2021년 2학기 컴퓨터공학설계및실험1 최종 프로젝트 보고서

1. 프로젝트 목표

Maze 프로젝트를 추가 구현하여, 사용자가 직접 미로를 탈출하는 플레이가 가능하도록 한다. 출발 지점과 도착 지점은 DFS/BFS와 동일하게 가장 왼쪽 상단의 칸에서 가장 오른쪽 하단의 칸으로 한다.

2. 실험 환경

macOS Monterey Version 12.1

MacBook Air (M1, 2020)

8GB

Xcode Version 13.1

3. 변수

프로젝트에 사용된 변수는 모두 <ofApp.h>에 선언되어 있다.

```
struct Line{
    float startX;    /* x coordinate of the line's starting point */
    float startY;    /* y coordinate of the line's starting point */
    float endX;      /* x coordinate of the line's end point */
    float endY;      /* y coordinate of the line's end point */
};

enum Direction{       /* refers to neighboring cells */
    UP, DOWN, LEFT, RIGHT, NONE
};

struct Vertex{
    /* Can or cannot go ... */
    bool up;
    bool down;
    bool left;
    bool right;
    /* Position of this vertex */
    int x;
    int y;
    /* This vertex is visited or not */
```

```
    bool visited;
    /* Parent position */
    Direction parent;
};
        int imageHEIGHT;                /* Total HEIGHT */
        int imageWIDTH;                 /* Total WIDTH */

        int mazeHEIGHT;                 /* Maze HEIGHT */
        int mazeWIDTH;                  /* Maze WIDTH */

        vector<vector<char>> image;     /* Saves all information from the text file. */
        vector<Line> walls;             /* Saves wall information */
        vector<vector<Vertex>> maze;    /* Saves maze cell information */

        stack<Vertex*> DFS_stack;       /* DFS stack */
        queue<Vertex*> BFS_queue;       /* BFS stack */

        vector<Line> searched_path;     /* All path searched by algorithm */
        vector<Line> shortest_path;     /* Shortest path */

        int isOpen;                     /* File opened(1) or not(0). */

        ofRectangle FILEbutton;         /* FILE button */
        int FILEbuttonClicked;          /* FILE button clicked(1) or not(0). */

        ofRectangle DFSbutton;          /* DFS button */
        int isDFS;                      /* DFS on(1) or off(0). */

        ofRectangle BFSbutton;          /* BFS button */
        int isBFS;                      /* BFS on(1) or off(0). */

        ofRectangle QUITbutton;         /* QUIT button */

        ofRectangle PLAYbutton;         /* PLAY button */
        int isPLAY;                     /* PLAY on(1) or off(0). */

        vector<Line> play_path;         /* Path searched by user */
        float time;                     /* Time spent after play */
        bool reached;                   /* Flag: reached destination or not */
        int currX;                      /* User's current x position */
        int currY;                      /* User's current y position */
```

변수 선언에 필요한 구조체와 각 변수에 대한 설명은 주석으로 작성했다.

4. 함수

프로젝트에 사용된 함수들은 다음과 같다. 간단한 설명을 주석으로 작성했다.

```
        /* Open .maz file, save maze information.
```

```
        Call processImage() to save information of each cell. */
    bool readFile(ofFileDialogResult openFileResult);

    /* Initialize the information of each cell. */
    void processImage();

    /* Free dynamically allocated memories. */
    void freeMemory();

    /* Draw menu bar containing FILE, DFS, BFS, PLAY, QUIT. */
    void drawMenu();

    /* DFS */
    bool DFS();

    /* BFS */
    bool BFS();

    /* Draw the path found by DFS. */
    void drawDFS();

    /* Draw the path found by BFS. */
    void drawBFS();

    /* Used during DFS/BFS to save the searched paths. */
    void getSearchedPath(Vertex* curr);

    /* After DFS/BFS, find and save the shortest path. */
    void getShortestPath();

    /* After specific direction key is pressed,
        operates according to the corresponding key. */
    void PLAY(Direction way);

    /* Draw the path as the user progressed. */
    void drawPLAY();
```

추가된 플레이 모드를 on/off하기 위해 버튼을 추가했다. 이 버튼의 위치와 크기를 setup() 함수에서 설정해주고, isPLAY 플래그를 0으로 초기화했다.

```
void ofApp::setup(){
    . . .

    PLAYbutton.set(190, 10, 50, 20);
    isPLAY = 0;

    . . .
}
```

