CSE3013 (컴퓨터공학 설계 및 실험 I)
미로 프로젝트

서강대학교 컴퓨터공학과 박수현 (20181634)

제 2분반; 담당교수: 서강대학교 컴퓨터공학과 장형수

1 설계 문제 및 목표

미로 프로젝트에서는 미로를 만드는 프로그램과 미로를 보여주는 GUI 프로그램, 그리고 미로에서 가장 짧은 경로를 찾는 프로그램을 제작한다.

미로 프로젝트 1주차 실습에선 완전 미로를 생성하는 프로그램을 제작한다. 완전 미로란 임의의 서로 다른 출발점과 도착점을 연결하는 경로가 오직 하나만 존재하는 미로를 의미한다. 완전미로를 생성하기 위한 알고리즘으로 recursive backtracker, Kruskal's algorithm, Prim's algorithm, Eller's algorithm 중 하나를 택할 수 있다. 1주차 숙제는 순환 경로가 존재하는 불완전한 미로를 생성하는 프로그램을 제작하는 것이다. 순환 경로가 존재하는 불완전한 미로란 폐쇄된 공간은 존재하지 않으나 두 지점을 연결하는 경로가 하나 이상 존재하는 미로를 말한다. 실습과 숙제 모두 확장자가 .maz인 파일로 미로를 출력한다.

미로 프로젝트 2주차 실습에선 1주차에서 만든 프로그램이 생성한 미로(확장자가 .maz인 파일의 내용)를 읽어 들여, MFC가 제공하는 윈도우 환경에서의 GUI로 그 미로를 그려내는 프로그램을 제작해야 한다. 이를 위해 미로를 표현하기 위한 효율적인 자료구조를 선택하여 활용한다. 여기에 3주차 프로젝트 내용을 위한 DFS 버튼과 BFS 버튼을 추가하는 것이 2주차의 숙제이다.

미로 프로젝트 3주차에선 2주차에서 만들어진 프로그램에 가장 짧은 경로를 찾는 기능을 제공하도록 하는 것이다. 경로를 찾는 방법은 DFS(3주차 실습)와 BFS(3주차 숙제)를 사용한다. 이 때, MFC의 GUI를 이용하여 탐색 과정과 결과를 사용자에게 보여주어야 한다.

2 요구사항

2.1 설계 목표 설정

구현 1주차에는 완전 미로와 불완전 미로 생성 로직을 각각 구현한다. 각각의 미로 생성에 필요한 자료구조와 알고리즘을 고안한다.

구현 2주차에는 1주차에서 생성한 미로를 읽어 화면에 그리는 GUI를 구성한다. 이를 위해 다음과 같은 항목들의 구현이 필요하다.

- 화면에 UI 요소를 그리는 로직
- 미로를 표현하기 위한 자료구조의 고안

구현 3주차에는 미로 해결 알고리즘을 구현한다. 이를 위해 다음 알고리즘의 이해와 구현이 필요하다.

- 깊이 우선 탐색DFS
- 너비 우선 탐색BFS

2.2 합성

깊이 우선 탐색depth first search 깊이 우선 탐색, 또는 DFS는 그래프 탐색 방법의 일종이다.

$N(u)$를 u와 인접한 노드들의 집합이라고 정의한다면, $N(u)$의 모든 원소 v_i에 대해 $N(v_i)$를 전부 방문한 후 v_{i+1}을 방문하는 방식으로 동작한다. 이를 위해 스택을 사용한다.

간단히 말하자면, DFS는 진행 방향을 정하고 갈 수 있는 만큼 진행한 후 더 이상 진행할 수 없으면 (막다른 길을 마주하면) 이전 노드로 되돌아와 다른 경로로 진행하는 방식으로 동작한다. 이 때 이전 노드로 되돌아오는 것을 백트래킹backtracking이라 한다. 시작점부터 종점까지의 경로를 성공적으로 찾았을 경우 DFS의 스택에는 그 경로가 저장되어 있다는 특징이 있다.

