\_back(temp1);

pair<int, int>().swap(temp2);

}

}

}

}

void ofApp::activateTele(int rowN) {

int j;

// 후보군에 충분한 set이 있을 경우 텔레포트 플랫폼 활성화

if (candidateSets.size() >= 2) {

//랜덤하게 두 set을 고른 후, 각 set에서 다시 랜덤하게 coordinate를 고른다.

//이렇게 고른 두 coordinate는 활성화될 텔레포트 플랫폼의 좌표 쌍이다.

int selectedSet1 = rand() % candidateSets.size();

int selectedSet2 = rand() % (candidateSets.size() - 1);

if (selectedSet2 >= selectedSet1) selectedSet2++;

int selectedCoord1 = rand() % candidateSets[selectedSet1].coordinates.size();

int selectedCoord2 = rand() % candidateSets[selectedSet2].coordinates.size();

int y1 = candidateSets[selectedSet1].coordinates[selectedCoord1].first;

int x1 = candidateSets[selectedSet1].coordinates[selectedCoord1].second;

int y2 = candidateSets[selectedSet2].coordinates[selectedCoord2].first;

int x2 = candidateSets[selectedSet2].coordinates[selectedCoord2].second;

//배열 tele에 확정된 텔레포트 플랫폼의 정보를 담는다.

lastTeleN++;

tele[y1][x1].isTele = 1;

tele[y1][x1].teleNum = lastTeleN;

tele[y1][x1].gotoX = x2;

tele[y1][x1].gotoY = y2;

tele[y2][x2].isTele = 1;

tele[y2][x2].teleNum = lastTeleN;

tele[y2][x2].gotoX = x1;

tele[y2][x2].gotoY = y1;

//텔레포트 플랫폼 확정 후 후보군에서 제거

candidateSets[selectedSet1].coordinates.erase(candidateSets[selectedSet1].coordinates.begin() + selectedCoord1);

candidateSets[selectedSet2].coordinates.erase(candidateSets[selectedSet2].coordinates.begin() + selectedCoord2);

//연결된 두 텔레포트 플랫폼 확정 후

//1. maze의 room number 갱신 - 두 set의 room number를 같게 만든다.

int set1Num = candidateSets[selectedSet1].setNum;

int set2Num = candidateSets[selectedSet2].setNum;

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

if (maze[rowN][j].roomNum == set2Num) {

maze[rowN][j].roomNum = set1Num;

}

if (maze[rowN + 1][j].roomNum == set2Num) {

maze[rowN + 1][j].roomNum = set1Num;

}

}

//2. 후보군 갱신 - set이 합쳐짐에 따라 후보군이 속한 set 또한 변경된다.

candidateSets[selectedSet1].coordinates.reserve(candidateSets[selectedSet1].coordinates.size() + candidateSets[selectedSet2].coordinates.size());

candidateSets[selectedSet1].coordinates.insert(candidateSets[selectedSet1].coordinates.end(), candidateSets[selectedSet2].coordinates.begin(), candidateSets[selectedSet2].coordinates.end());

candidateSets.erase(candidateSets.begin() + selectedSet2);

if (selectedSet2 < selectedSet1) selectedSet1--;

if (candidateSets[selectedSet1].coordinates.empty()) {

candidateSets.erase(candidateSets.begin() + selectedSet1);

}

}

}

void ofApp::fillRoomNum(int rowN) {

int j;

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

if (maze[rowN][j].roomNum < 0) {

lastRoomN++;

maze[rowN][j].roomNum = lastRoomN;

}

}

return;

}

void ofApp::eraseLastRightWalls() {

int j, k;

for (j = 0; j < width - 1; j++) {

if (maze[height - 1][j].roomNum != maze[height - 1][j + 1].roomNum) {

int compareWith = maze[height - 1][j + 1].roomNum;

maze[height - 1][j].hasRightWall = 0;

for (k = 0; k < width; k++) {

if (maze[height - 1][k].roomNum == compareWith) {

maze[height - 1][k].roomNum = maze[height - 1][j].roomNum;

}

}

int selectedSet1, selectedSet2;

int found1 = 0, found2 = 0;

for (k = 0; k < candidateSets.size(); k++) {

if (candidateSets[k].setNum == maze[height - 1][j].roomNum) {

selectedSet1 = k;

found1 = 1;

}

else if (candidateSets[k].setNum == compareWith) {

selectedSet2 = k;

found2 = 1;

}

}

if (found1 && found2) {

candidateSets[selectedSet1].coordinates.reserve(candidateSets[selectedSet1].coordinates.size() + candidateSets[selectedSet2].coordinates.size());

candidateSets[selectedSet1].coordinates.insert(candidateSets[selectedSet1].coordinates.end(), candidateSets[selectedSet2].coordinates.begin(), candidateSets[selectedSet2].coordinates.end());

candidateSets.erase(candidateSets.begin() + selectedSet2);

}

}

}

return;

}

void ofApp::printMaze() {

FILE\* fp = fopen("maze.maz", "w");

int i, j;

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

fprintf(fp, "+-");

printf("+-");

}

fprintf(fp, "+\n");

printf("+\n");

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

fprintf(fp, "|");

printf("|");

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

if (tele[i][j].isTele == 1) {

fprintf(fp, "%c", tele[i][j].teleNum);

printf("%c", tele[i][j].teleNum);

}

else {

fprintf(fp, " ");

printf(" ");

}

if (maze[i][j].hasRightWall == 1) {

fprintf(fp, "|");

printf("|");

}

else {

fprintf(fp, " ");

printf(" ");

}

}

fprintf(fp, "\n");

printf("\n");

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

fprintf(fp, "+");

printf("+");

if (maze[i][j].hasDownWall == 1) {

fprintf(fp, "-");

printf("-");

}

else {

fprintf(fp, " ");

printf(" ");

}

}

fprintf(fp, "+\n");

printf("+\n");

}

fclose(fp);

printf("\n");

return;

}