플레이 버튼을 클릭하면 isPLAY 플래그를 0으로 set해서 플레이 모드를 켜도록 했다. 그리고 목적지에 도착하지 않은 상태로 초기화하고, 현재 위치를 출발 지점(가장 왼쪽 상단 칸)으로 초기화한 뒤, 경로를 저장하는 play_path를 초기화한다. 미로의 모든 칸을 방문하지 않은 칸으로 초기화하되 출발 지점은 반드시 방문한 상태이므로 방문한 칸으로 변경한다. 그 다음 시간 측정을 위해 ofResetElapsedTimeCounter()를 호출해 이 시점부터 시간이 측정되도록 한다.

이 함수에서의 시간복잡도 및 공간복잡도는 visited 정보를 초기화하는 데 필요한 O(mazeHEIGHT * mazeWIDHT)이다.

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

    /* PLAY button clicked */
    if (PLAYbutton.inside(x, y)) {
        if (isOpen) {
            if (isDFS || isBFS) {
                cout << "Please turn off DFS/BFS first." << endl;
            }
            else {
                isPLAY = !isPLAY; //toggle PLAY
                if (isPLAY) {
                    /* Initialize status */
                    reached = false;
                    currX = currY = 0;

                    /* Initialize path */
                    play_path.clear();

                    /* Mark all nodes as not visited. */
                    int i, j;
                    for (i = 0; i < mazeHEIGHT; i++) {
                        for (j = 0; j < mazeWIDTH; j++) {
                            maze[i][j].visited = false;
                        }
                    }

                    /* Starting point is always visited.   */
                    maze[0][0].visited = true;

                    /* timer start */
                    ofResetElapsedTimeCounter();
                }
            }
        }
        else
            cout << "You must open file first" << endl;
    }

    . . .
```

```
}
```

draw() 함수를 수정해 isPLAY 플래그가 1이라면 파일이 열렸을 경우 drawPLAY() 함수를 호출해 사용자가 진행한 대로 경로를 그려준다. 한편, 출발 지점과 도착 지점의 위치를 눈에 띄게 나타내기 위해 미로를 불러왔을 때 출발 지점과 도착 지점에 색을 칠했다.

```
void ofApp::draw(){

    . . .

    /* Draw maze. */
    if (isOpen) {
        . . .

        /* Starting point */
        ofRectangle startRect;
        startRect.x = 22.5;
        startRect.y = 62.5;
        startRect.width = 35;
        startRect.height = 35;
        ofSetColor(ofColor::lightPink);
        ofDrawRectangle(startRect);

        /* End point */
        ofRectangle endRect;
        endRect.x = 40 * mazeWIDTH - 17.5;
        endRect.y = 40 * mazeHEIGHT + 22.5;
        endRect.width = 35;
        endRect.height = 35;
        ofSetColor(ofColor::lightGreen);
        ofDrawRectangle(endRect);
    }

    . . .

    /* PLAY */
    if (isPLAY) {
        if (isOpen) {
            drawPLAY(); //Draw the path as the user progressed.
        }
        else
            cout << "You must open file first" << endl;
    }

    . . .

}}
```

추가적으로 구현된 PLAY() 함수에서는 drawPLAY() 함수는 다음과 같다. 도착 지점에 도달하지 않은 경우에는 플레이를 시작한 이후 시간이 얼마나 흘렀는지를 보여주고, 도착 지점에 도달한 경우에는 도달하는 데까지 걸린 시간을 보여주도록 했다. 그리고 play_path에 저장된 대로 사용자가 진행한 경로를 그려주었다.

이 함수에서의 시간복잡도 및 공간복잡도는 O(play_path.size())이다. 즉, 사용자가 특정 시점까지 거쳐온 거리의 길이를 n이라고 하면 O(n)이다.

```
void ofApp::drawPLAY(){
    if (reached) {
        ofSetColor(ofColor::lightGreen);
        ofDrawBitmapString(to_string(time), 320, 25);
    }
    else {
        time = ofGetElapsedTimef();
        ofSetColor(ofColor::lightGreen);
        ofDrawBitmapString(to_string(time), 320, 25);
    }

    ofSetColor(ofColor::black);
    for(int i = 0; i < play_path.size(); i++) {
        ofDrawLine(play_path[i].startX, play_path[i].startY, play_path[i].endX, play_path[i].endY);
    }
}
```

플레이 모드에서 사용자가 입력하는 키에 따라 작동하도록 keyPressed() 함수 내부에 다음과 같이 코드를 추가했다. isPLAY 플래그가 1로 set되었을 경우에만 작동하며 입력받은 키에 따라 PLAY() 함수에 해당 방향을 인자로 전달해주도록 했다. 만약 이미 도착 지점에 도달한 경우에는 더 이상 움직일 수 없도록 했다.