DFS의 시간 복잡도는 그래프 (V,E)에 대해 인접 행렬로 표현된 그래프의 경우 $\mathcal{O}\left(V^2\right)$, 인접 리스트로 표현된 그래프의 경우 $\mathcal{O}(V+E)$를 보인다.

너비 우선 탐색breadth first search 너비 우선 탐색, 또는 BFS는 DFS와 마찬가지로 그래프 탐색 방법의 일종이다.

BFS는 큐 Q에 첫 노드를 넣고, Q가 빌 때까지 Q의 첫 원소 q에 대해 $N(q)$ 중 아직 방문하지 않은 정점들을 전부 Q에 추가하면서 그래프를 탐색해 나간다. 다시 말하면 인접한 노드들부터 차례로 탐색해나가며, 트리의 경우 깊이가 얕은 노드부터 탐색해나간다. 이는 BFS가 시작점부터의 거리가 $n+1$인 정점을 방문했을 경우 시작점으로부터의 거리가 n인 정점은 이미 모두 방문했음을 의미하며, 간선의 가중치가 모두 같은 경우 이는 최단 경로를 찾는 데 쓰일 수 있다.

BFS의 시간 복잡도는 그래프 (V,E)에 대해 인접 행렬로 표현된 그래프의 경우 마찬가지로 $\mathcal{O}\left(V^2\right)$, 인접 리스트로 표현된 그래프의 경우 $\mathcal{O}(V+E)$를 보인다.

재귀 백트래킹recursive backtracker 재귀 백트래킹은 완전 미로를 만드는 알고리즘 중 하나이다. 완전 미로는 최소 스패닝 트리minimum spanning tree로 생각할 수 있는데, 이 점에서 착안해 미로의 모든 칸을 노드로 생각한다면 랜덤한 점에서 랜덤하게 MST를 만드는 식으로 완전 미로를 구성할 수 있다. 따라서 DFS를 응용해 MST를 구성하여 미로를 생성할 수 있다.

2.3 분석

1주차: 완전 미로를 생성하는 로직 높이 h, 너비 w의 미로에서 백트래킹은 다음과 같이 작동한다.

BACKTRACK(x, y, h, w)
1 Calculate unvisited adjacent cells to (x, y)
2 Set visited state of (x, y) to TRUE
3 **while** there exists unvisited adjacent cells to (x, y)
4 (nx, ny) = random unvisited adjacent cell to (x, y)
5 Make path from (x, y) to (nx, ny)
6 BACKTRACK(nx, ny, h, w)
7 Recalculate unvisited adjacent cells to (x, y)

랜덤하게 결정한 (x, y)에서 백트래킹을 시작하면 된다.

자료구조는 가로 길을 나타내는 $h \times (w-1)$ 크기의 불 배열과 세로 길을 나타내는 $(h-1) \times w$ 크기의 불 배열로 표현하였다. 실제로는 각각 `horizontal_route`와 `vertical_route`라는 이름으로 구현되었는데, 미로에서 (x, y)에서 $(x, y-1)$로 갈 수 있다면 `vertical_route[y - 1][x]`가 `1`이고, 아닐 경우 `0`인 자료구조이다. 이 자료구조의 공간 복잡도는 $\mathcal{O}(h \times w)$이다.

인접한 셀의 개수를 계산하는 알고리즘은 4방향만 체크하면 되므로 $\mathcal{O}(1)$으로 충분하다. 백트래킹은 각 셀마다 인접한 미방문 셀을 방문하는데, 결국 모든 셀을 방문하게 되므로 시간 복잡도도 $\mathcal{O}(h \times w)$이다.

1주차: 불완전 미로를 생성하는 로직 불완전 미로의 생성은 생성된 완전 미로에서 추가로 몇 갸의 벽을 없애 주는 것으로 가능하다. 모든 칸들을 랜덤하게 돌면서 옆 셀과 벽으로 막혀 있는 셀이 있다면 길을 만들어 주는 식으로 n번 반복한다. 자료구조는 그대로 사용하였다.

