컴퓨터공학 실험: 미로 프로젝트

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 완전미로를 생성하는 프로그램을 제작한다. 완전미로란 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

미로 프로젝트의 각 주차별 프로그램을 설계하는 과정에서의 목표를 설정하고, 그 과정에서 요구되는 사항들을 정리한다.

1주차

① 미로의 길이 N, 너비 M을 입력받은 후, 입력받은 N,M에 대해 동적 할당으로 2차원 배열을 만든다.

② Eller's Algorithm을 사용하여 실습에서는 완전미로를, 과제에서는 불완전 미로를 만들어 본다.

③ 만든 미로를 '+,-,|'을 사용한 텍스트 형식의 그림으로 "~~.maz"파일로 출력하고, 작성한 프로그램에 대해 메모리를 해제시킨다.

2주차
① 1주차에서 만든 미로로 출력파일을 입력으로 받는다.
② 자신이 만든 자료구조를 사용하여 MFC 상에서 미로를 그린다.
③ 작성한 소스에 대해 동적할당 한 것을 해제시킨다.

3주차
① 2주차에서 구현한 자료구조를 가지고 DFS를 구한다. 이를 가지고 방문한 모든 경로를 저장한다.
②구한 해답과 경로를 2주차에서 그린 미로 위에 그린다. 시작점과 끝 점을 다른 색으로 구별 가능ㄴ하게 그려준다.

2.2 합성

미로 프로젝트의 각 주차별 프로그램의 설계 및 구현을 위해 요구되는 이론, 자료구조, 알고리즘 등을 조사 분석하여 전체적인 설계를 수행한다.

①Eller's Algorithm(1주차)
임의의 한 방에서 시작해서 DFS를 돌리는 방법, (1,1)을 Root로 하는 Tree 생성 등이 있다. 이 중 가장 시간복잡도, 공간복잡도면에서 Eller's Algorithm이 가장 효율적이라고 생각하여, 이 알고리즘을 선택하였다. 모든 자원은 동적 할당을 이용해 메모리에 적재하고, 필요 없어지면 반드시 메모리를 해제시켜준다.
② MFC(2주차)
이번 프로젝트에서는 MFC가 지원하는 두 가지 Draw Direct, Draw Buffered 중 Buffer에 쌓아놨다가 한 번에 그려주는 후자를 택하였다. 따라서 이번 프로젝트에서 그리는 모든 작업은 drawBuffered 함수 내에서 수행하도록 한다. 또한, setWindow 함수를 호출하여 그림을 그릴 "캔버스" 역할을 할 윈도우를 생성하도록 한다.
③ DFS Algorithm
DFS를 Iterative하게 수행해야 하기 때문에 Stack을 직접 만들어 Push, Pop함수를 사용한 Linked List 자료구조를 사용하도록 한다.

DFS를 제외하고 구성한 모든 자료구조들의 공간 복잡도와 시간 복잡도는 O(NxM)이며, DFS의 공간, 시간복잡도는 b^d 라고 할 수 있다.(여기서 b는 노드에서 뻗어나갈수 있는 자식의 수, d는 트리의 깊이)

2.3 분석

먼저 1주차의 주된 목표인 Eller's Algorithm에 대해서 알아본다. 이 알고리즘은 완전 미로를 만들기 위한 알고리즘으로, 차례로 한 행씩 미로를 만들어가는 특징이 있는 알고리즘이다. 이 알고리즘은 다음과 같은 절차를 따른다.

1. 미로의 첫 번째 줄을 초기화 한다. 첫 번째 줄에 속한 N개의 방은 N개의 서로 다른 집합 속에 속하게 된다.

2. 미로의 현재 행에서 첫 번째 방부터 시작하여 마지막 방까지 서로 인접해 있는 방들이 어떤 집합에 속하는지 비교한다. 인접해 있는 두 방이 서로 다른 집합에 속해있다면 두 방 사이의 벽을 남겨둘지 제거할지 임의로 선택한다. 벽을 제거하면 인접한 방들을 연결하는 통로가 생기는데 이렇게 해서 연결된 방들은 모두 같은 집합으로 합쳐진다.