```
void ofApp::keyPressed(int key){
    ...
    if (key == OF_KEY_RIGHT) {
        if (isPLAY && !reached) {
            if (maze[currY][currX].right) {
                Direction tmp = RIGHT;
                PLAY(tmp);
            }
        }
    }
    if (key == OF_KEY_LEFT) {
        if (isPLAY && !reached) {
            if (maze[currY][currX].left) {
                Direction tmp = LEFT;
                PLAY(tmp);
            }
        }
    }
```

```
    if (key == OF_KEY_UP) {
        if (isPLAY && !reached) {
            if (maze[currY][currX].up) {
                Direction tmp = UP;
                PLAY(tmp);
            }
        }
    }
    if (key == OF_KEY_DOWN) {
        if (isPLAY && !reached) {
            if (maze[currY][currX].down) {
                Direction tmp = DOWN;
                PLAY(tmp);
            }
        }
    }
}
```

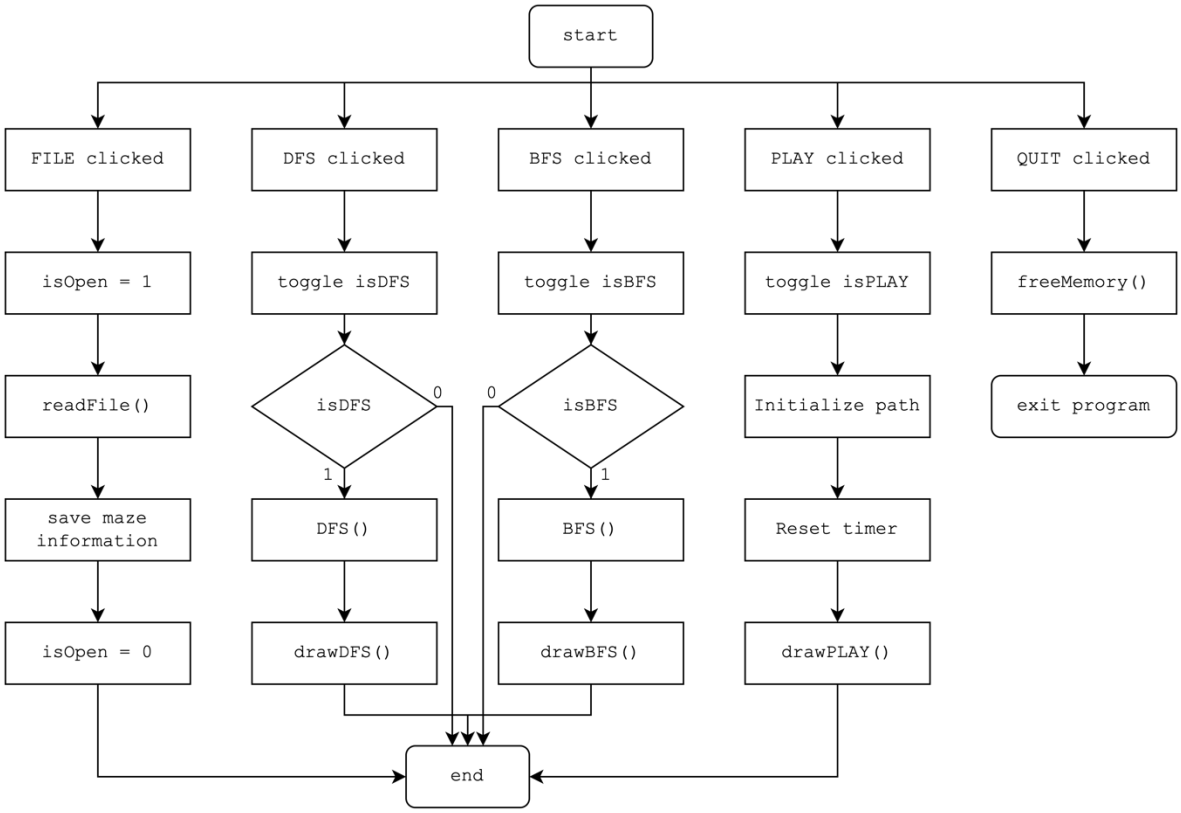
추가적으로 구현된 PLAY() 함수에서는 인자로 전달받은 방향에 따라 해당 방향으로 현재 위치를 이동하는 코드를 구현했다. 만약 새로 뻗어나가는 경로라면 해당 경로를 play_path에 push해 해당 경로를 drawPLAY() 함수에서 그려줄 수 있도록 했다. 만약 기존에 거쳐왔던 경로로 돌아가는 경우라면 play_path에서 pop해서 그려지지 않도록 했다. 즉 play_path는 LIFO(last in first out)의 스택이다. PLAY() 함수에서는 이동 후 만약 도착 지점에 도달하면 reached 변수를 1로 set했다.

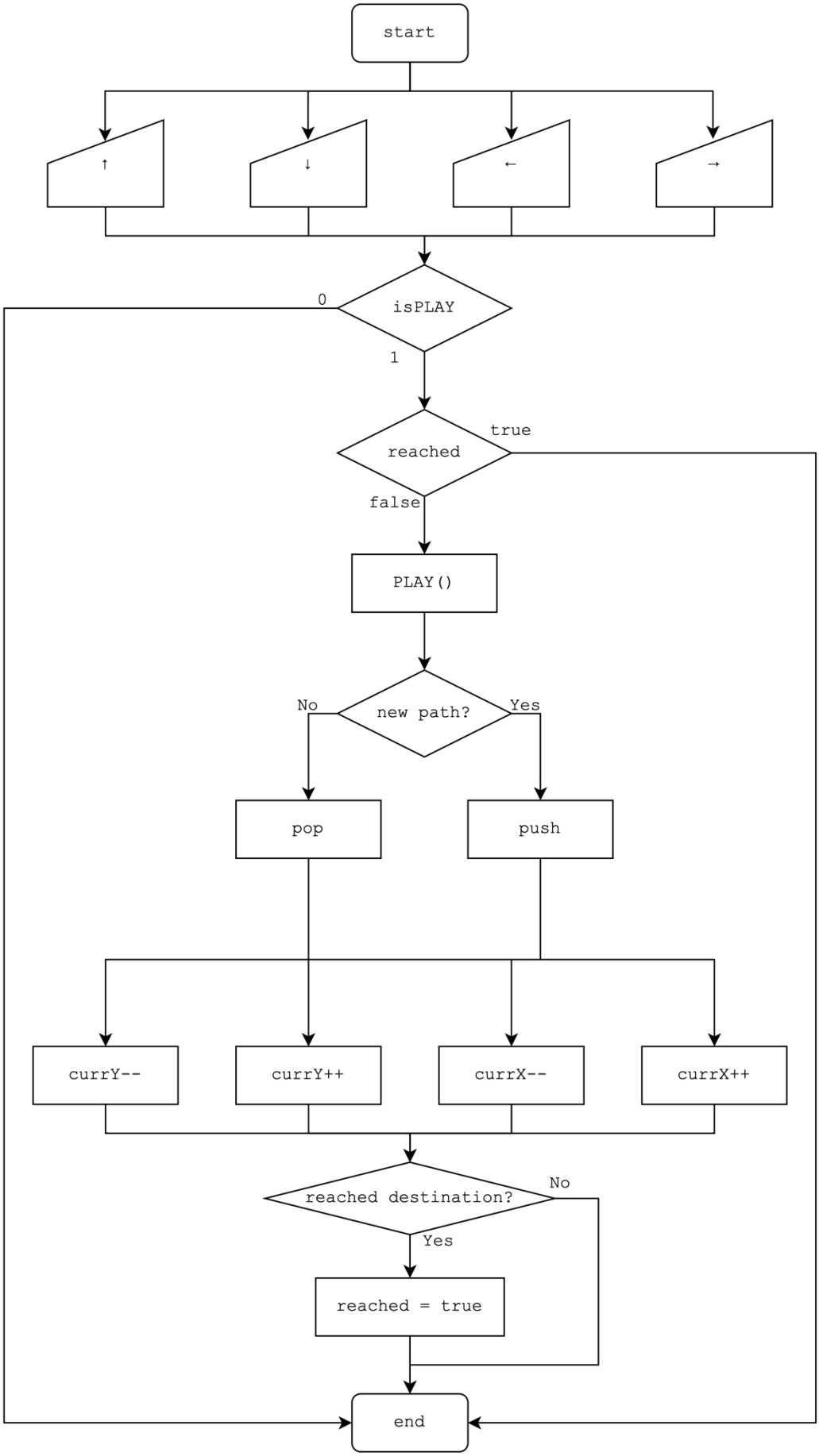
```
void ofApp::PLAY(Direction way){
    Line tmp;