DELETE-WALLS($delete$, h, w)
1 $delete = \min\left(delete, \frac{\min(h,w)}{2}\right)$
2 **while** $delete > 0$
3 (x, y) = random cell in maze field
4 (nx, ny) = random adjacent cell to (x, y)
5 **if** there is no direct path between (x, y) and (nx, ny)
6 Make new direct path
7 $delete = delete - 1$

정점이 hw개 존재하는 그래프의 최소 신장 트리에는 노드가 $hw-1$개 존재한다. 또한 미로의 모든 벽을 삭제하면 간선은 $h \times (w-1) + (h-1) \times w$개 존재한다. 따라서 완전 미로 상태에서 셀과 셀 사이의

벽은

$$\begin{aligned}&[h\times(w-1)+(h-1)\times w]-[hw-1]\\=&hw-h-w+1\end{aligned}$$

개이다. 따라서 총 $h\times(w-1)+(h-1)\times w$개의 원래 존재하던 벽 중 $hw-h-w+1$개를 고르는 확률로 벽을 지우는 데 성공할 수 있다.

n개의 벽 중 k개의 벽이 남아 있을 때, 지우기에 성공하는 확률은 $\frac{k}{n}$이다. 따라서 이 때 처음으로 존재하는 벽을 지우기까지의 시도 횟수를 X라 하면 $X\sim\mathrm{Exp}\left(\frac{k}{n}\right)$이므로 이 때 $\mathrm{E}(X)=\frac{n}{k}$이다.

이 알고리즘에서 벽을 r개 지우도록 한다면 벽 r개를 지우는 데 필요한 시도 횟수의 기댓값은 n번째 조화수 $H_n=\sum_{k=1}^{n}\frac{1}{k}$라고 할 때

$$\begin{aligned}&\sum_{k=0}^{r-1}\frac{h\times(w-1)+(h-1)\times w}{hw-1-k}\\=&\sum_{k=0}^{r-1}\frac{2hw-h-w}{hw-1-k}\\=&(2hw-h-w)\left(H_{h\times w-1}-H_{h\times w-r-1}\right)\end{aligned}$$

이다.

한편 상수 $\gamma\approx0.577$에 대해 $\lim_{n\to\infty}(H_n-\ln n)=\gamma$임이 알려져 있으므로, $hw=A\to\infty$일 때

$$
\begin{aligned}
&\lim_{A\to\infty}\sum_{k=0}^{r-1}\frac{h\times(w-1)+(h-1)\times w}{hw-1-k}\\
&=\lim_{A\to\infty}\left[(2A-h-w)\left(H_{A-1}-H_{A-r-1}\right)\right]\\
&=\lim_{A\to\infty}\left[(2A-h-w)\left(\ln(A-1)-\ln(A-r-1)\right)\right]\\
&=\lim_{A\to\infty}\left[(2A-h-w)\ln\frac{A-1}{A-r-1}\right]\\
&=\lim_{A\to\infty}\ln\left(\frac{A-1}{A-r-1}\right)^{2A-h-w}\\
&=\lim_{A\to\infty}\ln\left(1+\frac{r}{A-r-1}\right)^{2A-h-w}\\
&=\lim_{A\to\infty}\ln\left(1+\frac{r}{A-r-1}\right)^{2r\times\frac{A-r-1}{r}+2r+1-h-w}\\
&=\lim_{A\to\infty}\ln\left[\left[\left(1+\frac{r}{A-r-1}\right)^{\frac{A-r-1}{r}}\right]^{2r}\left(1+\frac{r}{A-r-1}\right)^{2r+1-h-w}\right]\\
&=\lim_{A\to\infty}\ln\left[\left(1+\frac{r}{A-r-1}\right)^{\frac{A-r-1}{r}}\right]^{2r}+\lim_{A\to\infty}\ln\left(1+\frac{r}{A-r-1}\right)^{2r+1-h-w}\\
&=\ln\lim_{A\to\infty}\left[\left(1+\frac{r}{A-r-1}\right)^{\frac{A-r-1}{r}}\right]^{2r}+(2r+1-h-w)\ln\lim_{A\to\infty}\left(1+\frac{r}{A-r-1}\right)\\
&=\ln e^{2r}+(2r+1-h-w)\ln 1\\
&=2r
\end{aligned}
$$

