컴퓨터공학 설계 및 실험1 기말 프로젝트

전공: 컴퓨터공학과　　분반: 수요일　　학번: 20171643　　이름: 박준혁

1. 개요

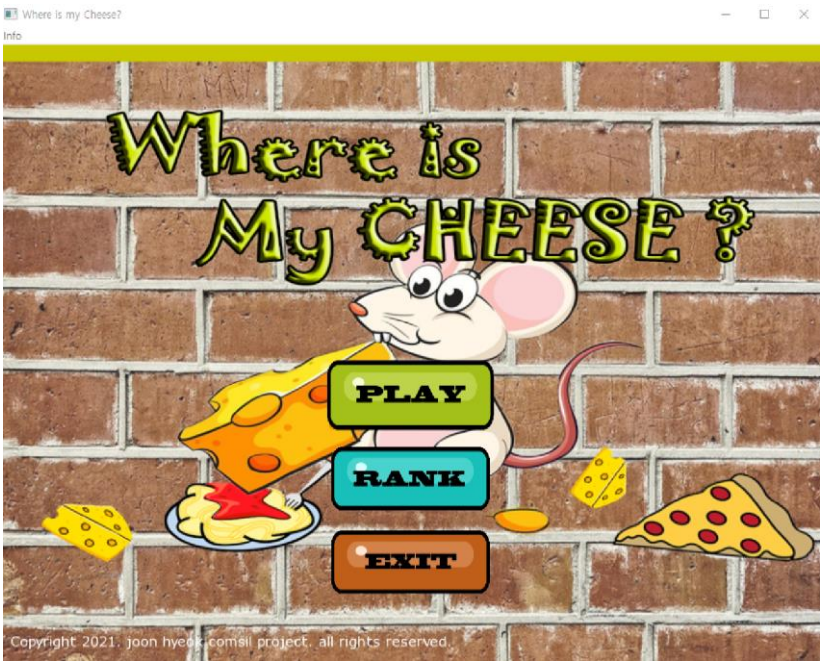
이번 기말 프로젝트를 위해 제작한 프로그램의 이름은 **'Where is my Cheese?'** 라는 이름의 **미로탈출 게임**이다. 매 게임이 실행될 때마다 임의의 미로가 주어지고, 미로의 특정 지점에서 쥐가 출발하여 치즈가 있는 목적지까지 도달하면 점수를 계산하고 1회의 게임이 종료된다. 이번 학기에서 다루었던 '미로 생성', '미로 표현', '최단 거리 계산' 개념과, 이전의 테트리스에서 다루었던 일부 개념들을 기반으로 구현하였다.

2. 프로젝트 목표

이번 프로젝트 개발 과정에서 가장 중요하게 생각한 목표는 다음과 같았다.

"플레이할만한 가치가 있는 게임을 만들어보자. '나'부터 흥미를 느낄 수 있는 게임을 만들어보자."

따라서 본인이 개발 과정에서 가장 심혈을 기울인 부분은 바로 **'게임으로서의 자격'**을 갖추는 것이었다. 이를 위해 몇 가지 기능을 추가였는데, 첫 째는 **'랭킹 시스템'**이다. 사용자의 매 게임 실행 별 점수를 기록하여 랭킹화하는 것이다. 점수는 '소요 시간'과 '지나간 경로의 개수'를 특정한 공식을 이용해 계산한다. 점수가 낮을수록 더 높은 랭킹을 기록하게 된다. 이러한 랭킹 시스템은 게임의 기본 요소인 '경쟁심'을 입히기 위함이다.



두 번째는 바로 **'10초 간격으로 치즈의 위치가 변하는 기능'**이다. 단순히 고정된 목적지까지 도달하면 종료되는 미로탈출 게임의 경우 사용자 입장에서 '박진감'이라는 게임의 핵심 요소를 느낄 수 없다. 따라서 본인은 '10초'를 기준으로 하여 쥐가 10초 내에 치즈에 도달하지 못하면 해당 시점의 쥐의 위치에서 가장 먼 위치로 치즈의 위치가 Reset되도록 설정하였다. 10초인 이유는, 본인이 직접 수 차례 플레이한 결과 10초가 가장 적당했기 때문이다.

마지막으로는, **'그럴듯한 게임 GUI 구축을 위한 다양한 기능'**들이다. 게임은 무엇보다도 사람들의 이목을 집중시키는 그래픽 인터페이스가 중요하다. 따라서 본인은 이번 프로젝트 간에 다양한 부분에 있어서 '그럴듯한 게임 그래픽'을 구현하기 위해 노력하였다. 물론 80~90년대의 게임 그래픽 수준에 불과하지만, 그럼에도 현재까지 본인이 OFS에 대해 학습한 내용들을 토대로 현실적인 범위 내에서 최대한 노력하였다. 이에 대한 자세한 내용은 이어서 설명하도록 한다.

이 밖의 프로그램 외적인 목표로는, 이번 프로젝트를 통해 '특정한 프로그램을 개발하기 위한 '기획'->'기초 사항 설계'->'특정 기능별 함수 구현'->'각 기능을 합쳐 전체 프로그램 완성'->'프로그램의 버그 확인 및 수정/개선'의 **일련의 일반적인 프로그램 제작과정을 몸으로 익혀보고 싶다는 점**이었다. 이에 대한 소감 및 후기는 보고서 후반에 이어진다.

서론은 여기까지 하고, 프로젝트에 대한 자세한 설명을 시작하겠다.

3. 소스코드 (주석 포함)

전체 코드는 따로 제출하기 때문에 프로그램의 구조 및 기능 설명에 불필요한 부분들은 생략하였다. (.cpp와 .h파일을 합쳐 1000줄이 넘는다.) 한편, 이번 프로젝트 구현 간에 '미로실습'에서 조교님께 제공받았던 ofxWinMenu 소스를 일부 차용하였는데, 이는 제공받은 코드이므로, 따로 설명이 필요 없다 판단하여 보고서에 담지 않는다. (해당 소스의 기능을 실제 프로그램 실행 간에 '스크린 박스 표현 부분' 외에는 사용하지 않아 설명이 무의미한 측면도 있다.)

<ofApp.h>

```
//------ Fundamental C Structures for Implementaion ------//
struct Line {                       // 직선을 표현하기 위한 구조체
        float x1, y1;               // 직선의 양 끝 점 (x1, y1)과 (x2, y2)
        float x2, y2;
};

struct Point {                      // 점을 표현하기 위한 구조체
        int x, y;                   // (x, y)
};

struct Node {                       // 연결리스트를 구축하기 위한 노드 구조체
        char date[15];              // 게임 플레이가 종료된 날짜 및 시간을 기억
        int score;                          // 1회의 게임에서의 최종 점수를 기억
        struct Node *link;          // link포인터
};

class ofApp : public ofBaseApp {
public:
        //------ Fundamental Data Structures Declaration ------//
        struct Line *lines;// 미로를 화면 상에 그리기 위한 직선 배열
        struct Point *shortRoute;  // 최단 경로를 점 별로 기억할 스택
        struct Point *allRoute;             // 전체 경로를 점 별로 기억할 배열
        char **Maze;                // maze.maz로부터 미로를 읽어들일 2차원 배열
        int **Visit;                // 미로의 특정 칸 방문 여부를 기억할 2차원 배열
        int linesSize = 0; // lines 배열의 크기를 기억할 변수
        int sIdx = 0;               // 최단 경로 스택의 top을 지칭하는 변수
        int aIdx = 0;               // 전체 경로 배열의 크기를 기억할 변수
        struct Point move[4] = {   // DFS 최단거리 탐색에 사용될 방향 구조체
                { 1,0 }, { 0,1 }, { -1,0 }, { 0,-1 }
        };
        int *rows, *vert, *horiz;   // 임의의 미로를 생성하는데 사용되는 배열들
        ofPolyline shortroute;      // 최단경로를 그리는 과정에서 사용될 ofPolyLine
        ofPolyline allroute;        // 전체경로를 그리는 과정에서 사용될 ofPolyLine

        //------ Fundamental Coordinates Representation ------//
        int HEIGHT;                         // 미로의 높이를 기억
        int WIDTH;                          // 미로의 폭을 기억
        int coordX;                         // 쥐의 현재 위치를 기억할 변수
        int coordY;
        int targetX;                // 치즈(목적지)의 위치를 기억할 변수
        int targetY;

        //------ Essential Variables for Implementation ------//
        int selectIdx = 0; // 메인화면에서 슬롯 선택 위치를 판단하는 인덱스변수
```

```
int currentframe; // 현재 프레임수를 입력받을 변수
int playtime;               // 게임 플레이 소요 시간을 기억할 변수
int loadframe;              // 로딩 화면 시간 delay를 위한 프레임 기억 변수
struct Node * head = NULL;      // 랭킹 연결리스트의 헤드
int numOfRanks = 0;      // 연결리스트의 노드 개수 기억
char printScore[10];// 게임 플레이 간 스코어 표시를 위한 문자열
char date[10];              // 게임 플레이 종료 후 날짜 및 시간을 기억할 문자열
int currentTopRanker = 999;      // 현재 랭킹 시스템에서 최고점을 기억할 변수

//------ Flag Variables for Efficient Branching ------//
int isOpen;                         // maze.maz 파일 정상 오픈 여부 플래그
int first_heap_alloc_flag = 1;      // 게임 최초 실행 시의 메모리 할당 플래그
int first_line_flag = 1;     // 미로 생성 시 첫 번째 라인을 구성할 플래그
int last_line_flag = 0;             // 미로 생성 시 마지막 라인을 구성할 플래그
int initscrflag = 1;// 게임 초기화면 표시 플래그
int isload = 0;             // 로딩화면 표시 플래그
bool isgame;                // 게임 시작 플래그
int upflag = 0;             // '상'키가 눌렸을 때의 플래그
int downflag = 1;// '하'키가 눌렸을 때의 플래그
int rightflag = 0; // '우'키가 눌렸을 때의 플래그
int leftflag = 0;   // '좌'키가 눌렸을 때의 플래그
int resetinitflag = 1;      // 10초 간격 리셋을 위한 10초 소요 여부 플래그
int reachtargetflag = 0;// 목적지 도달 여부 플래그
int endsoundflag = 0;    // 게임 종료 플래그
bool isdfs;                         // DFS 결과 표시 플래그
int gameexit = 0;// 게임 종료 플래그
int rankscrflag = 0;// 랭킹 표현 플래그

//------ Fundamental Functions Declaration ------//
void freeMemory();      // 매 회 게임 간 재설정 및 게임 최종 종료 시 메모리 해제
bool readFile();   // maze.maz 파일을 읽어들여 미로 구축 및 화면 표시 준비
Line pathFind(int i, int j, int flag);  // 미로 구축 시 보조 함수
void randomize(int n, int *set);    // 임의 미로 생성 시 랜덤하게 초기 설정
void randomizeH(int n, int *set);   // 미로 생성 시 수평선 랜덤하게 설정
void randomizeV(int n);   // 미로 생성 시 수직선 랜덤하게 설정
void makemazefile(void);// 임의의 미로를 담은 maze.maz 파일을 생성
void drawMaze(FILE* fp, int n);     // 미로를 maze.maz에 그리는 함수
void createRankList(void); // rank.txt 파일을 읽어 랭킹 구축
void writeRankFile(void);  // rank.txt 파일에 현재의 랭킹을 업데이트
void newRank(int score);  // 매 게임 종료 시 새로운 점수를 랭킹에 삽입
void initScreen(void);      // 게임의 초기화면을 표시하는 함수
void loadingScreen(int loadframe); // 로딩화면을 표시하는 함수
void showRankBoard(void);       // 랭킹을 화면에 표시하는 함수
bool DFS();                         // 최단 경로 계산을 수행하는 DFS함수
void dfsdraw();             // DFS 결과 최단 경로를 화면에 표시하는 함수
void drawclock(double current);    // 게임 시 시계 동작을 표시하는 함수
```

```
        void PlayGame(void);// 1회의 게임을 전체적으로 통제하는 함수
        Point maximumDist(void);// 10초 간격 목적지 리셋을 위한 함수

        //------ Base OFW & GUI Interface Declaration ------//

        (.. 중략 ..)

        //------ External Data Declaration for Importing ------//
        ofTrueTypeFont rankplace;          // GUI 및 사운드 등 게임 구현을 위해 필요한
        ofTrueTypeFont rankplace2;         // 외부 참조 파일을 프로그램에 로드하기 위한
        ofTrueTypeFont gamescore;          // 변수 선언부

        (.. 중략 ..)

        ofImage clocktimer10;
        ofImage clocktimer11;
        ofImage clocktimer12;
};
```