    switch(way) {
        case UP:
            /* 돌아가는 길이라면 경로를 지운다. */
            if (maze[currY-1][currX].visited) {
                play_path.pop_back();
                maze[currY][currX].visited = false;
            }
            /* 새로운 길이라면 경로를 추가한다. */
            else {
                tmp.startX = 40 * currX + 40;
                tmp.startY = 40 * currY + 80;
                tmp.endX = 40 * currX + 40;
                tmp.endY = 40 * (currY - 1) + 80;
                play_path.push_back(tmp);
                maze[currY-1][currX].visited = true;
            }
            currY--;
            break;

        case DOWN:
```

```
            /* 돌아가는 길이라면 경로를 지운다. */
            if (maze[currY+1][currX].visited) {
                play_path.pop_back();
                maze[currY][currX].visited = false;
            }
            /* 새로운 길이라면 경로를 추가한다. */
            else {
                tmp.startX = 40 * currX + 40;
                tmp.startY = 40 * currY + 80;
                tmp.endX = 40 * currX + 40;
                tmp.endY = 40 * (currY + 1) + 80;
                play_path.push_back(tmp);
                maze[currY+1][currX].visited = true;
            }
            currY++;
            break;

        case LEFT:
            /* 돌아가는 길이라면 경로를 지운다. */
            if (maze[currY][currX-1].visited) {
                play_path.pop_back();
                maze[currY][currX].visited = false;
            }
            /* 새로운 길이라면 경로를 추가한다. */
            else {
                tmp.startX = 40 * currX + 40;
                tmp.startY = 40 * currY + 80;
                tmp.endX = 40 * (currX - 1) + 40;
                tmp.endY = 40 * currY + 80;
                play_path.push_back(tmp);
                maze[currY][currX-1].visited = true;
            }
            currX--;
            break;

        case RIGHT:
            /* 돌아가는 길이라면 경로를 지운다. */
            if (maze[currY][currX+1].visited) {
                play_path.pop_back();
                maze[currY][currX].visited = false;
            }
            /* 새로운 길이라면 경로를 추가한다. */
            else {
                tmp.startX = 40 * currX + 40;
                tmp.startY = 40 * currY + 80;
                tmp.endX = 40 * (currX + 1) + 40;
                tmp.endY = 40 * currY + 80;
                play_path.push_back(tmp);
                maze[currY][currX+1].visited = true;
            }
```

```
                currX++;
                break;