이고, 따라서 hw의 미로에서 벽을 r개 지운다면 평균적으로 $2r$번의 연산이 필요하다. 결론적으로 평균적인 경우 완전 미로에서 벽들을 더 지워 불완전 미로를 만드는 데 드는 시간 복잡도는 $\mathcal{O}(r)$으로 계산 가능하다. 불완전 미로를 처음부터 새로 만드는 경우를 생각한다면 완전 미로를 먼저 만들어야 하므로 전체 시간 복잡도는 $\mathcal{O}(h\times w+r)$이다.

자료구조는 동일한 것을 사용하였으므로 공간 복잡도도 마찬가지로 $\mathcal{O}(h\times w)$이다.

2주차: 미로를 화면에 그리는 로직 미로를 화면에 그리기 위해 미로를 2차원 동적 배열에 저장한다.

이 때 각 칸은 '+', '−', '|' 또는 '.'으로 구성되는데, '.'의 경우 길이 있음을, '+', '−', '|'의 경우 벽이 있음을 나타내므로 이에 따라 적절한 크기의 사각형을 적절한 위치에 그리면 될 것이다. 따라서 $rows\times cols$의 미로의 경우 직사각형을 $rows\times cols$번 그리는데, 지금 확인하는 칸이 '+'일 경우 작은 정사각형을, '-'일 경우 좌우로 긴 직사각형을, '|'일 경우 상하로 긴 직사각형을 적절한 위치에 그린다. 크기는 `WALL_WIDTH`와 `CELL_SIZE`에 미리 정의한다.

i와 j가 홀수일 경우는 현재 확인하고 있는 칸은 원래 미로의 각 칸이 되는 경우이고 i 혹은 j가 짝수일 경우는 원래 미로의 벽 혹은 통로가 되는 경우이다. 따라서 이 점을 이용해 직사각형을 그리기 시작하는 왼쪽 위 x 좌표는 i에 대해

$$\left(CELL\_SIZE \times \left\lfloor \frac{i}{2} \right\rfloor\right) + \left(WALL\_WIDTH \times \left\lceil \frac{i}{2} \right\rceil\right)$$

이고, 이는 y 좌표에 대해서도 마찬가지이다. 직사각형을 그리는 작업이 상수 시간만큼 걸린다고 가정하면 미로를 그리는 데 소요되는 시간 복잡도는 $\mathcal{O}(rows \times cols)$ 이고 미로를 저장하는 공간 복잡도도 $\mathcal{O}(rows \times cols)$이다.

특히 '.' 칸들에만 정점이, 인접한 '.' 칸들 사이에만 간선이 있다고 생각하면 미로를 그래프처럼 생각하고 탐색할 수 있으므로 굳이 가로/세로 길을 계산해 저장할 필요가 없다.

2주차: 툴바에 버튼 추가 MFC API를 사용하여 툴바에 버튼을 추가한다. 프로젝트의 리소스는 `MFC_-Main.rc`라는 파일이 관리하는데, 프로젝트의 `IDR_MAINFRAME`에서 Event Handler Wizard를 이용해 메뉴를 추가할 수 있다.

3주차: 그래프를 탐색하는 로직 그래프를 DFS와 BFS 두 방법을 사용해 탐색한다.