<ofApp.cpp>

```
using namespace std;
clock_t inittime;

//------ Basic Setup Settings for Openframeworks Program ------//
void ofApp::setup(void) {
        ofSetWindowTitle("Where is my Cheese?"); // 프로그램 최상단 윈도우형 화면 제목
        ofSetFrameRate(15);                // OFS의 프레임 비율을 설정
        ofBackground(200, 200, 0);         // 배경화면 색상 설정 - 진한 노란색
        windowWidth = ofGetWidth()*1.5; // 게임 화면 크기 설정
        windowHeight = ofGetHeight()*1.5;
        isdfs = false;      // isdfs 플래그 0 초기화
        isOpen = 0;                 // isOpen 플래그 0 초기화

        //------ GUI Settings (Offered from TA when MAZE Practice Class) ------//
        (.. 중략 .. : ofxWinMenu 기초 설정부)

        //------ External Data Loading Part ------//
        gamelogo.load("gamelogo.png");          // GUI 및 사운드 bgm 구현을 위한 각종
        playbutton.load("playbutton.png");// 외부데이터를 로드하고, 크기, 색상, 볼륨,
        rankbutton.load("rankbutton.png");      // 속도 등을 제각각 설정하는 부분
        exitbutton.load("exitbutton.png");
        returnbutton.load("returnbutton.png");
        quitbutton.load("quitbutton.png");
        menubgm.load("menubgm.mp3");
```

```
        menubgm.setVolume(0.1);
        menusound.load("menusound.mp3");
        menusound.setVolume(0.6);
        menusound.setSpeed(2);
        if (initscrflag)      // 초기화면의 경우
                menubgm.play(); // 초기화면 bgm 실행
        (.. 중략 .. : 너무 방대한 길이)
        numberone.load("numberone.png");
        hooray.load("hooray.mp3");
        hooray.setSpeed(1.4);
        hooray.setVolume(0.2);
        decoration.load("decoration.png");

        inittime = clock();           // 게임 시작 시간을 기억
        createRankList(); // rank.txt 파일을 읽어 미리 랭킹 구축
        bShowInfo = 1;
}

(.. 중략 .. : 일부 empty 함수, 혹은 ofxWinMenu 관련 함수들은 생략)

//------ Core Functions Operating on frame-by-frame basis ------//
void ofApp::keyPressed(int key) {  // 키보드 입력에 대한 처리 부분
        if (key == VK_ESCAPE) {   // esc 누르면 종료 되도록 설정
                ofExit();
        }
        // 초기화면에서 play 슬롯을 선택한 상황에서 엔터키를 누르는 경우 : 게임 실행
        if (key == OF_KEY_RETURN && selectIdx == 0 && initscrflag == 1) {
                initscrflag = 0;    // 초기화면 해제
                reachtargetflag = 0;        // 목적지 도달 플래그 0 초기화
                menubgm.stop(); // 초기화면 bgm 해제
                freeMemory();    // 초기 설정으로 복구
                readFile();          // maze.maz 파일을 읽어 미로 구축

                if (isOpen) {        // maze.maz 파일 정상 오픈 시
                        isload = 1;         // 로딩화면 표시 (순전히 게임적 특성을 위함.)
                        coordX = coordY = 0;      // 쥐의 초기 위치 설정
                        targetX = WIDTH - 3;      // 치즈의 초기 위치 설정
                        targetY = HEIGHT - 3;
                }
        }
        // 초기화면에서 rank 슬롯을 선택한 상황에서 엔터키를 누르는 경우 : 랭킹 표시
        if (key == OF_KEY_RETURN && selectIdx == 1 && initscrflag == 1) {
                menusound.play();          // 슬롯 선택 사운드
                initscrflag = 0;    // 초기화면 해제
                rankscrflag = 1;  // 랭킹화면 표시
        }
        // 랭킹 표시 화면에서 다시 엔터키를 누르는 경우 : 초기화면으로 복귀
```

```
else if (key == OF_KEY_RETURN && selectIdx == 1 && rankscrflag == 1) {
        menusound.play();       // 슬롯 선택 사운드
        initscrflag = 1;    // 초기화면 표시
        rankscrflag = 0;  // 랭킹화면 해제
}
// 초기화면에서 exit 슬롯을 선택한 상황에서 엔터키를 누르는 경우 : 프로그램 종료
if (key == OF_KEY_RETURN && selectIdx == 2 && initscrflag == 1) {
        menusound.play();       // 슬롯 선택 사운드
        gameexit = 1;           // heap 메모리 해제를 위한 종료 플래그
        freeMemory();    // 모든 힙 할당 메모리 해제
        ofExit(); // 프로그램 종료
}

if (initscrflag && !isgame) {       // 초기화면에서
        if (key == OF_KEY_UP) {   // '상'키를 누르는 경우
                menusound.play();       // 슬롯 이동 사운드
                selectIdx--;       // 슬롯 표시 인덱스 변경
                if (selectIdx < 0) // 제한점 설정
                        selectIdx = 0;
        }

        if (key == OF_KEY_DOWN) {       // '하'키를 누르는 경우
                menusound.play();       // 위의 조건의 반대급부 수행
                selectIdx++;
                if (selectIdx == 3)
                        selectIdx = 2;
        }
}
// 게임 실행 상태 및 최종 목적지에 도달하지 않은 상태에서의 키입력 처리
if (isgame && !isload && !(coordX == targetX && coordY == targetY)) {
        if (key == 'q' || key == 'Q') {       // q키를 누르는 경우 1회 게임 탈출
                initscrflag = 1;    // 다시 초기화면 표시
                isgame = 0;             // 게임 화면 해제
                gamebgm.stop(); // 게임 bgm 해제
                menubgm.play(); // 초기화면 bgm 실행
        }

        if (key == OF_KEY_RIGHT) {       // '우'키를 누르는 경우
                if (coordX < WIDTH - 4) { // 미로 최우측의 경우 이동 불가
                        if (Maze[1 + coordY][1 + coordX + 1] == ' ') { // 미로 뚫려있을때
                                coordX += 2;      // 이동
                                rightflag = 1;      // 우측 플래그 설정 -> 쥐의 모양 변경을 위해
                                leftflag = upflag = downflag = 0; // 다른 방향 플래그 해제
                        }
                }
        }
```

```
                if (key == OF_KEY_LEFT) { // '좌'키를 누르는 경우
                        if (coordX) {       // 위의 조건부와 유사하게 작동
                                if (Maze[1 + coordY][1 + coordX - 1] == ' ') {
                                        coordX -= 2;
                                        leftflag = 1;
                                        rightflag = upflag = downflag = 0;
                                }
                        }
                }

                if (key == OF_KEY_UP) {
                        if (coordY) {
                                if (Maze[1 + coordY - 1][1 + coordX] == ' ') {
                                        coordY -= 2;
                                        upflag = 1;
                                        leftflag = rightflag = downflag = 0;
                                }
                        }
                }

                if (key == OF_KEY_DOWN) {
                        if (coordY < HEIGHT - 4) {
                                if (Maze[1 + coordY + 1][1 + coordX] == ' ') {
                                        coordY += 2;
                                        downflag = 1;
                                        leftflag = upflag = rightflag = 0;
                                }
                        }
                }
        }
}

void ofApp::draw() {        // 매 프레임 별로 화면을 그리는 함수
        int i, sec;         // 인덱스 변수 i와, 초를 기억할 sec변수
        char str[256];      // 저작권 문자열 표시
        Point temp;         // 가장 먼 꼭지점 좌표를 리턴받을 변수
        ofSetColor(255, 255, 255);// 모든 로드 이미지파일의 밝기 최대 설정
        ofSetLineWidth(5);          // 라인 굵기를 5로 설정
        brickwall.draw(0, 0, windowWidth, windowHeight);  // 게임의 바탕화면 :벽돌

        if (initscrflag) {    // 초기화면 플래그 설정 시
                initScreen();       // 초기화면 표시 함수
                loadframe = ofGetFrameNum();    // 로딩화면 표시 시 로딩 시작 프레임 기억
        }       // 을 위한 설정
```

```
if (rankscrflag) {  // 랭킹화면 플래그 설정 시
        rankboard.draw(33, 0, 900, 600);   // 랭킹보드 그림
        returnbutton.draw(670, 580, 210, 110);      // 복귀버튼 그림
        showRankBoard();          // 랭킹 화면에 표시 함수
}

if (isload)          // 로딩화면 플래그 설정 시
        loadingScreen(loadframe);// 로딩화면을 일정 프레임 동안 표시
// 순전히 게임적 요소를 갖추기 위한 로딩화면일 뿐, 실질적 데이터 로딩과정 전무
if (isgame) {       // 게임 플레이 플래그 설정 시
        whitewall.draw(10, 10, 600, 600);   // GUI구축 이미지 파일 표시
        quitbutton.draw(650, 580, 180, 80);          // 종료 버튼
        scoreboard.draw(637, 75, 300, 200);          // 스코어보드
        clockboard.draw(710, 350, 150, 150);// 시계
        decoration.draw(800, 420, 150, 250);// 우측 하단 쥐 그림

        if (!reachtargetflag) {       // 목적지에 도달하지 못했을 때
                if (resetinitflag) { // 10초 간격으로 플래그 설정 및 해제
                        resetcriteria = clock();      // 10초 간격 계산 기준점
                        resetinitflag = 0;
                }
                resetfull = clock();          // 10초 간격 계산 끝점
                drawclock(resetfull);        // 매 초마다 시계그림 회전 기능
                sec = (int)((resetfull - resetcriteria) / (double)1000);
                // 기준점으로부터 흐른 시간 초를 기억하는 sec변수
                if (sec > 9) {       // 10초가 지난 경우
                        temp = maximumDist();   // 현 위치에서 가장 먼 꼭짓점 확인
                        targetX = temp.x;          // 해당 꼭지점으로 목적지 변경
                        targetY = temp.y;
                        whoosh.play();    // 치즈가 움직이는 사운드 효과
                        resetinitflag = 1; // 다시 10초를 세기 위한 플래그 설정
                }
        }
        else
                drawclock(resetfull);        // 목적지 도달한 경우 시계 멈추도록 설정

        ofSetColor(0, 0, 0);          // 미로를 그리는 선은 까맣게 표시
        for (i = 0; i < linesSize; i++)
                ofDrawLine(20 * lines[i].x1, 20 * lines[i].y1, 20 * lines[i].x2, 20 * lines[i].y2);
        // 미로를 그리는 반복문
        ofSetColor(255, 255, 255);// 미로 제외하고는 모두 밝기 최대
        if (isOpen) {       // 에러 처리
                if (isdfs) // DFS가 정상적으로 수행되었다면,
                        dfsdraw();          // DFS 결과인 최단경로를 화면에 표시
                PlayGame();       // 게임 플레이를 진행하는 게임 플레이 메인 함수(추후 설명)
        }
```

```
		}

		if (bShowInfo) {
			ofSetColor(255);  // 카피 라이트 문자열 표시
			sprintf(str, "Copyright 2021. joon hyeok comsil project. all rights reserved.");
			myFont.drawString(str, 15, ofGetHeight() - 20);
		}
	}

//------ Assistant Functions for draw() in 'Game Play Mode' ------//
// 치즈의 10초 간격 새로운 위치를 찾아주는 함수. 미로의 4개의 꼭짓점들이
// 후보이다. 아래의 temp배열은 이들을 가리킨다.
// 현위치를 기준으로 절대 거리를 각각 계산하여 판단한다.
Point ofApp::maximumDist(void) {
		int maxIdx;		// 가장 거리간 먼 꼭지점의 인덱스 기억
		int maximum;// 최댓값 비교를 위한 변수
		Point temp[4] = { {WIDTH - 3,HEIGHT - 3},{WIDTH - 3,0},{0,HEIGHT - 3},{0,0} };
		Point tmp = temp[0];		// 초기 설정

		maximum = (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y);
		maxIdx = 0;
		// 가장 거리가 먼 지점 찾는 과정 : 점과 점 사이의 거리를 피타고라스정리를 이용해
		tmp = temp[1];  // '절대적인 거리'를 비교함. 경로의 개수가 아님.
		if (maximum < (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y)) {
			maximum = (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y);
			maxIdx = 1;
		}
		tmp = temp[2];
		if (maximum < (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y)) {
			maximum = (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y);
			maxIdx = 2;
		}
		tmp = temp[3];
		if (maximum < (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y)) {
			maximum = (coordX - tmp.x)*(coordX - tmp.x) + (coordY - tmp.y)*(coordY - tmp.y);
			maxIdx = 3;
		}

		return temp[maxIdx];		// 절대거리가 가장 먼 꼭지점을 반환
}

void ofApp::drawclock(double current) {		// 게임플레이 동안 시계를 그리는 함수
		switch ((int)((resetfull - inittime) / (double)1000) % 12) { // 초별로
		case 0:
			clocktimer1.draw(710, 350, 150, 150);
			break;
		case 1:  // 매초 간격으로 시계가 동적으로 움직이는 것처럼 보이게 하는 함수이다.
```

```
                clocktimer2.draw(710, 350, 150, 150);
                break;   // 실제로는 이처럼 12개의 상황별 그림을 보여주는 과정이다.
        case 2:
                clocktimer3.draw(710, 350, 150, 150);
                break;
(.. 중략 ..)