3. 현재 줄과 다음 줄 사이의 벽을 제거하고 두 방 사이의 수직 경로를 만든다. 현재 줄의 방 중 하단의 벽을 제거하고 다음 줄의 방과 연결될 방은 각 집합마다 하나 이상을 임의로 선택하여야 한다. 두 방 사이에 수직 경로가 만들어지면 이러한 두 방이 같은 집합에 속하게 되므로 첫 번째 줄의 방과 같은 집합에 속한 방들이 두 번째 줄에 나타나게 된다.

4. 3번에서 완성하지 않은 다음 줄의 나머지 방들을 구체화한다. 바로 전 줄과 수직의 경로로 연결된 방들 이외에 나머지 방들은 각각 서로 다른 집합에 속하게 한다.

5. 마지막 줄에 도달할 때까지 2번~4번까지의 과정을 반복한다.

6. 미로의 마지막 줄에서는 인접해 있으며 서로 다른 집합에 속한 방들 사이의 모든 벽을 제거한다. 따라서 이 과정을 수행한 후 크기 NxM인 미로의 모든 NxM개의 방은 같은 집합에 속하게 된다.

미로 프로젝트 2주차의 문제는 자료구조에 정보를 어떻게 효율적으로 저장하는 가이다. 자료구조만 제대로 만들어 놨다면, 그 자료구조의 정보 자체를 MFC 함수들을 이용해서 그려주면 되므로 그리는 것 자체는 문제가 되지 않는다. 그림을 그리는 데에는 오른쪽과 아래만 필요하기 때문에 구조체에 오른쪽, 아래를 포함하는 구조체를 저장한 뒤 그리면 된다.

3주차 “경로 탐색”에서는 또 다른 자료 구조인 Maze와 Visited 2차원 배열이 쓰인다. Maze배열은 실제적인 경로를 DFS로 탐색하기 위해 쓰이는 자료 구조이며, 해답 경로의 탐색 중에 방문한 적이 있는 방을 체크해두는 배열은 Visited이다. Visited는 0으로 초기화하며, 방문한 i, j 번째 방을 1로 세팅한다.(즉, Visited[i][j] = 1;) Maze 자료구조는 다음과 같은 구성을 가진다.

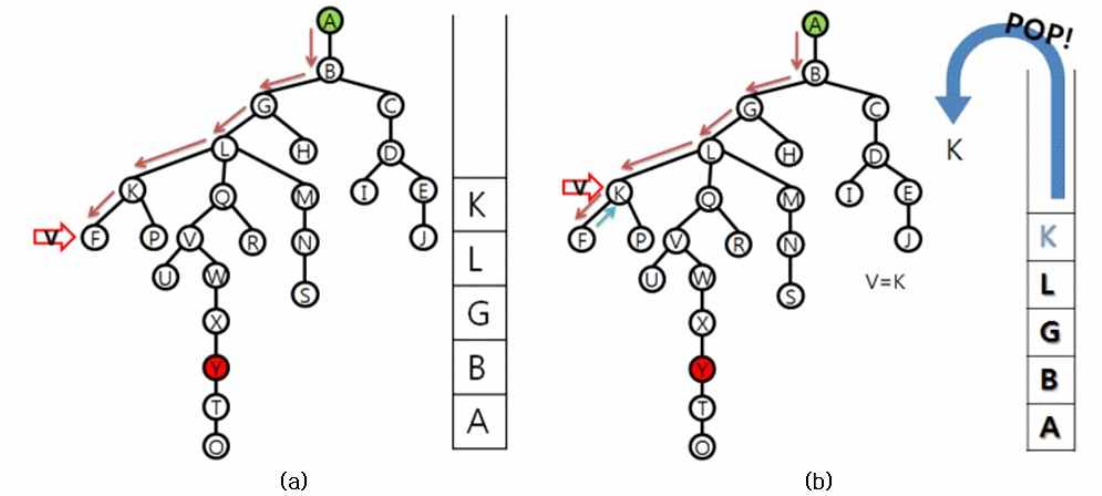
Maze[i][j]번째 방의 최하위 bit이 세팅되어 있다면 :
위쪽으로 갈 수 있는 길이 있다는 뜻이다
Maze[i][j]번째 방의 최하위에서 두 번째 bit이 세팅되어 있다면 :
아래쪽으로 갈 수 있는 길이 있다는 뜻이다
Maze[i][j]번째 방의 최하위에서 세 번째 bit이 세팅되어 있다면 :
왼쪽으로 갈 수 있는 길이 있다는 뜻이다
Maze[i][j]번째 방의 최하위에서 네 번째 bit이 세팅되어 있다면 :
오른쪽으로 갈 수 있는 길이 있다는 뜻이다
Maze[i][j]번째 방의 최상위 bit이 세팅되어 있다면 :
i, j번째 방에 실제로 방문했다는 뜻이다.