배열의 좌상단 인덱스 $(0, 0)$, 크기를 $n \times m$이라고 할 때, 미로의 시작점은 $(1, 1)$이고, 종점은 $(n-2, m-2)$이고, 어떤 칸 (x, y)를 노드라고 생각할 때 그 노드에 연결된 노드들은 인접한 상하좌우 4칸, 즉 $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$로 생각할 수 있다.

각 탐색 알고리즘의 구현은 다음과 같았다.

- **DFS** – 스택 S를 정의한다. S에 첫 노드를 넣고, S의 첫 원소 s에 대해 $N(s)$ 중 아직 방문하지 않은 정점들을 전부 S에 추가하고, S의 첫 원소를 제거한다. 이를 S가 빌 때까지 반복한다.
- **BFS** – 큐 Q를 정의한다. Q에 첫 노드를 넣고, Q의 첫 원소 q에 대해 $N(q)$ 중 아직 방문하지 않은 정점들을 전부 Q에 추가하고, Q의 첫 원소를 제거한다. 이를 Q가 빌 때까지 반복한다.

세부적으로, $p = (p_x, p_y)$에 대해 $N(p) = \{(p_x+1, p_y), (p_x-1, p_y), (p_x, p_y+1), (p_x, p_y-1)\}$ 으로 정의할 수 있다.

방문 여부를 체크하기 위한 $n \times m$ 배열을 따로 만들어 관리한다. 각각의 알고리즘에서 최악의 경우 스택이나 큐에 $n \times m$개의 원소가 들어가고 나온다. 따라서 각각 알고리즘의 공간 복잡도는 $\mathcal{O}(n \times m)$ 이며, 시간 복잡도도 마찬가지다.

2.4 제작

2.3에서 설명한 이론과 실제 코드를 비교하고 분석한다.

프로젝트에서 수정하여야 하는 주요 파일들은 다음과 같다.

파일 이름	기능
generate_perfect_maze.c	완전 미로를 생성하는 C 소스
generate_imperfect_maze.c	불완전 미로를 생성하는 C 소스
MFC_Main.rc	MFC 프로젝트에서 리소스들을 정의하고 있는 파일
User Code/usercode.cpp	미로를 읽고, DFS와 BFS를 진행하고, 화면에 그리는 등 핵심적인 로직이 들어가는 부분

미로 프로젝트 1주차 1주차에 구현된 사항은 다음과 같다.

generate_perfect_maze.c: 완전 미로를 생성한다. 이론의 의사 코드를 그대로 구현했다.

generate_imperfect_maze.c: 완전 미로를 생성한다. 이론의 의사 코드를 그대로 구현했다.

미로 프로젝트 2주차 2주차에 구현된 사항은 다음과 같다.

화면에 미로를 그리는 로직: 의사 코드를 그대로 구현했으나, 실제 구현에서는 미로에 그레이디언트를 추가했다. 아래는 실제 구현된 코드이다.

```
static void drawBuffered(){
    for (int i = 0; i < rows; i++) {
        int x = CELL_SIZE * (i / 2) + WALL_WIDTH * (i - i / 2);
        for (int j = 0; j < cols; j++) {
            int y = CELL_SIZE * (j / 2) + WALL_WIDTH * (j - j / 2);

            // this adds gradients just for fun
            double blend = (double) (i + j) / (rows + cols);
            COLORREF color = RGB(
                0x00 * (1 - blend) + 0x9c * blend,
                0xbc * (1 - blend) + 0x27 * blend,
                0xd4 * (1 - blend) + 0xb0 * blend
                );

            switch (field[i][j]) {
                case '+':
                    DrawSolidBox_I(y, x,
                        y + WALL_WIDTH, x + WALL_WIDTH,
```

```
                        0, color, color);
                    break;
                case '-':
                    DrawSolidBox_I(y, x,
                        y + CELL_SIZE, x + WALL_WIDTH,
                        0, color, color);
                    break;
                case '|':
                    DrawSolidBox_I(y, x,
                        y + WALL_WIDTH, x + CELL_SIZE,
                        0, color, color);
                    break;
            }
        }
    }
}
```