        case 10:
                clocktimer11.draw(710, 350, 150, 150);
                break;
        case 11:
                clocktimer12.draw(710, 350, 150, 150);
                break;
        }
  }

//------ Create a random basis maze of WIDTH 15, HEIGHT 15 ------//
void ofApp::makemazefile(void) {  // 매 게임 회차를 위한 임의의 미로를 생성하는 함수
        FILE*fp = fopen("maze.maz", "wt");          // 파일을 작성하도록 스트림 설정
        int width, height; // 폭과 높이를 기억
        int set = 0, i;      // 수업 시간에 사용했던 Eller's Algorithm을 사용한다.
        width = 15;
        height = 15;

        if (first_heap_alloc_flag) {  // 최초 게임 실행 시에만 메모리를 할당한다.
                rows = (int*)malloc(sizeof(int)*width);
                vert = (int*)malloc(sizeof(int)*(width - 1));
                horiz = (int*)malloc(sizeof(int)*width);
        }       // 다 차례 게임 회차 실행 시 메모리가 터지지 않게 하기 위해 메모리를 한 번만
        // 할당하고 최초 이후 게임 실행부터는 해당 메모리를 재사용하는 것이다.

        for (i = 0; i < width; i++)
                fprintf(fp, "+-");  // 최상단 테두리 그림
        fprintf(fp, "+₩n");

        for (i = 0; i < height; i++) {          // 높이별로 순회하며,
                if (first_line_flag) // 첫 번째 줄을 그릴 때는
                        randomize(width, &set);   // Randomize 함수 호출
                else {    // 그 외의 줄에 대해서는,
                        randomizeH(width, &set); // 수평선 별로 랜덤하게 설정
                        if (i == (height - 1))        // 마지막 라인일 때만 따로 분기
                                last_line_flag = 1;
                        randomizeV(width);        // 수직선 별로 랜덤하게 설정
                }
                drawMaze(fp, width);       // 미로를 maze.maz 파일에 문자들로 그린다.
                first_line_flag = 0;          // 플래그 변수 해제
        }
        for (i = 0; i < width; i++)
```

```
            fprintf(fp, "+-");  // 최하단 테두리 그림
        fprintf(fp, "+₩n");

        fclose(fp);
}

void ofApp::randomize(int n, int *set) {      // 최초 seed 배정 함수
        int prevIdx = -1; // 초기 set 초기화
        int i, j;

        srand(time(NULL));          // seed 배정
        for (i = 0; i < n - 1; i++) {
            vert[i] = rand() % 2;       // 수직선을 랜덤하게 설정

            if (vert[i]) {          // 수직선이 있는 경우
                for (j = ++prevIdx; j <= i; j++)     // set를 나눈다.
                    rows[j] = *set;
                prevIdx = i;        // prevIdx가 i를 기억
                (*set)++;           // set가 늘어나야함. (당연)
            }
        }

        for (i = ++prevIdx; i < n; i++)
            rows[i] = *set;    // 한 줄에 대하여 set 배정을 수행
        (*set)++;
}

void ofApp::randomizeV(int n) {    // 지속적 수직선 랜덤 배정 함수
        int prevSet;
        int flag = 0;       // 랜덤 할당 플래그
        int i, j;

        for (i = 0; i < n - 1; i++) {
            if (rows[i] != rows[i + 1]) {           // set가 바뀌는 지점에서
                if (last_line_flag) flag = 1; // 마지막 라인인 경우에만 특별 분기
                else flag = rand() % 2;     // 그 외의 라인은 모두 랜덤하게 설정

                if (flag) {           // 랜덤플래그가 1일 때
                    vert[i] = 0;        // 수직선이 없다고 설정

                    prevSet = rows[i + 1];      // set를 끌어와서
                    rows[i + 1] = rows[i];

                    for (j = 0; j < n; j++)       // set를 합친다.
                        if (rows[j] == prevSet)
                            rows[j] = rows[i];
                }
```

```
                        else
                                vert[i] = 1;        // 그 외에는 수직선이 존재
                }
                else
                        vert[i] = 1;        // set가 같은 경우엔 vert[i]=1로 설정
        }
}

void ofApp::randomizeH(int n, int *set) {    // 지속적 수평선 랜덤 배정 함수
        int prevSet;
        int flag = 0;       // 수평선 존재 유무를 가리킬 플래그 변수
        int i;

        prevSet = rows[0];          // 초기 set
        for (i = 0; i < n; i++) {
                horiz[i] = rand() % 2;       // 수평선 임의 배정

                if (!horiz[i])        // 없는 경우엔 플래그 변수를 설정
                        flag = 1;

                if (i < n - 1) {     // 라인을 순회하는데,
                        if (rows[i + 1] != prevSet) {          // set가 다르다면,
                                if (!flag) // 이때 수평선이 없는 경우
                                        horiz[i] = 0;       // 그대로 수평선 없게 냅두고
                                else      // 수평선이 있는 경우
                                        flag = 0;// 플래그 변수 해제

                                prevSet = rows[i + 1];      // 앞의 set를 업데이트
                        }
                }
                if ((i == n - 1) && !flag)  // 플래그가 해제되어 있고, 마지막 인덱스일 땐
                        horiz[i] = 0;       // 특수하게 수평선 없게 설정

                if (horiz[i]) {       // 수평선이 있을 때는 set를 그대로 따라게 하면 된다.
                        rows[i] = *set;
                        (*set)++;
                }
        }
  }

//------ Read 'Maze.maz' file to create a random basis graphic maze ------//
// 임의 미로 생성 기능을 통해 생성된 maze.maz 파일을 읽어 OFS 상에서 실제로 미로를
// 구축하는 함수
bool ofApp::readFile(void) {
        ifstream inputFile, tempFile;
        string line;
        int i, j;
```

```
makemazefile();  // 우선 앞서서 임의 미로를 생성해놓고(maze.maz를 생성해놓고)
HEIGHT = 0;

inputFile.open("maze.maz");          // 파일을 열어 읽는다.
tempFile.open("maze.maz");

if (inputFile.is_open()) {    // 파일이 정상적으로 열렸다면,
        ofFile file("maze.maz"); // 한 번 더 확인
        string tempStr;

        if (!file.exists()) { // 파일이 안열렸다면 오류 처리
                cout << "Target file does not exists." << endl;
                return false;
        }
        else {    // 파일을 찾은 경우 isOpen 플래그 설정
                cout << "We found the target file." << endl;
                isOpen = 1;
        }

        while (getline(inputFile, line)) {      // maze.maz파일을 줄별로 읽음
                WIDTH = line.length();     // 한 줄의 길이를 폭으로 기록
                HEIGHT++;          // 높이를 카운트
        }

        if (first_heap_alloc_flag) {  // 최초 게임 실행 시에만 메모리 할당
                Maze = new char*[HEIGHT];          // 그 외에는 메모리 효율성을 위해 기할당
                for (i = 0; i < HEIGHT; i++)          // 메모리를 재사용할 것
                        Maze[i] = new char[WIDTH];
                lines = new struct Line[HEIGHT*WIDTH + 10];        // 적절하게 크기 배정
        }

        for (i = 0; i < HEIGHT; i++) {        // 라인 별로
                getline(tempFile, tempStr);          // 문자열을 읽어
                for (j = 0; j < WIDTH; j++)
                        Maze[i][j] = tempStr[j];    // 해당 문자열을 미로 배열에 문자별 기록
        }
        shortRoute = new struct Point[10000];       // 최단 경로 배열 할당 (동적)
        allRoute = new struct Point[10000];          // 전체 경로 배열 할당 (동적)
        // 아래는 미로 배열을 기반으로 미로 그리기를 위한 기반 구축 루프
        for (i = 0; i < HEIGHT; i++) {
                for (j = 0; j < WIDTH; j++) {
                        if (Maze[i][j] == '-')
                                lines[linesSize++] = pathFind(i, j, 1); // 적절한 크기 설정
                        else if (Maze[i][j] == '|')
                                lines[linesSize++] = pathFind(i, j, 0);
                }
```

```
                }
                return true;
        }
        else {
                printf("   Needs a '.maz' extension₩n");      // 에러처리
                return false;       // 실습 코드의 잔존물
        }
}

Line ofApp::pathFind(int i, int j, int flag) {   // 미로를 화면 상에 적합하게 그리
        Line path = { j,i,j,i };        // 기 위해 직선의 끝점 간격을 적절하게 조정해서 반
        // 환해주는 함수로, 해당 작업이 전체 코드에서 몇 차례 반복적으로 일어나기 때문에
        if (flag) {          // 이를 함수모듈화하였다.
                path.x1 -= 0.5;
                path.y1 += 0.5;
                path.x2 += 1.5;
                path.y2 += 0.5;
        }
        else {
                path.x1 += 0.5;
                path.y1 -= 0.5;
                path.x2 += 0.5;
                path.y2 += 1.5;
        }

        return path;       // Line 구조체 변수를 반환함.
  }

//------ Main Function for 'Game Play Mode' ------//
// 1회의 게임 플레이를 전체적으로 통제하는 이번 프로젝트의 핵심 함수
// 프레임 별로, 상황 플래그 별로, 키 입력 별로 모두 다른 동작을 하는
// 함수이다. 함수에 대한 자세한 설명은 주석으로 이어간다.
void ofApp::PlayGame() {
        Point coord;       // 현위치를 기억할 변수
        coord.x = coordX;          // 현 위치 업데이트
        coord.y = coordY;
        // 목적지에 치즈를 그린다.
        cheese.draw(20 * targetX + 15, 20 * targetY + 15, 35, 35);

        if (coordX != targetX || coordY != targetY)  // 목적지에 도달하지 않은
                allRoute[aIdx++] = coord;// 모든 이동을 allRoute배열에 기록한다.

        if (upflag)          // '상'키가 눌렸을 경우 쥐가 위를 바라보게 한다.
                ratup.draw(20 * coordX + 15, 20 * coordY + 15, 35, 35);
        else if (downflag)// '하'키가 눌렸을 경우 쥐가 아래를 바라보게 한다.
                ratdown.draw(20 * coordX + 15, 20 * coordY + 15, 35, 35);
        else if (rightflag) // 이하 유사 기능
```

```
            ratright.draw(20 * coordX + 15, 20 * coordY + 15, 35, 35);
    else if (leftflag)
            ratleft.draw(20 * coordX + 15, 20 * coordY + 15, 35, 35);
    printf("(%d, %d)₩n", coordX, coordY);       // 현재 좌표를 콘솔 상에 표시

    if (coordX == targetX && coordY == targetY) {       // 치즈에 도달한 경우
            reachtargetflag = 1;        // 목적지 도달 플래그 설정

            gamebgm.stop(); // 게임 bgm을 끈다.
            if (!endsoundflag) {        // 정확히 '종료 시점'에 대해서 (플래그 이용)
                    DFS();    // 최단 경로 계산 with DFS
                    isdfs = 1;          // DFS 결과를 그리라는 명령을 draw함수에 전달
                    endgame = clock();          // 게임이 종료된 시점을 기록
                    playtime = (int)((endgame - startgame) / (double)1000); // 소요시간
                    playtime += aIdx / 2;       // 과 전체경로 인덱스를 이용하여 점수를 계산
                    if (playtime > 999)         // 점수가 999점 이상인 경우(정말 낮은 점수)
                            playtime = 999;  // 무의미하다 판단하여 이때부터는 모두 999점으로 통일
                    if (playtime < currentTopRanker)   // 현재 플레이 점수가 1등에 해당하면
                            hooray.play();      // 1등 기념 효과음 출력
                    else        // 1등이 아니면 모두 통일된 종료 사운드 실행
                            endsound.play();

                    currentframe = ofGetFrameNum() + 104;    // 잠시 대기를 위한 프레임 계산
                    endsoundflag = 1;           // '종료 시점'만을 나누기 위한 플래그 설정
            }       // 시점을 정확히 나눈 이유 : 위의 조건문 상황이 프레임별로 반복하면 안됨.
            ofSetColor(255, 255, 255);// 밝기 최대 설정하여
            gameover.draw(60, 50, 480, 480);  // 게임 종료 상황을 그림

            sprintf(printScore, "%d", playtime);// 점수를 화면에 표시할 문자열 설정
            ofSetColor(0, 0, 0);        // 게임 스코어 문자열은 어둡게 표시하기 위함
            gamescore.drawString(printScore, 720, 200);         // 화면에 표시
            ofSetColor(255, 255, 255);// 점수를 제외하고는 다시 밝게 표시하기 위함

            if (playtime < currentTopRanker)   // 1등일 때는 1등 기념 그림을 표시함.
                    numberone.draw(33, 0, 900, 600);

            if ((currentframe - ofGetFrameNum()) % 105 == 0) {// 잠시 대기 시간이 지나면
                    newRank(playtime);          // 점수를 랭킹에 포함시키고
                    initscrflag = 1;    // 초기화면으로 복귀
                    isgame = 0;         // 1회의 플레이 종료
                    menubgm.play(); // 초기화면 bgm을 다시 튼다.
            }
    }
    else {    // 목적지에 도달하지 못한 경우 (전반적인 상황)