Maze[i][j]번째 방의 최상위에서 5번째 bit이 세팅되어 있다면 :
해답 경로를 표시하는 것이다.

►내가 사용한 이론

①DFS 탐색

node A부터 시작하여 그래프로 변환된 미로를 DFS 탐색방법으로 계속 탐색하면서 출구인 node Y를 찾을 때, 아래그림 (a)의 F처럼 방문한 적이 없었던 자식 node가 더 이상 존재하지 않는 vertex에 도달하게 되면 (b)처럼 stack에 저장해둔 node를 pop시켜서 이전 node로 되돌아 갈 수 있을 것이다. 자신이 구현한 미로의 자료구조를 어떻게 stack에 저장할 것인지 생각해보자.



②Iterative와 Recursive

2.4 제작

1주차 목표인 미로 만들기는 실제 Eller's Algorithm을 사용하면 어렵지 않게 구현이 가능하다. 즉, N행 M열에 대해 이중 for문을 돌리서 만약 mMiro[i][j] 와 mMiro[i][j+1]과 같다면 뚫고 그렇지 않으면 rand()%2를 통해 이 값이 0이면 오른쪽으로 그룹을 합치는, 즉 오른쪽으로 벽을 뚫는 수행을 한다. 이 때, 수행중인 행 안에서 만약 합쳐지는 그룹의 번호를 가지는 방이 존재하면, 그 그룹을 모두 바꿔주는 수행을 한다. 그리고는 아래로 뚫는 수행을 하는데, 이 수행 또한 rand()%2를 통해 이 값이 0이면 아래쪽으로 그룹을 확장한다. 즉 아래쪽으로 벽을 뚫는다. 이것을 할 때, 각 집합 당 적어도 하나의 길을 아래도 뚫어야 하므로 이를 카운터 해주며 나머지 줄은 따로 처리를 해준다.

위 알고리즘을 정상적으로 수행하였다면 미로가 출력될 것이다. 가장 윗 행을 우선적으로 출력한다. 즉, 열의 개수 만큼의 "+-"를 출력한다. 그 후 하나의 "+"를 출력한 후 줄을 내린다. 2중 for문을 돌면서 만약 홀수 번째 행에서 수행을 하고 있고, 해당 방이 오른쪽으로 뚫려있다면 " "(띄어쓰기 두 번)을, 아래로 뚫려있다면 " |"(띄어쓰기 한

번, | 한 번)을 출력한다. 만약 짝수 번째 행에서 수행을 하고 있고, 해당 방이 아래로 뚫려있다면 " +"(띄어쓰기 한 번, + 한 번)을 출력하고, 그 이외에는 "-+"를 출력한다. 이로써 미로를 만들어 실제로 텍스트 그림으로 된 미로를 생성할 수 있다.