툴바에 메뉴와 버튼 추가: Event Handler Wizard를 이용해 메뉴를 추가했으며, MFC_MainDoc.cpp의 코드에서는 다음과 같은 항목들이 추가되었다.

```
BEGIN_MESSAGE_MAP(CMFC_MainDoc, CDocument)
    //{{AFX_MSG_MAP(CMFC_MainDoc)
    ON_UPDATE_COMMAND_UI(ID_FILE_CLOSE, OnUpdateFileClose)
    ON_COMMAND(ID_FILE_CLOSE, OnFileClose)
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_TOOLS_DFS, &CMFC_MainDoc::OnToolsDfs)
    ON_COMMAND(ID_TOOLS_BFS, &CMFC_MainDoc::OnToolsBfs)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_DFS, &CMFC_MainDoc::OnUpdateToolsDfs)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_BFS, &CMFC_MainDoc::OnUpdateToolsBfs)
END_MESSAGE_MAP()
```

미로 프로젝트 3주차 3주차에 구현된 사항은 다음과 같다.

DFS/BFS: 의사 코드를 그대로 구현했으나, 실제 구현에서는 방문한 경로를 저장하는 방식을 구체화하였다. 다음은 BFS에서 방문하는 경로를 저장하는 로직을 추가한 코드이다.

```
    std::vector<std::vector<std::pair<int, int>>> prev_cell;
    prev_cell.resize(rows);
    for (int i = 0; i < rows; i++)
```

```
326         prev_cell[i].resize(cols);
327
328     std::queue<std::pair<int, int>> q;
329     q.emplace(std::make_pair(1, 1));
330     visited_cells[1][1] = true;
331
332     bool flag = false;
333
334     while (q.size() && !flag) {
335         int x = q.front().first;
336         int y = q.front().second;
337         checked_cells[x][y] = true;
338         q.pop();
339
340         for (int d = 0; d < 4; d++) {
341             int nx = x + dx[d];
342             int ny = y + dy[d];
343
344             if (visited_cells[nx][ny])
345                 continue;
346             if (field[nx][ny] != ' ')
347                 continue;
348
349             visited_cells[nx][ny] = true;
350             q.emplace(std::make_pair(nx, ny));
351             prev_cell[nx][ny] = std::make_pair(x, y);
352
353             if (nx == dest_x && ny == dest_y) {
354                 flag = true;
355                 break;
356             }
357         }
358     }
```

위의 코드에서 `visited_cells`, `prev_cell`을 참조한다.

또한 최단 경로를 그리기 위해 `drawBuffered` 함수에 줄 186–251을 추가하였다.