            playtime = (int)((clock() - startgame) / (double)1000);
            playtime += aIdx / 2;       // 현 점수를
```

```
                if (playtime > 999)
                        playtime = 999;
                sprintf(printScore, "%d", playtime);// 화면에 표시한다.
                ofSetColor(100, 100, 100);// 최종 점수 표시와 색을 구분하였다.
                gamescore.drawString(printScore, 720, 200);
                ofSetColor(255, 255, 255);
        }
}

//------ Find the shortest route when user reached the target ------//
// 목적지에 도달한 경우, 해당 목적지까지의 최단경로를 계산할 DFS 함수.
// 이때, 게임 플레이 상으로 리셋된 치즈의 최종 위치만을 판단한다.
// 즉, 최초 시작점 coordX = 0, coordY =0에서부터 종료 시점에서의 최종 치즈
// 위치까지의 최단 경로를 보이는 것이다. 중간에 지나친 목적지들은 고려하지 않음.
bool ofApp::DFS() {
        int moveOrNot;  // 움직일지 말지를 정하는 플래그 변수
        int i, j;
        struct Point p1, p2;

        if (first_heap_alloc_flag) {  // 최초 게임 시작 시의 동적할당
                Visit = new int*[HEIGHT]; // 그외에는 메모리 재사용할 것(상기)
                for (i = 0; i < HEIGHT; i++)
                        Visit[i] = new int[WIDTH];
                first_heap_alloc_flag = 0;  // 이 친구가 최초 시점의 가장 마지막 할당이기
        }       // 때문에 여기에서 플래그를 해제한다.

        for (i = 0; i < HEIGHT; i++) {        // 미로 배열을 기반으로
                for (j = 0; j < WIDTH; j++) {        // 움직일 수 있는 곳과 없는 곳을 Visit배열
                        if (Maze[i][j] != ' ')        // 에 기록해둔다.
                                Visit[i][j] = 1;
                        else
                                Visit[i][j] = 0;
                }
        }
        Visit[1][1] = 1;    // 최초 시작점은 이미 방문한 것으로 여긴다.
        p1.x = 1;           // 시작점 : 이때, 중요한 점은, OFS 화면 표시 좌표와, DFS 계산 시의
        p1.y = 1;           // 좌표가 다르다는 것을 인식하는 것이다.

        shortRoute[sIdx++] = p1; // 스택에 푸쉬

        while (sIdx) {      // 스택이 비기 전까지
                if (shortRoute[sIdx - 1].x == (targetX + 1) && shortRoute[sIdx - 1].y == (targetY + 1)) {
                        printf("Reached the Destination!!₩n");
                        return true;        // 목적지에 도달하면 종료
                }
```

```
                moveOrNot = 0; // 우선 이동하지 않음으로 설정
                for (i = 0; i < 4; i++) {     // 이동가능한지 확인하는 부분
                        p2.x = shortRoute[sIdx - 1].x + move[i].x;   // 새 고려 위치
                        p2.y = shortRoute[sIdx - 1].y + move[i].y;

                        if (!Visit[p2.y][p2.x]) {       // 해당 위치가 아직 미방문 위치라면
                                moveOrNot = 1; // 이동하기로 결정!
                                // 해당 이동 위치를 최단경로 스택에 푸쉬한다. 방문여부도 당연 기록
                                Visit[shortRoute[sIdx - 1].y + move[i].y][shortRoute[sIdx - 1].x + move[i].x] =
1;

                                shortRoute[sIdx++] = p2;
                                break;
                        }
                }
                if (!moveOrNot)  // 이동 가능성이 없다면 이동가능성이 있을 때까지 스택을
                        sIdx--;   // 팝한다.
        }
}

void ofApp::dfsdraw() {    // DFS 결과를 토대로 최단경로를 화면에 표시하는 함수
        int wholePath = aIdx;      // 게임이 종료되는 시점에만 발동한다.
        int shortestPath = sIdx;   // 인덱스를 기억
        int i;

        for (i = 0; i < wholePath; i++) {
                allroute.addVertex(20 * allRoute[aIdx - 1].x + 30, 20 * allRoute[aIdx - 1].y + 30);
                aIdx--;   // 전체 경로를 화면에 빨간색으로 표시한다.
        }
        for (i = 0; i < shortestPath; i++) {
                shortroute.addVertex(20 * shortRoute[sIdx - 1].x + 10, 20 * shortRoute[sIdx - 1].y + 10);
                sIdx--;   // 최단 경로를 화면에 녹색으로 표시한다.
        }
        ofSetLineWidth(5);
        ofSetColor(140, 0, 0);
        allroute.draw();   // 배열을 토대로 ofPolyLine OFS 문법을 이용해 간단하게
        ofSetColor(0, 140, 0);       // 그려준다. ofPolyLine이기 때문에 오버라이드되면서
        shortroute.draw();           // 그려진다는 것을 인식한다.
  }

//------ Functions for Graphic User Interface ------//
// 앞선, 임의 미로 생성 함수에서 미로를 maze.maz 파일에 문자 단위로 출력하는 함수
void ofApp::drawMaze(FILE* fp, int n) {
        int i;

        if (!first_line_flag) {          // 첫 째 라인이 아닐 때의 출력
                fprintf(fp, "+");
```

```
                for (i = 0; i < n; i++) {
                        if (horiz[i])         // 수평선이 있으면 아래를 출력
                                fprintf(fp, "-+");
                        else      // 없으면 아래를 출력
                                fprintf(fp, " +");
                }
                fprintf(fp, "₩n"); // 줄넘김
        }
        fprintf(fp, "|");    // 테두리를 그린다.
        for (i = 0; i < n - 1; i++) { // 수직선을 그리는 루프
                fprintf(fp, " ");

                if (vert[i])
                        fprintf(fp, "|");
                else
                        fprintf(fp, " ");    // 빈 공간
        }
        fprintf(fp, " |₩n");
}

void ofApp::initScreen(void) {        // 게임의 초기화면을 그리는 함수
        gamelogo.draw(50, 0, 900, 600);    // 로고를 그려주고
        switch (selectIdx) {          // 슬롯 선택 인덱스에 따라
        case 0:  // 선택된 슬롯을 그럴듯한 게임처럼, 동적으로 크기를 키워준다.
                playbutton.draw(375, 345, 210, 110); // 즉, 선택되었음을 명시적으로
                rankbutton.draw(380, 450, 200, 100); // 표현하는 것이다.
                exitbutton.draw(380, 550, 200, 100); // 미선택 슬롯은 그대로 출력
                break;
        case 1:
                playbutton.draw(380, 350, 200, 100);
                rankbutton.draw(375, 445, 210, 110);
                exitbutton.draw(380, 550, 200, 100);
                break;
        case 2:
                playbutton.draw(380, 350, 200, 100);
                rankbutton.draw(380, 450, 200, 100);
                exitbutton.draw(375, 545, 210, 110);
        }
}

void ofApp::loadingScreen(int loadframe) { // 로딩화면을 그리는 함수
        int currentframe = ofGetFrameNum();       // 로딩화면은 프레임 범위 별로 동작이 다르다.
        int frameN = currentframe - loadframe;
        // 아이폰 로딩 모양을 본뜬 부분으로, 일반적인 게임의 로딩화면에서 아이디어를 얻었다.
        if (frameN < 15) // 프레임별로 정해진 그림을 애니메이션처럼 보이는 것에 불과함.
                loadone.draw(0, 0, windowWidth, windowHeight);
```

```
        if (frameN >= 15 && frameN < 30)          // 즉, 실질적인 로딩과정은 아니다.
                loadtwo.draw(0, 0, windowWidth, windowHeight);
        if (frameN >= 30 && frameN < 45)
                loadthree.draw(0, 0, windowWidth, windowHeight);
        if (frameN >= 45 && frameN < 60)
                loadfour.draw(0, 0, windowWidth, windowHeight);

        if (frameN >= 60) {         // 60개의 프레임이 지나가면, 로딩을 멈추고, 진짜 게임 플레이
                isload = 0;         // 모드로 들어간다. isload를 해제하고, isgame를 설정한다.
                isgame = 1;
                gamebgm.play(); // 게임 bgm을 실행하며,
                startgame = clock();        // 소요시간 측정을 위한 기준점을 잡아주고 함수 종료
        } // 로딩화면을 도입한 이유는, 자연스러운 게임의 형태를 갖추게 하기 위함.
}

//------ Functions for Implementing Ranking System ------//
// 아래의 함수는, rank.txt 파일을 이용해 랭킹을 구축하는 함수이다.
// 즉, 외부에 저장지를 둠으로써 게임이 종료되었다 다시 실행되어도 기존의 랭킹이
// 유지된다.
void ofApp::createRankList(void) {
        int score;          // 점수 기억
        char temp[15];    // 기록날짜 기억
        FILE*fp = fopen("rank.txt", "rt");    // rank.txt 파일을 연다.
        struct Node* newNode;    // 연결리스트 구축을 위한 변수 선언부
        struct Node* currNode = head;     // currNode는 헤드를 기억

        if (fp == NULL || fscanf(fp, "%d", &numOfRanks) == EOF) {//에러처리
                printf("Ranking File Open Error !₩n");
                return;
        }// 한편, rank.txt 파일은 부재하더라도 게임 실행에 문제가 되지는 않는다.
        // 랭킹이 없는 최초의 게임 실행 시점을 고려하였다.
        while (fscanf(fp, "%s %d", temp, &score) != EOF) {   // 파일을 줄별로 읽어
                newNode = (Node*)malloc(sizeof(Node));   // 노드 동적 할당
                newNode->score = score;
                strcpy(newNode->date, temp);      // 노드 구성
                newNode->link = NULL;

                if (head == NULL) {         // 최초 삽입일 때
                        currentTopRanker = newNode->score;       // 탑 랭커를 기억. 맨 위가 1등이므로
                        head = newNode;
                        currNode = newNode;     // 머리 삽입
                }
                else {    // 노드의 꼬리 삽입과정
                        currNode->link = newNode;
                        currNode = newNode;
                }
```

```
        }
        fclose(fp);
}

void ofApp::writeRankFile(void) {   // 매 플레이 종료될때 rank.txt를 업데이트할 함수
        FILE* fp = fopen("rank.txt", "wt");  // 스트림을 구축한다.
        Node* currNode = head;
        currentTopRanker = currNode->score;       // 탑 랭커를 기억한다.

        fprintf(fp, "%d₩n", numOfRanks);
        while (currNode) {          // rank.txt 파일에 쓴다.
                fprintf(fp, "%s %d₩n", currNode->date, currNode->score);
                currNode = currNode->link;
        }
        fclose(fp);
}

void ofApp::newRank(int score) {   // 매 플레이 종료시마다 점수를 랭킹에 포함함.
        Node* newNode;
        Node* currNode = head;  // 머리에서부터 조회 시작할 것
        Node* prevNode = NULL;

        newNode = (Node*)malloc(sizeof(Node)); // 해당 시점을 기록한다.
        sprintf(newNode->date, "%d/%d,%d:%d", ofGetMonth(), ofGetDay(), ofGetHours(), ofGetMinutes());
        newNode->score = score; // 이름 대신 시점으로 사용자를 구분하는 것

        while (currNode) {          // 적절한 위치를 선형적으로 탐색
                if (score <= currNode->score)
                        break;
                prevNode = currNode;
                currNode = currNode->link;
        }

        newNode->link = currNode;         // 적절한 위치에 삽입
        if (prevNode == NULL)
                head = newNode;
        else
                prevNode->link = newNode;

        numOfRanks++;             // 연결리스트 크기를 업데이트
        writeRankFile();   // rank.txt 파일에 업데이트
}

void ofApp::showRankBoard(void) {          // 랭킹을 화면에 표시하는 함수
        char score[30];
        Node* currNode = head;
```

```
	int i = 0;

	while (currNode) {
		if (i == 5)		// 상위 5등까지만 보여준다.
			break;
		sprintf(score, "%s", currNode->date);		// 출력한다.
		rankplace.drawString(score, windowWidth / 2 - 68, 250 + 52 * i);
		sprintf(score, "%d", currNode->score);
		rankplace2.drawString(score, windowWidth / 2 + 75, 250 + 52 * i);
		currNode = currNode->link;
		i++;
	}
}

//------ Free all heap allocated memories during the process ------//
// 아래의 함수는 프로그램 최초 실행 때 플래그로 인해 분기되어 동적할당되었던
// 모든 변수에 대해 메모리 해제를 진행하는 함수이다. 만약, 최초 상황이 아닌 상황
// 에서 이 함수가 호출되면, 동적할당 해제를 제외하고, 플래그 변수 초기설정 복귀 및
// 각종 인덱스 변수 초기화 작업이 이루어진다.
void ofApp::freeMemory() {
	int i;
	struct Node * delNode;
	struct Node * currNode = head;

	first_line_flag = 1;		// 플래그를 모두 게임 첫 상태로 복구
	last_line_flag = 0;
	endsoundflag = 0;

	if (gameexit) {	// 프로그램 종료 시에만 동적할당 해제. 재사용을 위함.
		for (i = 0; i < HEIGHT; i++)
			delete[] Maze[i];
		delete[] Maze;
		for (i = 0; i < HEIGHT; i++)
			delete[] Visit[i];
		delete[] Visit;
		delete[] lines;
		delete[] shortRoute;
		delete[] allRoute;
		delete[] rows;
		delete[] vert;
		delete[] horiz;
		while (currNode) {
			delNode = currNode;
			currNode = currNode->link;
			free(delNode);
		}
```

```
            head = NULL;
        }
        linesSize = 0;
        sIdx = aIdx = isdfs = isOpen = 0;  // 인덱스 변수 및 플래그 변수 복구
        allroute.clear();    // ofPolyLine도 비워준다.
        shortroute.clear();
}
```