이 완전 미로에서 벽을 아무거나 제거하면 불완전 미로가 된다.

```
while(1)
{
    if(curx == n-1&&cury == m-1)
    {
        break;
    }

    check = 0;

    if(curx+1 !=n && mMiro[cury][curx].right && mMiro[cury][curx+1].visit) // 오른
    {
        curx++;
        push(curx,cury);
        mMiro[cury][curx].visit = false;
        check =1;
    }
    else if(cury+1 != m && mMiro[cury][curx].bot && mMiro[cury+1][curx].visit)//아래
    {

        cury++;
        push(curx,cury);
        mMiro[cury][curx].visit = false;
        check =1;
    }
    else if(cury !=0 && mMiro[cury-1][curx].bot && mMiro[cury-1][curx].visit)//위
    {

        cury--;
        push(curx,cury);
        mMiro[cury][curx].visit = false;
        check =1;
    }
    else if(curx !=0 && mMiro[cury][curx-1].right && mMiro[cury][curx-1].visit)//왼
    {
        curx--;
        push(curx,cury);
        mMiro[cury][curx].visit = false;
        check =1;

    }
```

2주차 목표인 미로 그리기는 MFC 프로젝트를 생성하여, User Code의 수정만을 하여 작성한다. SetWindow를 할 때 (0,0) ~ (M*20, N*20) 으로 충분히 잡고, 윈도우 좌표계로 세팅을 하면 배열과 대응되는 위치가 같기 때문에 혼동이 적다. 미로를 그릴 때는 입력 받은 텍스트 그림 그대로 수행하는데, 이중 for문을 돌면서 만약 '+' 면 무시를 하고, '-' 나 '|' 면 i, j에 대하여 window에 대응될 수 있게 수식을 만들었다. (위쪽 사진)

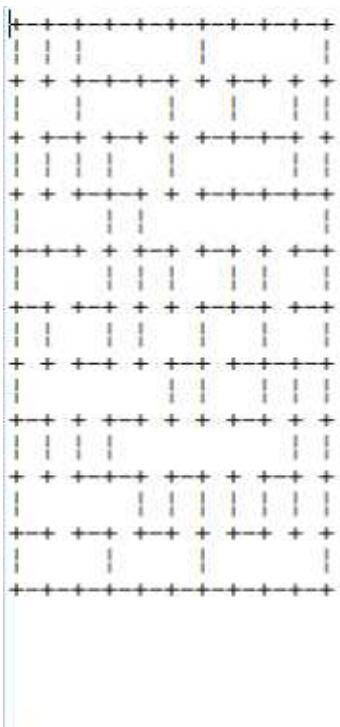
미로 프로젝트 3주차에는 자료구조 시간에 배웠던 그래프에 대해서 공부하게 되었다. 2주차에 제작했던 자료구조를 이용하여 왼쪽 벽과 위쪽 벽의 정보를 저장했다. DFS를 iterative 방식으로 구현하기 위해 stack을 template class 로 구현해서, 구조체 자체를 스택에 push 할 수 있도록 만들어 보았다. 사방으로 갈 수 있는지 확인하고 실제로 이동하게 하는 if문을 짜고, 만약 갈 수 없는 방에 도달하게 되면 pop()을 이용해서 이전 방으로 강제 이동하게 했다. 사방으로 갈 수 있는지 확인하는 if 문은 최단거리를 구할 수 있도록 오른쪽과 아래쪽 방향을 우선순위로 두고 체크 했고, 갈 수 없다면 왼쪽과 위쪽으로 이동할 수 있게 처리 해 주었다.

```
for(int i=0;i<m;i++)
{
    for(int j=0;j<n;j++)
    {
        if( !mMiro[i][j].visit )
        {
            if( j != n-1 && !mMiro[i][j+1].visit && mMiro[i][j].right ) // 오른
            {
            DrawLine_I( j+0.5, i+0.5, j+1.5, i+0.5, 3, RGB(100, 100, 100));
            }

            if( i != m-1 &&!mMiro[i+1][j].visit && mMiro[i][j].bot ) // 아래
            {
            DrawLine_I( j+0.5, i+0.5, j+0.5, i+1.5, 3, RGB(100, 100, 100));
            }

            if( j != 0 && !mMiro[i][j-1].visit && mMiro[i][j-1].right ) // 왼
            {
            DrawLine_I( j+0.5, i+0.5, j-0.5, i+0.5, 3, RGB(100, 100, 100));
            }

            if( i != 0 && !mMiro[i-1][j].visit && mMiro[i-1][j].bot ) // 위
            {
            DrawLine_I( j+0.5, i+0.5, j+0.5, i-0.5, 3, RGB(100, 100, 100));
            }
        }
    }
}

for(i=0;i<top;i++)
{
    DrawLine_I(s[i].x+0.5,s[i].y+0.5,s[i+1].x+0.5,s[i+1].y+0.5,2,RGB(200,0,0));
}
```