            default:
                ;
        }

        /* reached end point */
        if (currX == (mazeWIDTH-1) && currY == (mazeHEIGHT-1)) {
            reached = true;
        }
}
```

5. Flow chart




start
↑
↓
←
→
isPLAY
0
1
reached
true
false
PLAY()
new path?
No
Yes
pop
push
currY--
currY++
currX--
currX++
reached destination?
No
Yes
reached = true
end

6. 자료구조/알고리즘 및 시간/공간 복잡도

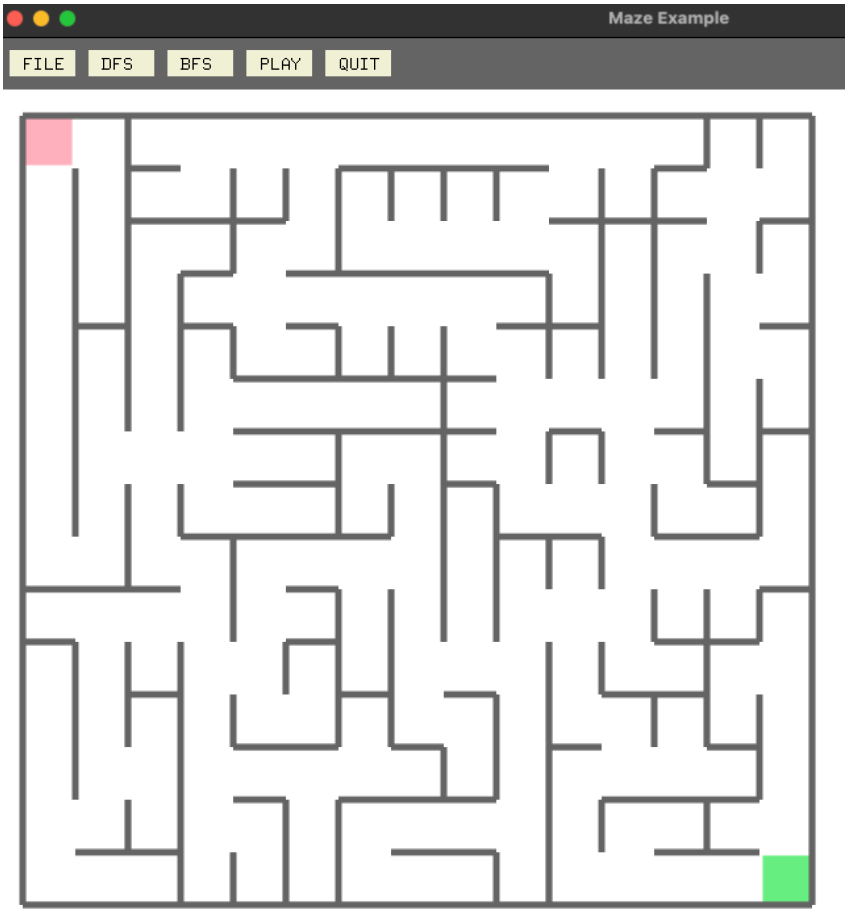
위에서 언급한 대로, 스택 형식으로 사용자가 거쳐온 path를 저장했다. 즉 새로 뻗어나가는 경로일 경우 해당 경로를 play_path에 push해 해당 경로를 drawPLAY() 함수에서 그려줄 수 있도록 하고, 기존에 거쳐왔던 경로로 돌아가는 경우라면 play_path에서 pop해서 그려지지 않도록 했다. 