4. 각 변수에 대한 설명

주석을 통해 전체적인 코드의 흐름 및 상세 내용을 설명하였다. 자, 이제는 위의 코드에서 핵심적으로 사용되는, 보다 정확하게 표현하면, ofApp.h 파일 내에 선언된 변수 중 프로그램을 동작시키는 주요 변수들에 대해 설명한다.

(1) 자료구조와 관련된 변수

- linesSize 변수는 미로를 ofDrawLine 함수를 통해 화면 상에 그릴 때 사용하는 '직선을 저장한 배열' lines의 인덱스를 나타내는 변수이다.
- sIdx 함수는 DFS에서 스택으로 사용되는 shortRoute 스택 배열의 top 지칭 변수로 사용되는 인덱스 변수이다.
- aIdx 변수는 사용자(쥐)의 전체 이동 경로를 저장하는 allRoute 배열의 크기 및 인덱싱을 위해 사용되는 인덱스 변수이다.

(2) 좌표 상의 주요 포인트를 지칭하는 변수

- HEIGHT와 WIDTH 변수는 미로의 크기를 지칭하는데 사용되는 변수로, maze.maz 파일을 읽어 들이는 부분에서 등장하여 경로 계산 과정, 게임 플레이 과정 등의 프로그램 전반에 걸쳐 지속적으로 사용되는 변수이다.
- coordX와 coordY 변수는 사용자(쥐)의 게임 상에서 '현재 위치'를 나타내기 위한 좌표 변수이다.
- targetX와 targetY 변수는 목적지(치즈)의 게임 상에서의 위치를 나타낸다.

(3) 각 함수들의 구현에 필요한 보조 변수들

- selectIdx 변수는 게임의 메뉴화면(초기화면)에서 슬롯의 선택을 프로그램이 구분할 때 사용하는 변수이다. 0은 play버튼을, 1은 rank버튼을, 2는 exit버튼을 각각 지칭한다.
- currentframe 변수는 게임 플레이 직전의 '로딩 화면' 구현 간에 시간 딜레이를 표현하기 위해 사용되는 frame 수 저장 변수이다. OFS의 내장함수 ofGetFrameNum 함수의 반환값을 저장한다.
- playtime 변수는 게임이 플레이될 때 플레이 소요 시간 및 점수 계산 때 사용되는 변수이다.
- loadframe 변수 역시 앞선 currentframe과 함께 로딩 화면 구현 시 딜레이를 생성하기 위한 변수이다. 마찬가지로 ofGetFrameNum 함수의 반환값을 저장하며, 딜레이 계산에서 '기준점'으로 사용된다.
- head 변수는 struct Node * 타입의 포인터형 변수로, 선언형태에서 알 수 있듯, 랭킹시스템을 위한 연결리스트의 머리를 나타내는 변수이다.
- numOfRanks 변수는 연결리스트의 노드 개수를 기억하는 변수이다.
- printScore 문자열 배열은 게임 플레이 종료 후의 계산된 점수를 문자열로 변환할 때 사용된다. OFS의 ofTrueTypeFont 클래스, 그 중에서도 drawstring 함수를 사용하기 위해 설계되었다.
- date 문자열 배열은 게임 플레이 종료 후의 종료 시점을 "월/일,시간:분"의 형태로 기억하기 위해 선언되었다. 이 역시 마찬가지로 drawString 기능을 사용하기 위함이다.
- currentTopRanker 변수는 현재 시점에서의 랭킹 시스템 최고 등수 점수를 기억하기 위한 변수이다. 이는, 게임 상에서 1등 점수가 기록되는 경우 몇 가지 이벤트를 발생시키기 위해 마련되었다.

(4) 프로그램 흐름을 제어하기 위한 플래그 변수

- isOpen 변수는 maze.maz 파일이 정상적으로 열렸는지를 알리는 플래그 변수이다. 따라서 readFile 함수에서 set된다.
- first_heap_alloc_flag 변수는 변수 이름 그대로 첫 번째 힙 동적 할당을 지시하기 위한 플래그 변수이다. 매 게임 플레이마다 일일이 자료구조 구축을 위한 동적 할당이 이루어지면, 매 게임 종료마다 메모리 해제를 한다고 해도 결국 얼마 지나지 않아 힙 공간이 꽉 차버리는 문제가 발생하는데, 이를 해결하기 위해 기존에 할당된 자료구조를 그대로 재사용한다. 이를 위해선, 각 함수에서 최초 실행과 그 이후의 실행을 분기해야 하는데, 이 기능을 바로 이 변수가 수행한다.
- first_line_flag 변수는 임의의 미로를 생성하는 makemazefile 함수 및 그 예하의 함수에서 사용하는 플래그 변수로, 미로의 첫 번째 줄을 만들 때의 상황만을 따로 분기하는데 사용한다. 반복문을 사용하여 미로를 만들 때, 미로의 테두리의 존재 때문에 첫 번째 줄은 반복에 포함시킬 수가 없는데, 이를 하나의 함수에서 처리하기 위해 이 플래그를 이용하는 것이다.
- last_line_flag 변수는 first_line_flag 변수와 유사한 이유로 선언된 변수이다.
- Initscrflag 변수는 게임의 초기(메뉴)화면을 표시할지 말지 여부를 draw 함수 및 각 종 게임 플레이 관련 함수에 전달한다.
- isload 변수는 로딩 화면을 표시할지 여부를 결정하는 플래그 변수이다.
- isgame 변수는 게임 플레이를 시작할지 여부를 결정하는 플래그 변수이다.
- upflag 변수는 게임 플레이 중에 키보드의 '상' 방향키가 눌렸을 때를 처리하기 위한 플래그 변수이다. downflag, rightflag, leftflag 변수도 마찬가지의 기능을 수행한다.
- resetinitflag 변수는 게임 플레이 상에서 '10초 간격으로 목적지 위치가 변경되는 기능'을 수행하는데 사용되는 플래그이다.
- reachtargetflag 변수는 게임 플레이 상에서 목적지에 사용자가 도달하였는

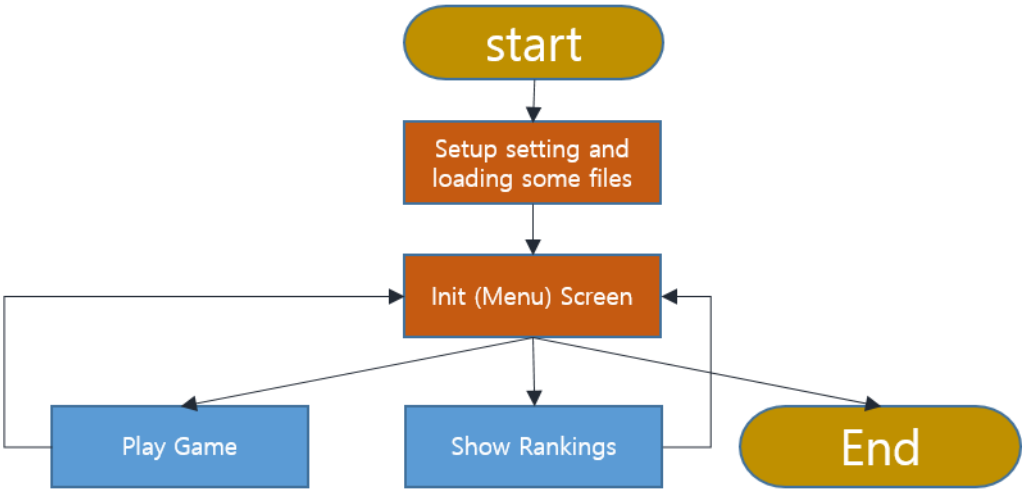
지 여부를 알리는 플래그이다.

- endsoundflag 변수는 게임 플레이 종료 상황에서 일부 몇 가지 이벤트를 생성하기 위해 '종료 시점'에 한한 동작 수행을 구현하고자 마련된 플래그 변수이다.
- isdfs 변수는 게임 플레이 종료 시 DFS 함수를 수행하여 최단 경로를 계산한 후, 해당 경로를 화면 상에 표시하기 위해 사용되는 플래그 변수이다.
- gameexit 변수는 게임 종료를 알리는 플래그이다. 이 플래그를 통해 freeMemory 함수의 기능이 분화된다.
- rankscrflag 변수는 랭킹 시스템을 화면 상에 표현할지 여부를 알리는 플래그 변수이다.

5. 자료구조 및 알고리즘에 대한 상세한 설명

'Where is my Cheese?' 프로그램의 **자료구조와 알고리즘에 대한 상세한 설명에 앞서 우선 프로그램의 흐름을 설명**하겠다. 프로그램의 흐름에 대한 설명이 선행되면 조금 더 자연스러울 것이라 생각한다.