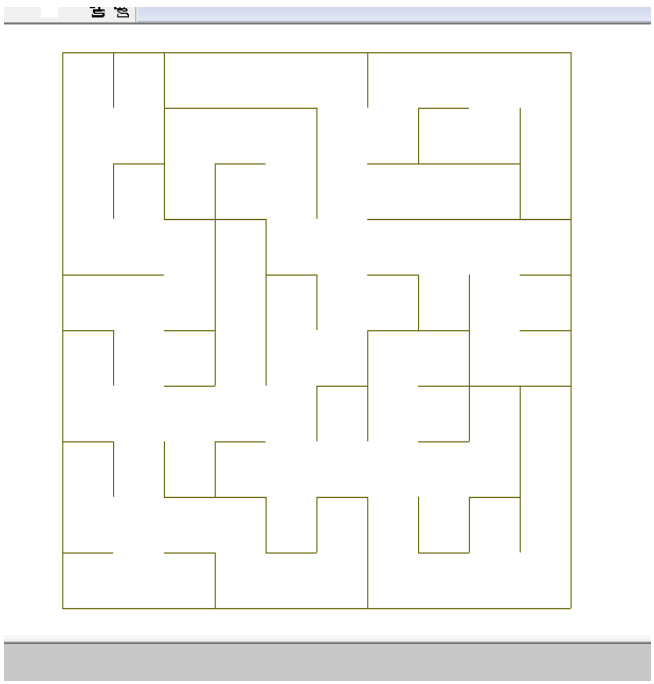
오른,왼,아래,위 쪽 어느 방향으로 갈 것인지를 짜는 코드

2.5 시험

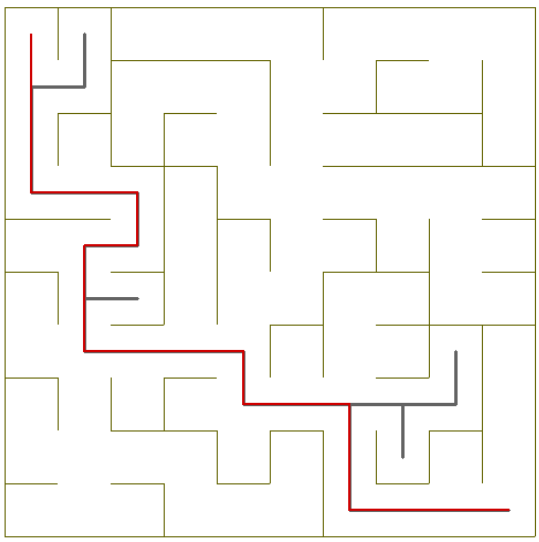
① 미로 생성화면



②미로 그리기 화면



③DFS 가실행 된 후



④가장 큰 미로 탐색

2.6 평가

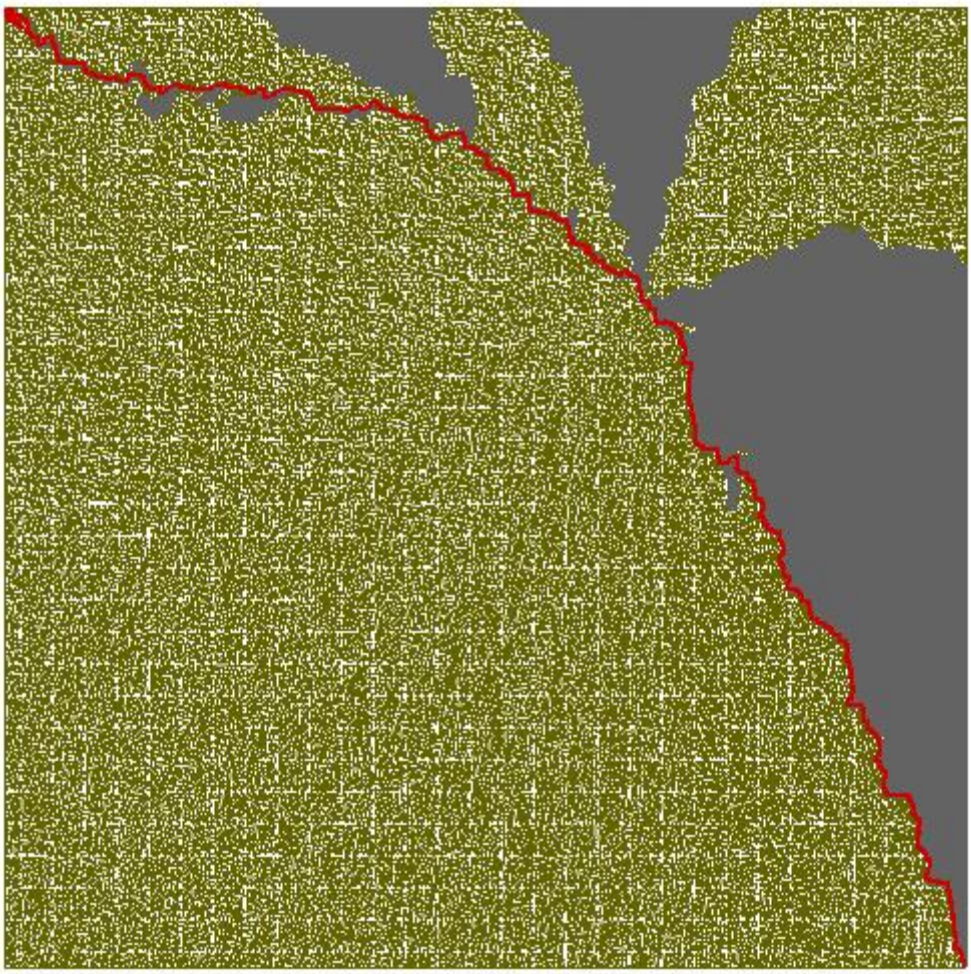
프로그램을 짜면서 미로를 그리는 경로가 실제로 생각한 것과 요구하는 것이 달라서 많이 힘들었다. 경우의 수가 많아서 최소경로를 찾는 것 뿐만 아니라 틀을 만드는 것부터 힘들었다. 하지만 다른 알고리즘이나 자료구조를 생각하면서 완벽한 미로를 짤 수 있었고 특히 DFS를 이용하여 경로를 찾는 것이 많이 힘들었다. 하지만 자료구조를 잘 이해하고 코딩을 하여 문제를 잘 해결했다. 현재 프로그램은 오작동 없이 잘 작동한다.

2.7 환경

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

2.8 미학