사용자가 거쳐온 path를 화면에 그려주는 함수의 시간 및 공간복잡도는 사용자가 지나온 path의 길이가 n일때 O(n)이었고, 플레이 모드를 시작할 때 미로의 모든 칸을 초기화해줄 때 O(mazeHEIGHT * mazeWIDTH)이었다. (이 때 mazeHEIGTH는 미로의 높이, mazeWIDTH는 미로의 너비이다.)

7. 창의적 구현

기존에 구현했던 미로 프로젝트는 DFS, BFS 알고리즘으로 계산된 경로를 관찰할 수 있었으나 사용자가 직접 미로를 탈출해보는 플레이는 불가능했다. 따라서 키보드의 방향키를 이용해 사용자가 직접 탈출 경로를 찾아보는 플레이 모드를 구현하고, 이 모드를 켤 수 있는 버튼을 새로 생성해 플레이가 가능하도록 했다. 또한 플레이 시 시작 지점과 도착 지점을 분명히 알 수 있도록 눈에 띄게 색깔을 칠해주었다.

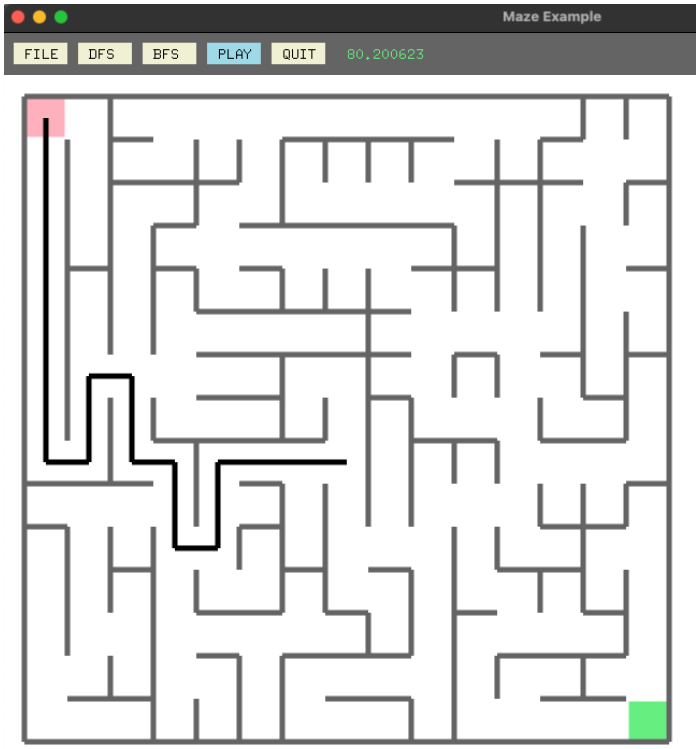
8. 프로젝트 실행 결과

PLAY 버튼 추가 생성

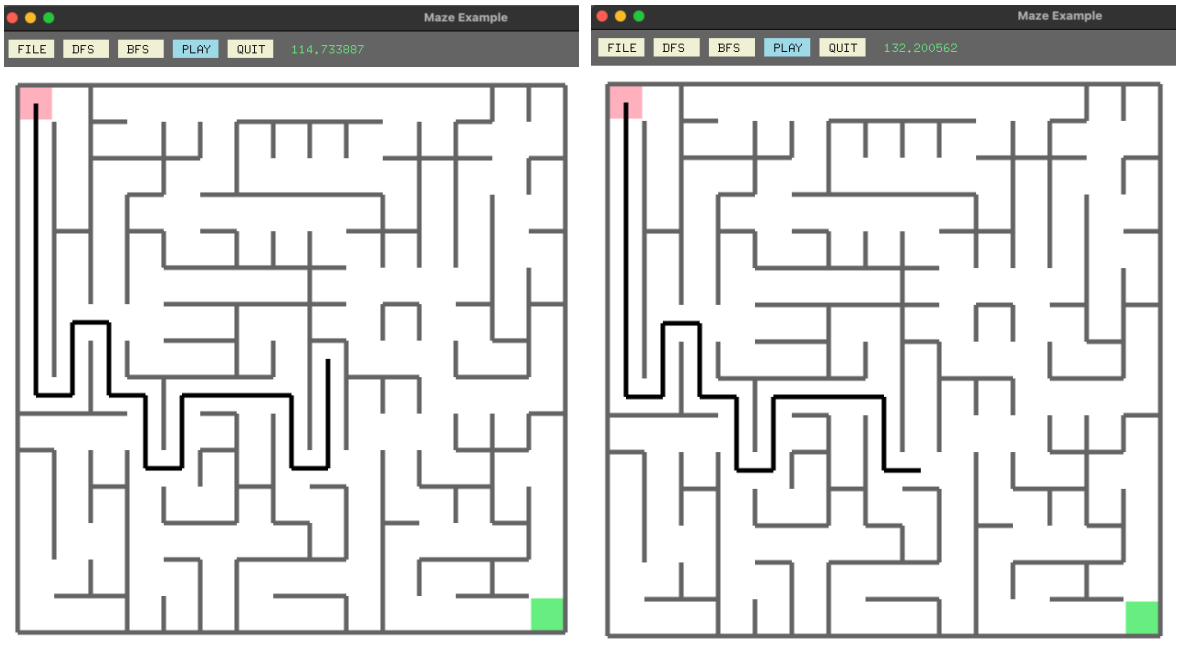


시작 지점과 도착 지점 표시

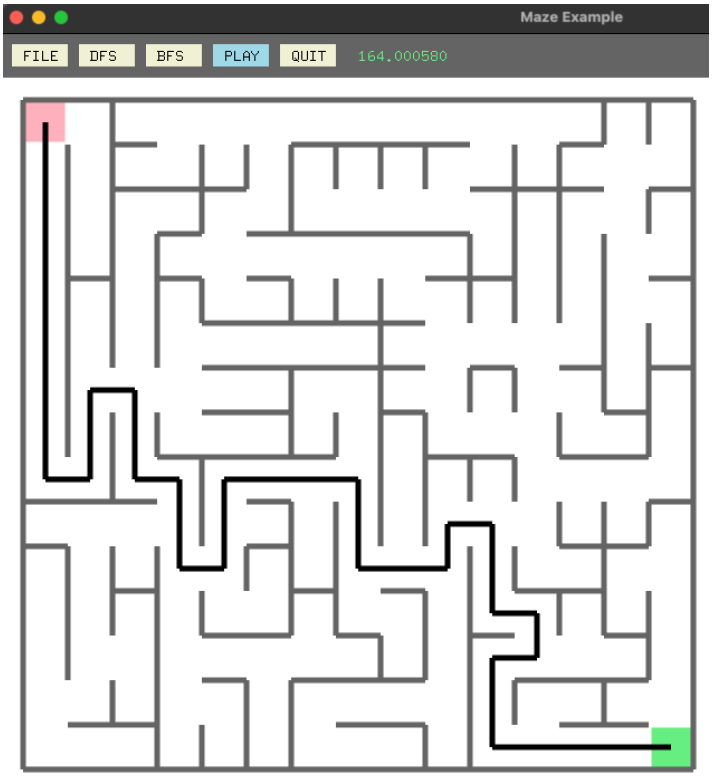
PLAY를 누른 경우, 타이머 시작