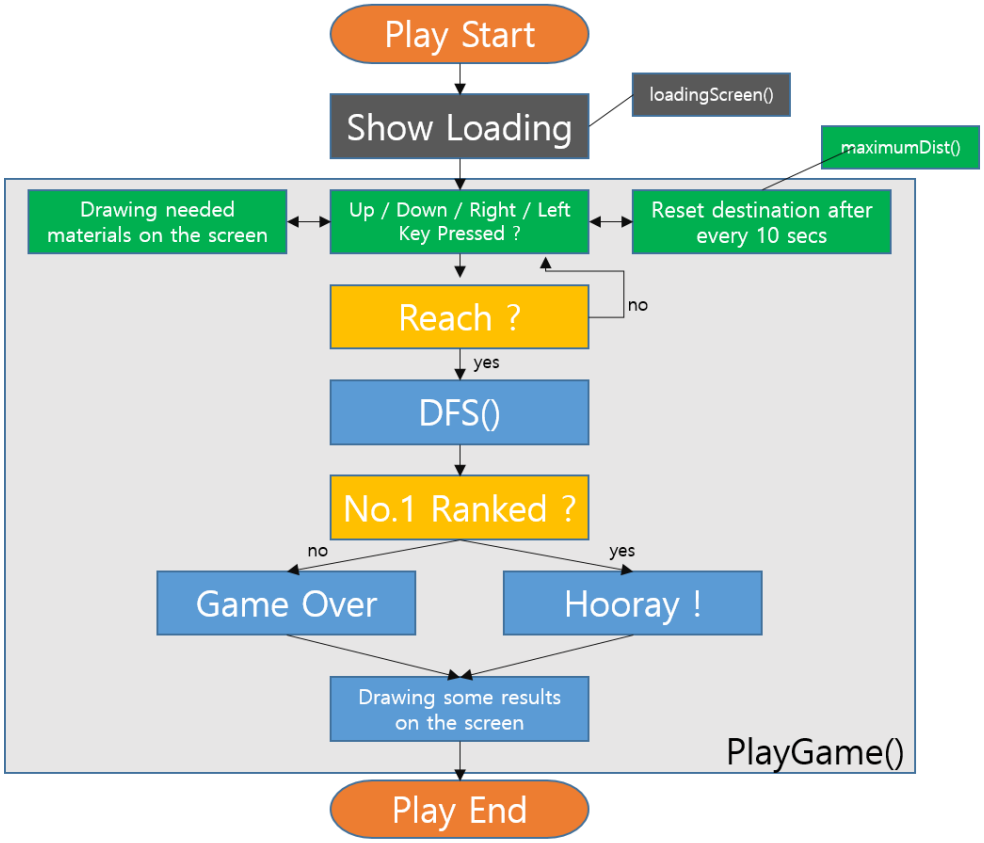
우선, 다음의 그림은 이 미로탈출 게임의 가장 포괄적인 흐름을 보여주는 1차 플로우 차트이다. 프로그램이 시작하면, 우선 가장 먼저 OFS 필수 설정을 포함하여 몇 가지 기초적인 설정을 setup 함수에서 진행한다. 이 부분에서 모든 그림, 사운드, 폰트 등의 데이터 파일에 대한 load 과정도 진행된다.



setup 설정이 끝나면, 이어서 initscrflag의 1 초기화로 인해 초기 메뉴화면이 자

동적으로 화면 상에 표시된다. (draw 함수와 initScreen 함수를 이용.) 이 상태에서 keyPressed 함수에서는 사용자의 방향키 '상', '하' 입력 여부를 이용해 슬롯 선택을 가능케 한다. 슬롯은 play, rank, exit의 3가지로, play를 선택하고 엔터키를 누르면 게임 플레이가 시작되고, rank를 선택하고 엔터키를 누르면 랭킹 시스템 표시가, exit를 선택하고 엔터를 입력하면 프로그램 종료가 이루어진다. 해당 과정이 위의 플로우 차트에 표시되어 있다.

우측의 플로우 차트는 게임 플레이 과정에 대한 플로우 차트이다. 로딩화면이 표시된 이후에 PlayGame 함수가 어떠한 흐름으로 게임 플레이를 통제하는지 간략하게 표시되어 있다. 물론 완벽히 정확한 흐름을 나타낸 것은 아니지만, 크게 보았을 때 우측의 플로우 차트와 동일한 흐름으로 게임 플레이가 진행된다.



나머지 Rank 슬롯 선택 시의 랭킹 시스템 표시와 Exit 슬롯 선택 시의 종료 구현 부분은 흐름이 간단하므로 플로우 차트는 생략한다. 오히려 중요하면서, 아직 언급하지 않은 흐름은 다음의 흐름이다.

[랭킹 시스템 구축] -> [(게임 플레이 슬롯 선택 시) 임의 미로 생성 및 maze.maz 파일 출력] -> [maze.maz 파일을 읽어 미로를 프로그램에 구축] -> [게임 플레이] -> [랭킹 시스템 업데이트]

의 과정이 정말 중요하다. 즉, 간단하게 말하면, 메뉴 화면에서 사용자가 play 슬롯을 선택하면, 게임 플레이에 앞서 로딩 화면이 나타나고, 또 그에 앞서 플레

이를 위한 랭킹, 임의 미로 생성 및 자료구조 구축 과정이 일어난다는 점이다. 또한, 플레이 종료 후 랭킹 업데이트 과정도 일어난다는 점이다.

자, 이제 **본격적으로 프로그램의 자료구조에 대해서 설명**한다. 이 프로그램에는 자료구조가 다양하게 사용되는데, 각 자료구조들의 목적 및 쓰임이 다르다. 따라서 목적 별로 항목화하면 다음과 같다. (일부 자료구조는 수업 시간에 다룬 내용을 그대로 차용하였고, 일부 자료구조는 변형을 가했다. 아예 획기적이고 새로운 자료구조 도입은 사실상 현재 수준에서는 불가능하기 때문에 변형을 가해 새로움을 더했다. ex) 엘러 알고리즘에 사용되는 vert, horiz나 allRoute 배열 등)

(1) 임의 미로를 생성하는데 사용되는 자료구조

이번 프로젝트 간에 임의 미로를 제작하는 알고리즘은, 수업 시간에 다루었던 Eller's Algorithm이다. 이때, Eller 알고리즘은 조금 더 적용하기 편하게 하고자 본인은 rows, vert, horiz라는 세 배열을 선언하였다. 세 배열 모두 '하나의 열 별로 미로를 차근차근 작성한다는 Eller 알고리즘의 특성'을 살리고자 '하나의 열' 별로 기능한다. rows배열은 각 칸 별 set의 값을 기록하고, vert는 수직선의 유무, horiz는 수평선의 유무를 기록한다.

(2) maze.maz를 기초로 미로를 구축하는데 사용되는 자료구조

이 역시 수업 시간에 다루었던 내용과 유사하게 구현하였다. Maze와 Visit이라는 2차원 배열을 도입한다. Maze는 maze.maz 파일을 문자 단위로 읽어 들여 미로를 인식하는 데에 사용되고, 이어서 게임 플레이 종료 후 최단 경로를 검색할 때 DFS 알고리즘에서도 사용된다. Visit 배열은 DFS 알고리즘 구현 시 특정 칸의 '방문 여부'를 기록한다. shortRoute라는 스택도 사용된다. 이 역시 DFS 알고리즘 구현에 쓰이는 스택으로, 최단 경로를 기록한다. lines 배열은 Maze 행렬을 기반으로 미로를 프로그램 화면에 출력할 때에 사용하는 배열이다.

(3) 랭킹 시스템을 구축하는데 사용되는 자료구조

간단하게 단순 선형 연결리스트를 사용하였다. head라는 public 변수가 바로 연결리스트의 머리이다.

(4) 사용자의 전체 경로를 기억하는 allRoute 선형 배열

플레이 간의 사용자의 모든 자취를 기록하는 배열 allRoute도 마련하였다.

이어서 **프로그램의 핵심 알고리즘들에 대해서 설명**한다. 다양한 알고리즘이 사용되었기 때문에 이 역시 항목화하였다. 자료구조 설명과 마찬가지로, 알고리즘 역시 수업 시간에 다룬 개념과, 본인이 직접 설계한 간단한 알고리즘들이 존재한다.

(1) 임의 미로를 생성하는 Eller's Algorithm

수업시간에 활용하고 실습한 내용을 차용하였다. Eller 알고리즘은 다음과 같이 요약할 수 있다.

"미로를 2차원으로 바라보지 않고, 1차원으로 바라보아, 미로의 한 열을 가리키는 배열과, 해당 열의 수평 벽 배열, 수직 벽 배열, 총 3가지의 열 관련 배열들(Column Related Arrays)을 도입하는 방식"

조금 더 디테일하게 확인해보자. 우선, vert와 horiz 배열은 1과 0을 요소로 가짐으로써, 벽이 존재하는 상황을 1로, 그렇지 않은 상황을 0으로 처리한다. 배열의 이름에서 유추할 수 있듯이 vert는 수직 벽면을, horiz는 수평 벽면을 나타낸다. maze는 '같은 순간'의 열의 set정보를 기억하고자 만들어진 배열이다.

randomizeV 함수는 set정보에 따라서 vert배열을 무작위로 구성하는 함수이다. randomizeH 함수는 set정보에 따라서 horiz배열을 무작위로 구성하는 함수이다. randomize 함수는 최초의 미로 작성 순간에서 set정보에 따라 vert배열을 무작위로 구성하는 함수이다. (첫 번째 열에 대해서만 기능 수행. 즉, first_line_flag 활성화 시에만 작동한다.) drawMaze 함수는 미로를 파일 상에 그려주는 함수로, 이 역시 첫 번째 열 출력 상황은 따로 처리해야 하기 때문에 분기를 필요로 한다. 코드 단위의 분석의 위의 코드 첨부 부분에서 진행했기 때문에 생략한다.

즉, 다시 요약하면, 미로를 2차원 행렬로 바라보지 않고, Column 별로 분해하여 각 열에 대해서 일일이 무작위 seed를 배정하여 배정된 정보에 따라서 열을 그리는 것이다. 이때 핵심은, '2차원'으로 바라보지 않고, set 배열, 수직 벽 배열, 수평 벽 배열을 다 따로 다룬다는 점이다. 이러한 Eller 알고리즘의 시/공간 복잡도를 판단하는 것은 간단하다. 복잡해 보이지만, 결국 vert, horiz, maze 배열은 모

두 사용자로부터 입력 받은 행, 열 정보에 대해서만큼만 연산이 수행된다. 즉, 점근적인 시선에서 height * width만큼만 대입 연산을 진행한다. 따라서 **시간 복잡도는 O(height*width)**이다.

(2) 최단 경로를 계산하는 Depth First Search

이 역시 코드 레벨의 분석은 위의 주석 첨부 부분에서 진행했으므로 알고리즘 자체에 대한 설명을 진행하겠다. 특정 정점을 조회할 때, 인접한 미방문 정점을 재귀적으로(스택을 사용하여) 방문하여 최종적으로 그래프 전체를 탐색하는 것을 골자로 한다. 이번 DFS 구현 간에는, 재귀호출이 아니라, 실재적인 스택 자료구조를 도입하였다. 앞서 언급했던 것처럼, shortRoute 스택은 최단 경로를 기억한다.

한편, 이번 구현은 인접 행렬 방식으로 구현되었기 때문에 DFS의 시간 복잡도는 O(n^2)임을 추정할 수 있다. 공간 복잡도의 경우도 마찬가지인데, 사실상 할당 연산이 이루어지는 것은 배열 초기화 작업만이기 때문에 가장 크기가 큰 Visit 배열의 크기, 즉, 행*열의 사이즈만큼의 공간 복잡도가 나타날 것이다. 즉, O(height*width)이다.

참고로, 최단 경로 탐색에 DFS를 선택한 이유는, BFS에 비해서 적어도 이 미로 찾기 상황에서는 더 효율적이기 때문이었다.

(3) 현재 위치에서 가장 먼 꼭지점을 찾는 알고리즘 (자체 구현)

이 게임 프로그램에 '게임 특유의 박진감'을 추가하기 위해 '10초 마다 목적지가 변경되는 기능'을 도입했다고 앞서 설명하였다. 이 부분에서, 목적지는 바로 현재 사용자의 위치를 기준으로 4개의 꼭지점 중 가장 먼 꼭지점을 찾는 알고리즘이 사용된다.

이 부분에 있어서 본인은, '현 위치를 기준으로 4개의 꼭지점에 대해서 각각 DFS를 수행하여 가장 최단 경로가 긴 꼭지점을 선택하는 방식'과 '현 위치를 기준으로 4개의 꼭지점까지의 각각의 절대적 거리(피타고라스 정리를 이용한) 비교

해 최장 거리의 꼭지점을 고르는 방식' 중 고민을 했는데, 시각적 측면과 시간복잡도적 측면, 구현 난이도적 측면을 모두 고려하여 후자의 방식을 택했다. 따라서 이 알고리즘은, 크기 4인 배열을 만들고, 각 배열은 4개의 꼭지점을 나타내게 한 후, 배열을 순회 및 비교하여 현 위치를 기준으로 가장 거리가 먼 지점을 검색하는 방식으로 진행된다. 이는 코드의 maximumDist 함수에 의해 구현된다. 시간복잡도는 고정된 연산이 진행되므로 O(1)이다.

(4) 사용자의 방향키 입력에 따른 움직임을 나타내는 알고리즘 (자체 구현)

이 부분 역시 섬세한 분기를 요구하는 부분이었기 때문에 따로 알고리즘으로 분류하였다. 우선, 각각의 방향 키 이동에 대한 제한 사항을 두는 것이 중요했다. 미로는 한정적인 nxn의 크기를 지닌 공간이기 때문에 각 테두리 부근에서는 이동할 수 없기 때문이다.