2.5 시험

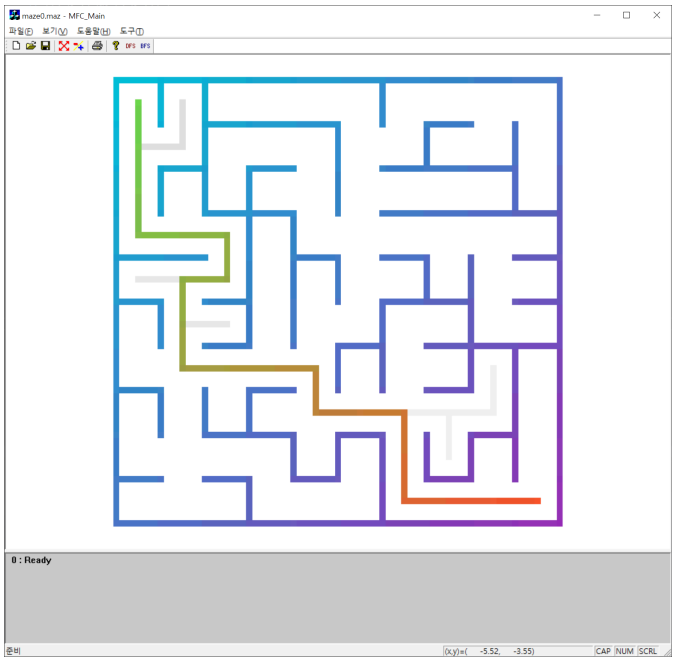

Fig. 1: DFS 수행 스크린샷

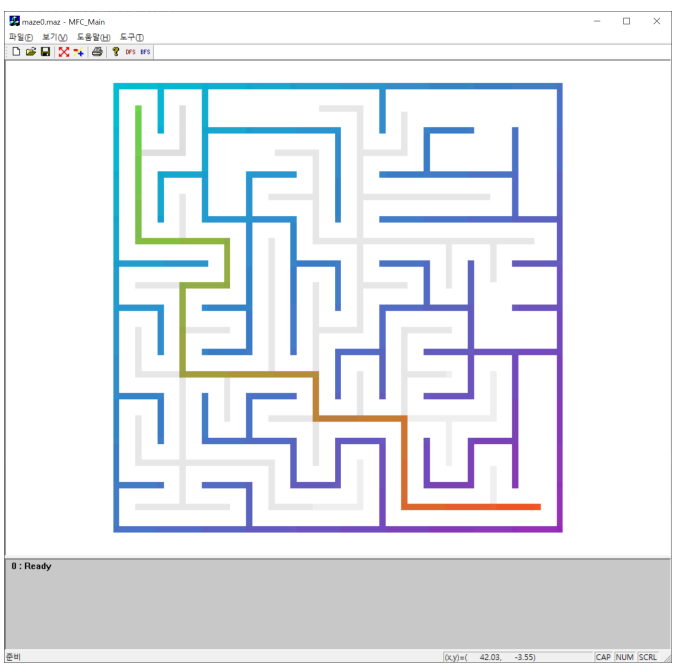

Fig. 2: BFS 수행 스크린샷

2.6 평가

Figure 1는 미로를 불러와 DFS로 미로를 해결했을 때의 스크린샷이다. 정상적으로 미로의 솔루션을 찾을 수 있었다.

Figure 2는 미로를 불러와 BFS로 미로를 해결했을 때의 스크린샷이다. 정상적으로 미로의 솔루션을 찾을 수 있었다.

BFS로 찾은 뒤 DFS로 다시 찾으라고 명령했을 경우나 반대 순서로 명령했을 경우에도 제대로 동작하였으며, 새 파일을 열어 경로를 탐색했을 경우에도 잘 동작하였다.

2.7 환경

MFC를 이용한 윈도우 프로그래밍이 요구되므로, 미로 프로젝트의 각 프로그램은 윈도우 환경에서 제작된다.

2.8 미학

프로젝트에서 정의하고 있는 미로의 모습대로 미로를 만들어야 하고, 이를 GUI 프로그램으로 그려내는 경우 또한 프로젝트에서 정의하고 있는 모습대로 그려야 한다.

2.9 보건 및 안정

입력되는 미로 파일의 포맷이 잘못되었을 경우 오동작이 발생할 수 있다. 이외의 경우 프로그램은 안정적이다.

3 기타

3.1 환경 구성

이 프로젝트를 제작하는 데 사용된 시스템의 소프트웨어/하드웨어 정보는 다음과 같다.

항목	값
운영 체제	Microsoft Windows Version 10.0.17763.557
아키텍쳐	x86-64
CPU	Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz
IDE	Microsoft Visual Studio Community 2019 (Version 16.0.3)
	Microsoft .NET Framework 4.7.03190
Platform Tools	Visual Studio 2019 (v142)

3.2 참고 사항

특별히 참고할 만한 사항은 없다.

3.3 팀 구성

서강대학교 컴퓨터공학과 박수현 (20181634) 100% 기여. 개인 프로젝트로 진행하였다.

3.4 수행기간

2019년 5월 27일부터 2019년 6월 21일까지.