2주차부터 MFC Toolkit 기반 GUI를 사용하여 프로젝트를 진행해왔다. 프로젝트의 요구사항에서 제시한 미로의 모양과 완벽히 일치하며, 색깔만 내가 원하는 색으로 설정하였다.



2.9 보건 및 안정

프로그램은 5개의 예제를 시행하기 때문에 메모리 초과 현상이 일어날 수 있다. 하지만 freeMemory 함수를 사용하여서 DFS 수행을 여러 번 반복하여도 에러가 발생하지 않는 것을 확인했다. 더불어 동적 할당을 사용하였기 때문에 아무리 값을 입력하여도 동작한다. 또, 입력한 값에 불완전 미로나 막힌 미로 등을 넣어도 이에 대한 처리를 수행하였기 때문에 에러가 발생하지 않는다.

3. 기 타

3.1 환경 구성

실제 프로젝트를 수행 환경에 대해 구체적으로 기술한다. 프로젝트 수행에 사용된 하드웨어 및 소프트웨어의 상세 정보를 정리한다.

본 프로그램은 Visual Studio 2010에서 구현하였으며, 초기에 주어진 MFC Toolkit을 사용하여 프로젝트를 진행하였다. 주어진 프로그램은 Usercode부분만 수정하였으며 버튼은 MFC가 제공하는 버튼 추가 기능을 사용하였다.

3.2 참고 사항

참고사항 없음

3.3 팀 구성

김지선 기여도 100%

3.4 수행기간

~6월 19일