또한, 주어진 미로를 토대로 진행하고자 하는 방향에 벽이 존재하면 이동을 불가하도록 설정하는 부분도 필요하다. 이는 각 방향 별로 **if (Maze[1 + coordY][1 + coordX + 1] == ' '), if (Maze[1 + coordY][1 + coordX - 1] == ' '), if (Maze[1 + coordY - 1][1 + coordX] == ' '), if (Maze[1 + coordY + 1][1 + coordX] == ' ')** 라는 조건문들을 토대로 구현할 수 있었다.

단순하지만, 이 미로탈출 프로그램의 가장 핵심적인 기능을 수행하기 때문에 견고하게 만드는 것이 중요했다. 즉, 한 부분에서도 분기 오류가 존재하지 않도록 구현하는 것이 중요했다. 시간복잡도는, 고정된 크기의 연산이 진행되므로 O(1)이다.

(5) PlayGame 함수

이 함수는 사실 엄밀한 의미의 알고리즘은 결코 아니지만, (4)와 마찬가지로 섬세한 분기와 각 상황에 대한 명령 처리를 일일이 해주어야 하는 부분이었기 때문에 알고리즘적 특성이 있어 분류하였다. 상세한 설명은 상단의 주석 첨부 부분에 있는 내용으로 대체한다.

6. 각 함수에 대한 설명

이 파트는 ofApp.h 헤더파일을 기반으로 설명한다. 단, 비슷한 목적을 위한 함수들끼리 묶어서 설명을 진행한다. 물론, 코드 레벨의 분석은 주석 첨부 부분에서 이미 진행했으므로, 함수의 기능과 목적에 대한 상세한 설명을 진행한다.

(1) Openframeworks 기초 함수들

- setup() : 프로그램의 기초 초기 설정 사항을 수행한다. 이 프로그램에서는, 각 종 이미지, 사운드 파일을 업로드하는 역할을 중점적으로 수행한다.
- draw() : 프로그램의 화면에 프레임 단위로 요구사항을 그리는 함수이다. 각 종 플래그 변수에 의해 통제되어 특정 상황에서 적합한 그림을 표시하도록 설계되었다.
- keyPressed() : 특정 key가 입력되었을 때 프로그램의 동작을 제어하는 함수이다. 이 역시 마찬가지로 다양한 플래그 변수에 의해 통제되어 특정 상황에서 적합한 행동을 수행하도록 설계되었다.

(2) 임의 미로를 생성하는 함수들

- makemazefile() : 임의 미로를 생성하는 함수이다. 아래의 예하 함수들을 반복적으로, 그리고 분기적으로 호출하여 임의 미로를 생성한다.
- randomize(int n, int *set) : 최초의 미로 작성 순간에서 set정보에 따라 vert 배열을 무작위로 구성하는 함수이다.
- randomizeV(int n) : set정보에 따라서 vert배열을 무작위로 구성하는 함수이다.
- randomizeH(int n, int *set) : set정보에 따라서 horiz배열을 무작위로 구성하는 함수이다.
- drawMaze(FILE *fp, int n) : 미로를 maze.maz 함수에 줄 단위로, 문자 단위로 출력하는 함수이다.

(3) 미로를 구축하는 함수들

- readFile() : maze.maz 파일을 읽어 정보를 토대로 미로를 구축하는 함수이다. Maze 배열과 lines 배열을 구축하는 것이 핵심이다.
- pathFind(int I, int j, int flag) : 미로를 화면 상에 그리기 위한 lines 배열을 구축할 때, 화면 상에 적절하게 그려질 수 있도록 값을 조정해주는 반복적 코드를 모듈화한 함수이다.
- DFS() : 게임 플레이 종료 시마다 초기 시작점을 기준으로 해당 종료 시점에서의 목적지까지의 최단 경로를 계산하는 함수이다.
- dfsdraw() : DFS의 결과를 화면에 출력한다. allRoute도 출력하는 기능을 한다.

(4) 랭킹 시스템을 구축하는 함수들

- createRankList() : rank.txt 파일을 읽어 연결리스트를 구축하는 함수이다.
- writeRankFile() : rank.txt 파일에 새로운, 업데이트된 연결리스트를 출력하는 함수이다.
- newRank(int score) : 연결리스트에 새로운 점수를 오름차순 순서로 삽입하는 함수이다.

(5) 게임 플레이 통제 및 GUI 구축에 이용되는 함수들

- initScreen() : 게임의 초기 메뉴화면을 통제하고, 그리는 함수
- loadingScreen(int loadframe) : 게임 플레이 시작 전 로딩 화면을 표시하는 함수. 사실, 로딩함수는 아무런 기능도 하지 않고 오로지 시간을 딜레이시

키며 화면에 애니메이션만 출력한다. 이러한 잉여 기능을 추가한 이유는, 바로 다름 아닌, 보고서 초반에 언급한 '게임적 특성'을 살리기 위함이다.

- showRankBoard() : 랭킹을 화면에 표시하는 함수
- PlayGame() : 게임 플레이 1회의 흐름을 전체적으로 통제하는 함수. 즉, 게임 플레이의 CPU 역할을 한다고 보면 된다.
- maimumDist() : 게임의 박진감을 추가하기 위해 도입된 '10초 간격으로 리셋되는 목적지의 위치' 기능을 구현하는데에 사용되는 최장거리 꼭지점 탐색 함수이다.
- drawclock() : 게임 플레이 시 우측 하단의 시계 흐름 애니메이션을 표현하기 위한 함수이다.

(6) freeMemory()

본인은 최초 프로그램을 설계할 때에, 각 플레이마다 시작점에서 자료구조 구축을 위해 동적할당을 진행하고, 종료시점에서 동적 할당 해제를 진행하는 방식으로 프로그램을 작성하였었다. 하지만 워낙 다루고 있는 자료구조의 크기가 크고 방대하다 보니 이러한 방식으로는 Re-Game을 몇 차례 진행하지 못하고 프로그램이 터지는 문제가 발생했다. 따라서 이를 해결하기 위해 **'기할당된 메모리의 재사용'이라는 개념을 도입**했다. 그리고 이 freeMemory함수는 해당 기능 구현에서 가장 핵심 역할을 수행한다. gameexit라는 플래그 변수와, first_heap_alloc_flag라는 플래그 변수를 이용해, 최초 자료구조 구축 시점에만 동적할당을 진행하고, 나머지 시점에서는 메모리 재사용을 하도록 한 후, 프로그램 종료 시점에서만 메모리 해제를 진행하도록 말이다. 이 함수가 바로 그 역할을 수행한다.

7. 창의적으로 구현한 부분

본인은 어린 시절 미술에 관심이 많았었는데, 이러한 점을 살려 이번 미로탈출 프로그램 구현에 있어서 **그래픽 인터페이스를 '실제 상용 게임스럽게' 구현하고자 노력한 부분**이 창의성이 있는 부분이라고 생각한다.

게임의 모든 측면에 있어서 이러한 부분에 가장 크게 신경을 썼다. 미로탈출의 클리셰인 '쥐'와 '치즈'를 고른 후, 이 게임이 왜 만들어졌고, 사용자가 왜 목적지에 빠르게 도달해야하는지를 각 화면 별 우스꽝스럽고 재밌는, 동시에 어느 정도 마감이 잘 되어있는 그림들을 사용하여 **최대한 사용자에게 직관적으로 다가갈 수 있게** 노력하였다.(포토샵과 기존 자료들을 이용하여 그림을 구성했고, 여러 시행착오로 OFS를 만져가며 적절한 위치를 찾았다.)

無기능의 로딩화면 도입 역시 이러한 이유로 설명이 가능하다. 최대한 이 프로그램의 그래픽 요건이 실제 사용자로 하여금 *'컴퓨터공학 실습 프로그램'이 아닌, '게임'으로서 받아들일 수 있도록* 신경썼다.

앞서 **게임 플레이 점수가 1등에 해당하는 경우 특별한 이벤트가 발생**한다고 언급하였는데, 이러한 부분도 나름 신박한 구현이었다고 생각한다.

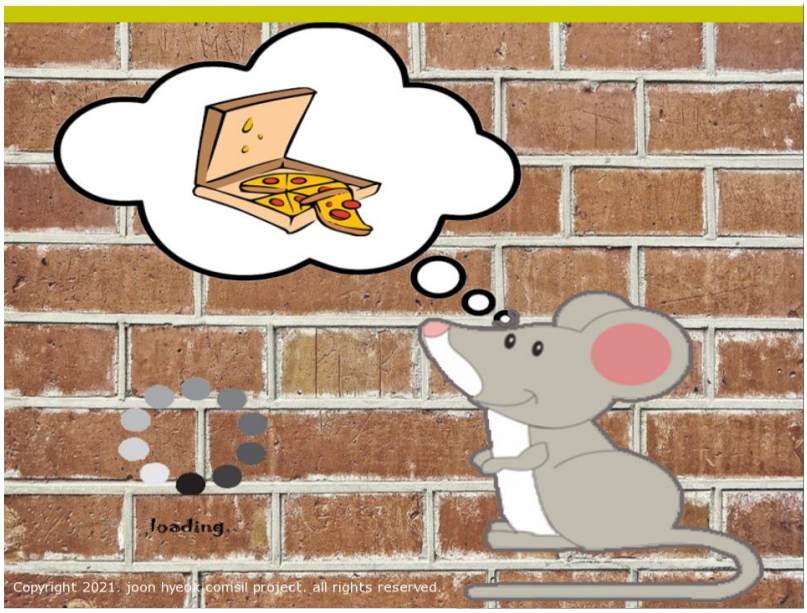
또한, 메뉴에서 슬롯 선택 시 **'포커스가 맞추어진 슬롯의 경우 크기가 커지는, 실제 게임스러운 특성'**을 살린 부분도 창의적인 부분이라고 조심스럽게 자평한다. 물론, 이 역시 다른 기존의 게임을 플레이한 경험에서 비롯되는 것이긴 하지만 말이다.

그래픽 측면에서의 노력을 제외하고 보았을 때는, **'10초 마다 목적지 위치가 바뀌는 기능'의 도입**이 창의적인 부분이 될 수 있다고 본다. 단순히 고정된 목적지로 이동하는 일반적인 미로탈출은 사용자에게 재미나, 긴장감을 줄 수 없다고 판단하여 도입하였다.

8. 프로젝트 실행 결과 캡쳐

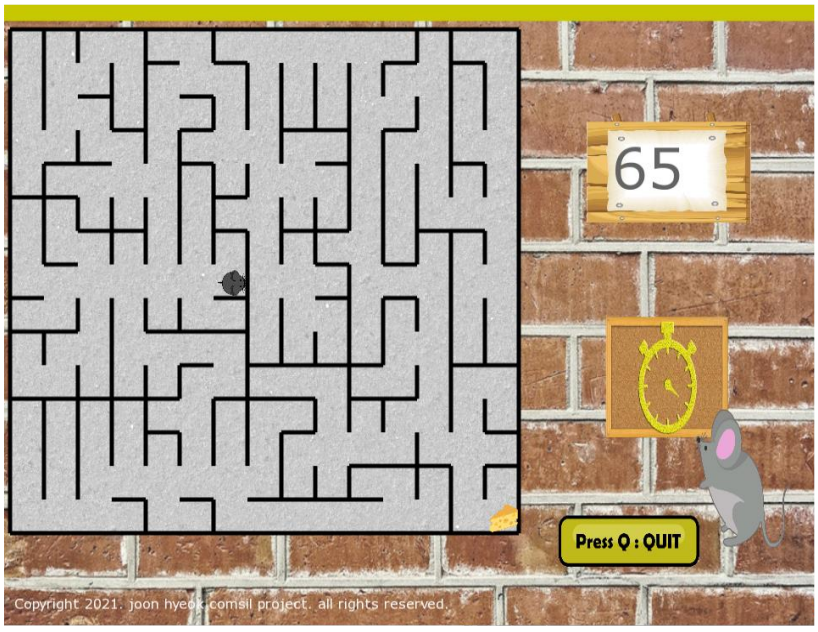
(1) 초기 (메뉴) 화면

(선택된 슬롯의 크기가 다른 슬롯에 비해 미세하게 크게 하여 사용자가 이를 눈으로 알아챌 수 있게 하였다.)