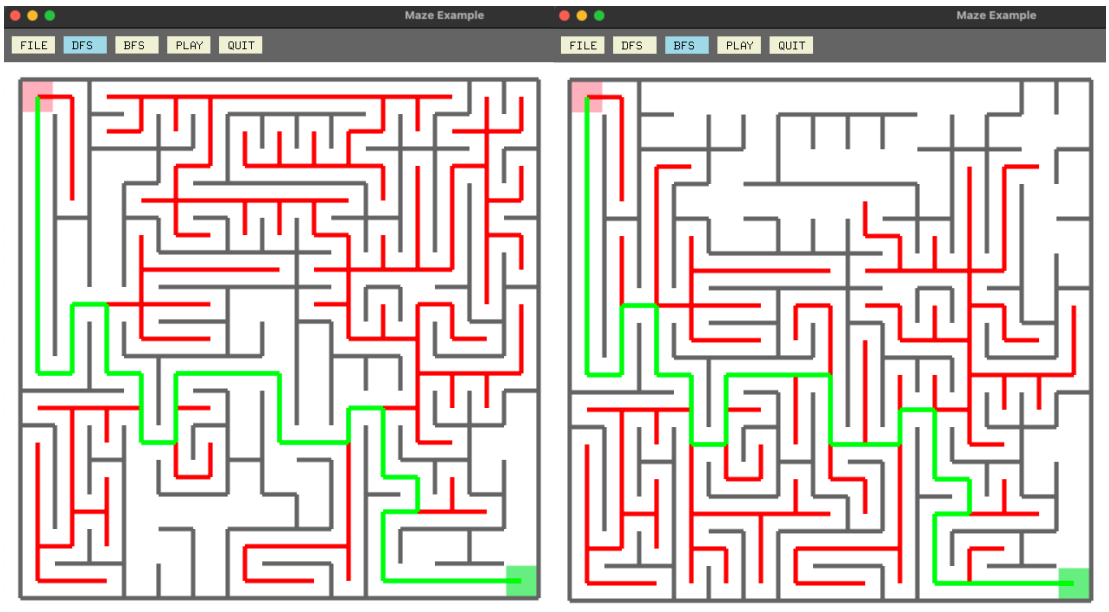
사용자가 키보드의 방향키를 눌러 조작할 수 있다.



왔던 길을 되돌아가는 경우 (예를 들어 막다른 길을 만나서 되돌아가는 경우) 그려졌던 길이 지워진다.



도착 지점에 도달하면, 타이머가 멈추고 방향키를 눌러도 더 이상 이동하지 않는다.



기존 프로젝트와 마찬가지로 DFS, BFS는 정상적으로 작동한다.

9. 느낀점 및 개선 사항

stdout으로 출력하던 프로그램 구현과 달리 openFrameworks를 이용해 의도하는 내용물을 출력하는 것이 어렵게 다가왔다. 조금 더 시간을 갖고 다양한 addon을 살펴보며 응용한다면 더 다채로운 결과물을 얻을 수 있을 것 같다. 또한 현재 프로젝트에서는 시간을 기록하고 이를 이용하지는 않는데, 테트리스 프로젝트에서 랭킹 시스템을 구현한 것처럼 기록을 저장하는 것도 추가적으로 구현하면 기능이 풍부해질 것으로 예상된다. 이러한 개선 사항에서 살펴볼 점은 미로 파일 단위로 기록할 것인가 혹은 미로 크기와 소요한 시간의 비율로 기록할 것인가, 유저의 이름/닉네임을 어떻게 입력받을 것인가, 기록을 별도 파일에 어떤 방식으로 저장할 것인가, 어떤 자료 구조를 이용할 것인가 등이 있다.