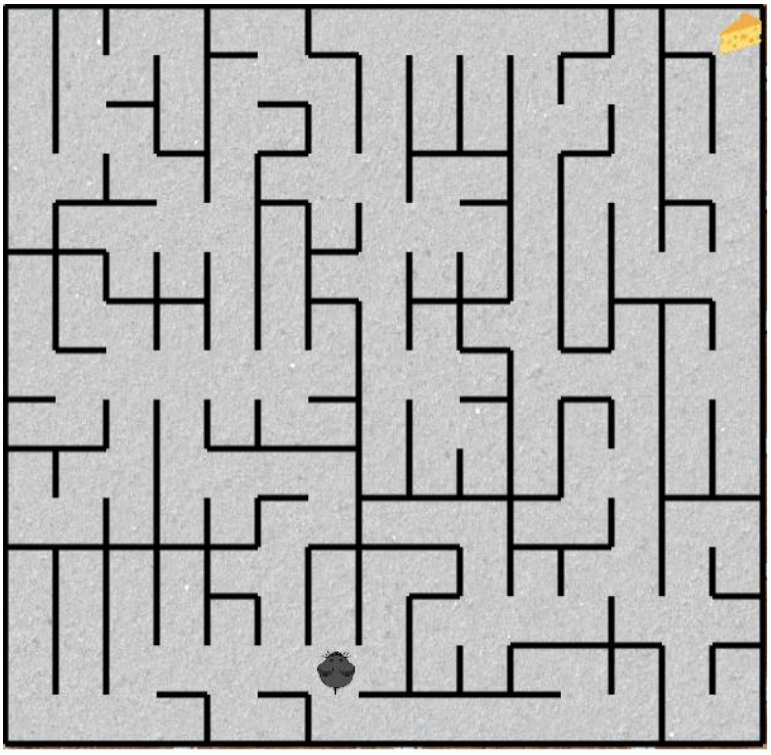
(2) 로딩 화면

(좌측 하단의 원형 무늬는, 아이폰의 로딩 애니메이션에 착안하여 유사하게 구현하였다.)



(3) 플레이 화면

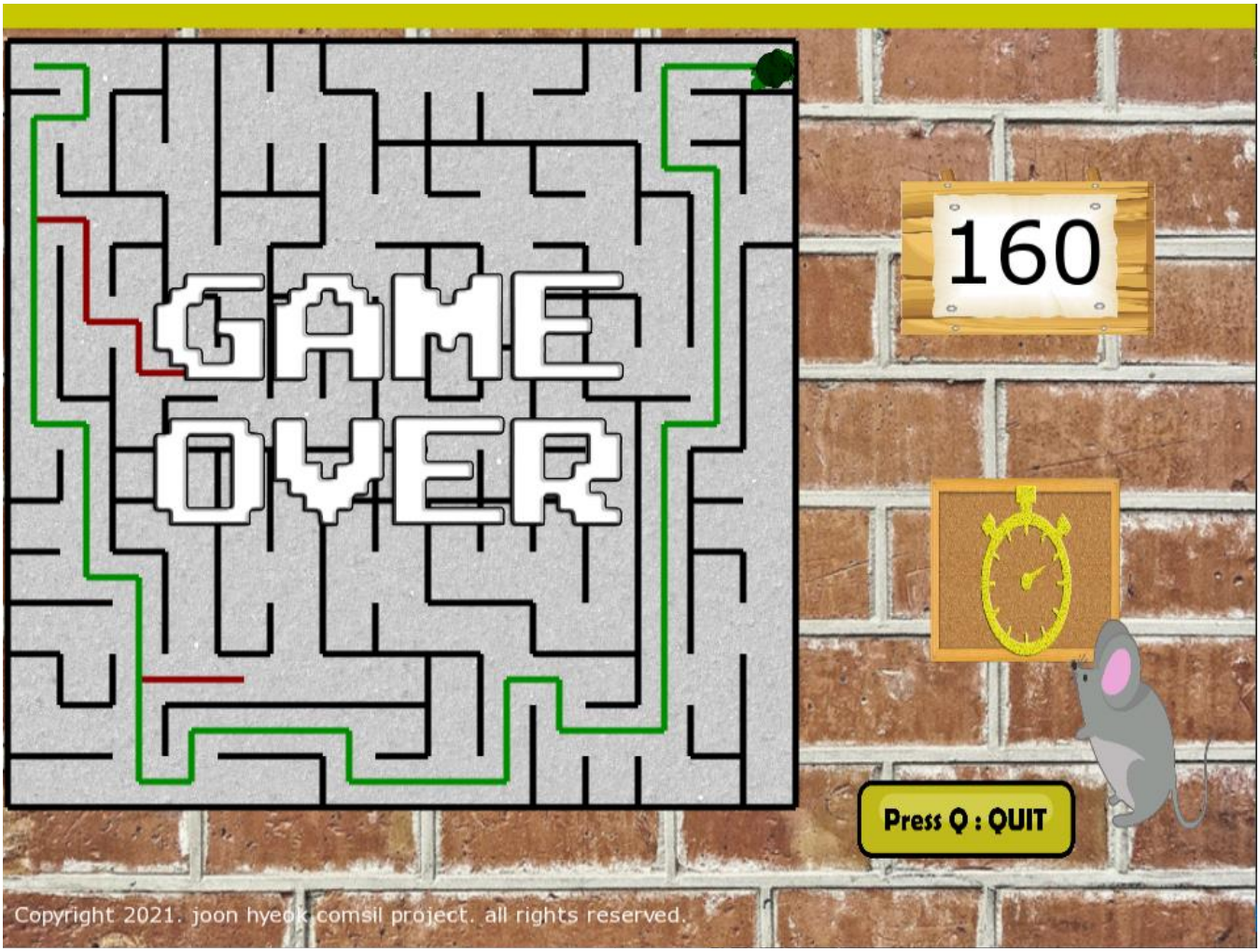
(미로가 있으며, 미로 내에서 쥐가 움직인다. 쥐는 움직이는 방향을 바라본다. 우측 하단에는 강제 종료 버튼이 있으며, 중간에는 시계 애니메이션, 상부에는 LIVE 스코어보드가 있다.)

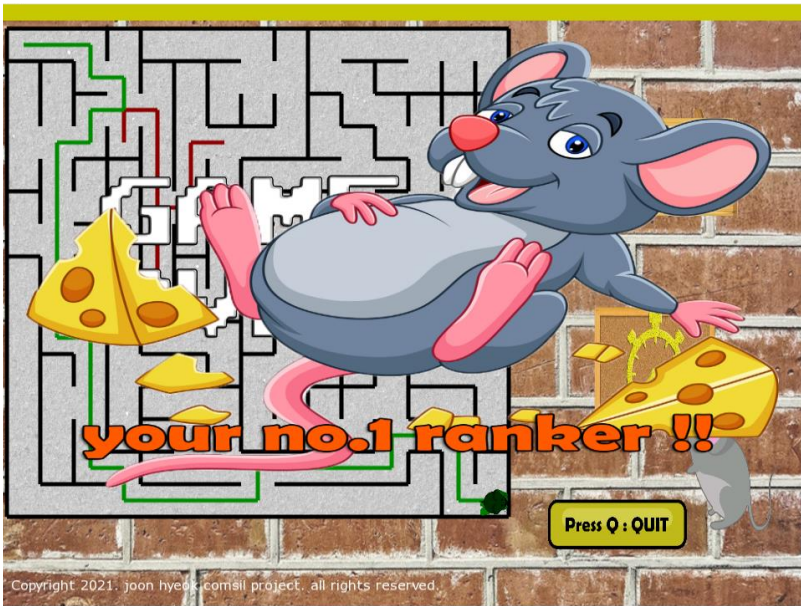


(4) 10초 간격으로 치즈가 이동한 모습

(첨부 이미지의 쥐의 위치에서 쥐와 가장 멀리 떨어진 치즈의 위치는 우측 상단 꼭지점이다.)

(5) 목적지에 도달했을 때의 화면 (초록색은 최단 경로, 빨간색은 쥐의 자취이다.) **(아래)**





(6) 1등 시의 특별 이벤트

(오로지 최초 점수와 1등 점수에 한 해서만 이 이벤트 그림이 화면에 표시된다. hooray!라는 사운드 효과도 특수하게 첨부되어 있다.)

(7) 랭킹 시스템

(엔터키를 누르면 자동적으로 다시 초기 메뉴 화면으로 돌아간다.)

9. 느낀 점 및 개선 사항

(1) 느낀 점

기획부터 직접 설계 및 구현, 버그 수정 및 개선 등의 과정을 스스로 총괄해 본 경험이 사실 이번 컴퓨터공학설계 및 실험1 기말 프로젝트가 처음이었다고 해도 과언이 아니었다. 그러한 일련의 과정 속에서 가장 크게 느낀 점은, **"본인이 스스로 재미를 느껴야 개발 과정이 전혀 힘들다 느껴지지 않고, 나아가 결과도 만족스럽다."**는 점이었다. 그렇다, 본인은 이번 프로젝트를 하면서 정말 **재밌었다.** 사실 처음에 프로젝트를 시작하는 순간에는 조금은 막막하고, 두려웠던

것도 사실이다. 하지만 주제를 정하고, 어떤 방향으로 설계를 할 것이며, 어느 부분에서 '나만의 특징'을 살릴 것인지를 마음 속으로 정하고 나니, 그 이후의 코딩 과정은 며칠을 꼬박 셌음에도 결코 힘들다 느껴지지 않았다. 앞으로 컴퓨터 공학도로서 학업을 지속하는 과정에서 이러한 '흥미'는 분명 좋은 영향으로 작용할 것이라 생각한다.

한편, '커다란 프로그램'을 만들 때에는 **시간 복잡도와 공간 복잡도를 충분히 고려해야 한다는 진리를 다시금 생각해볼 수 있던 시간**이기도 했다. 최초에 이 미로탈출 프로그램을 작성할 때에는 본인은 매 게임 플레이마다 시작 지점에서 동적 할당으로 자료구조를 구축하고, 종료 지점에서 동적 할당 해제 작업을 해주는 방식으로 구현하였는데, 이렇게 구현하니 게임을 5번 이상 연속으로 플레이하니 프로그램이 터지는 문제가 발생했다. 워낙 사용하는 메모리 공간이 방대하다 보니 여러 번 힙 할당을 하면 프로그램이 감당하지 못하는 것이었다. 따라서 이를 해결하기 위해 앞서 지속적으로 언급한 것처럼 '메모리 재사용' 개념을 도입하였는데, 이러한 일련의 과정에서 복잡도 고려 자세가 상당히 중요하다는 것을 다시 깨달을 수 있었다. 시간 복잡도적인 측면에서도 최대한 간단하게 필요한 기능들을 구현할 수 있도록 노력했다. '10초마다 목적지가 Re-Set되는 기능'에서 DFS를 사용하지 않고 절대 거리를 계산한 것 등이 바로 이러한 노력의 일환이다.

(2) 개선 사항

그럼에도 몇 몇 부분에서 아쉬운 부분이 존재한다. 무엇보다도 **3D 구현을 하지 못한 점**이다. 최초에 프로그램을 설계할 때, 본인은 3D 그래픽 구현도 고민해보았다. 실제로 OFS가 3D 관련 클래스도 제공한다고 알고 있었기에 조금 시간 투자를 하고 연구하면 해볼만하지 않을까 생각했다. 하지만 이를 시도하는 과정에서, 아직 3D 관련 개발을 해보지 않은 탓인지, 아니면 시간의 압박이 커서 그랬는지, 이유가 무엇이었든 결과적으로 3D 구현이 힘들다고 판단해 구현하지 못했다. 2021년도에 이 'Where is my Cheese?'와 같은 2D 그래픽 게임은 실제로 사용될 가능성이 낮기 때문에 이 부분이 조금 아쉽다.

또한, 랭킹 시스템을 구축하는 과정에서, 사용자로부터 문자열을 입력 받아 사용자 본인의 이름 혹은 별명을 기록하는 방식을 초기에 구현하고자 하였는데, OFS에서 **화면 혹은 팝업창을 통해 문자열을 사용자로부터 입력 받는 방법을 발견하지 못해 결국 구현할 수 없었던 점**이 꽤나 아쉽다. 분명 방법이 존재할 것 같은데, OFS가 국내/국외 모두에서 '정말 자주 사용되는 언어'는 아니다 보니 자료가 많이 부족하여 이 방법을 결국 찾지 못하였다.

따라서 만약 나중에 본인이 이 프로젝트 프로그램을 개선할 수 있는 기회나 능력이 생긴다면, 3D 그래픽으로 구현하고, 사용자로부터 이름을 입력 받을 수 있는 기능을 추가하고 싶다. 그 외로는, 조금 더 난이도를 높이기 위해 미로를 높이/폭이 30 이상인 커다란 형태로 바꾸는 방법도 가능할 것이다.