**CSE3013 컴퓨터공학 실험: 미로 프로젝트**

**담당교수: 서강대학교 컴퓨터공학과 장형수**

학과 : 컴퓨터공학과 분반 : 5반 학번 : 20171666 이름 : 이예은

**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 주차 별 프로그램을 설계하는 과정에서의 목표를 설정하고, 그 과정에서 요구되는 사항들을 정리한다.

|  |  |
| --- | --- |
| 1주차 | 완전미로를 이해하기 위해 완전미로를 구현한다. 이를 위해 eller’s 알고리즘을 이용한다. Eller’s 알고리즘을 사용하기 위해 방들의 집합요소 and 벽들의 정보를 가진 구조체들의 배열을 만들어야 하고 이 정보들을 이용해 출력할 수 있도록 한다. |
| 2주차 | 제공된 미로 maze\*.maz를 mfc graphic toolkit을 통해 출력하는 프로그램을 구현한다. 이를 위해 fscanf로 미로의 정보를 어떻게 받을지, 이를 구조체의 배열에 어떻게 넣을지 등을 생각해봐야 한다. 또한 1주차에서처럼 출력하되 mfc를 사용해야 하므로 이를 위한 그리기 함수를 조사한다. |
| 3주차 | 2주차에서 구현한 미로의 구조체 배열을 사용해 각각 dfs와 bfs로 경로를 찾는 프로그램을 구현한다. 이를 위해 dfs와 bfs의 기본 개념을 정리하고, 1,2 주차에 썼던 자료구조를 이용하여 어떻게 구현할지 생각해본다. |

**2.2 합성**

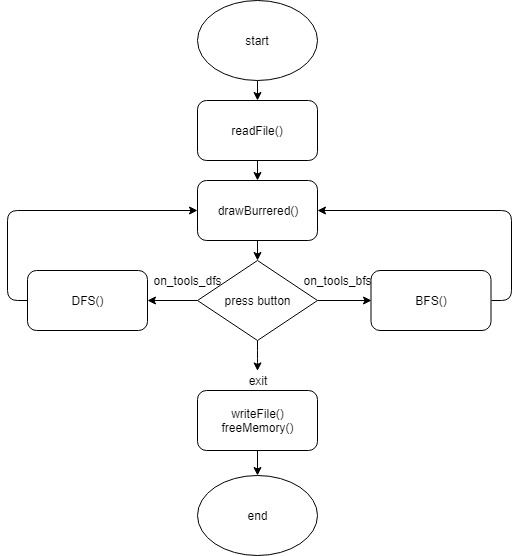
미로 프로젝트의 각 주차 별 프로그램의 설계 및 구현을 위해 요구되는 이론, 자료구조, 알고리즘 등을 조사 분석하여 전체적인 설계를 수행한다.

|  |  |
| --- | --- |
| 1주차 | 자료구조는 구조체를 이용한다. 이 때 구조체 안에는 집합의 정보와, 오른쪽 벽, 아래 벽의 정보를 저장한다. 이를 이용해 받은 수 n\*n 의 배열을 할당해준다. 그 후 eller’s 알고리즘을 이 배열을 이용해 구현한다. 이 때 0은 벽이 있다는 뜻이고, 1은 벽이 없다는 뜻으로 한다. 프린트를 할 때에는 벽이 있을경우 -나 |, 모서리의 경우 +로 출력해준다. |
| 2주차 | readFile()에서 .maz의 파일을 불러와서 이를 fscanf로 스캔받는다. 2주차부터는 구조체에 집합의 정보를 빼고, 오른쪽 벽과 아래쪽 벽의 정보만 저장해준다. 이를 모두 저장했다면 drawbuffered()함수를 불러 출력해준다. 출력을 해줄 때에는 그리기 도구를 사용해야 한다. 그리기 도구는 drawsolidbox\_I를 이용했는데 이는 drawsolidbox\_I (x1,y1,x2,y2,width,rgb(), rgb())로 불러 사용할 수 있다. 2주차에는 모서리와 벽의 구분 없이 모두 파란색 박스로 그려주고, 방의 경우는 하얀색으로 그려준다. |
| 3주차 | Dfs와 bfs를 구현해 버튼에 따라 작동하도록 구현한다. Dfs는 깊이 우선 탐색으로, 사방이 막힐 때 까지 계속 이동하는 방식이고, bfs는 너비 우선 탐색으로, 갈 수 있는 모든곳을 보면서 이동하는 방식이다. 이를 구현하기 위해 먼저 2주차에 썼던 구조체에 mark라는 변수를 하나 추가해 0으로 초기화해준다. 그리고 push 와 pop을 편리하게 하기 위해 각각 stack\_push, stack\_pop, que\_push, que\_pop함수를 만들어준다. 탐색을 하기 위해 미로를 그래프로 구현하고, 이를 이용해 dfs()와 bfs()를 하고 경로를 출력해준다. 이 때 방문을 했다면 이는 회색으로 그려준다. 그래프는 인접리스트로 구현하는 방식으로 구현한다. Bfs()의 경우, que로 구현하기 때문에 경로를 출력하기 힘들어, 갈 곳을 탐색할 때 그 이전 장소를 넣어주는 배열 path를 하나 선언해준다. 그 후 이 path 배열을 이용해 경로를 거슬러 올라가며 출력한다. |

**2.3 분석**

미로 프로젝트의 각 주차별 문제에 대해 기술하고, 상기 단계에서 설계한 알고리즘에 기반을 두어 목표로 하는 프로그램을 효과적으로 개발하는데 필요한 프로그램 기법들과 자료구조에 대하여 조사 분석하여 그것을 바탕으로 전체 프로그램 순서도를 작성한다. 또한 이러한 프로그램을 개발하는데 있어 고려해야할 모든 요소, 예를 들어, 입출력 양식, 관련 자료구조와 이론, 사용할 C언어 함수의 사용법 등 모든 가능한 고려 사항을 정리한다.

* Dfs : 깊이우선 탐색으로 stack 을 사용한다. 예를 들어 트리로 구현할 경우 가장 낮은 레벨까지 탐색하고 그 다음 자식으로 넘어가는 등의 방식이다.
* Bfs : 너비 우선 탐색으로 que 를 사용한다. 트리로 구현하는 경우, 모든 자식 노드들을 탐색 한 후 그 다음 레벨의 자식 노드를 탐색하는 방식이다.
* 입출력 양식 : 입력은 maz 파일을 파일 입출력을 통해 받는다. 다만 출력은 mfc의 그리기 도구를 사용하여 그림으로 그려준다.
* Stack : LIFO방식으로, 가장 나중에 들어온 것이 위에 쌓이고, 위부터 나간다. 맨 위를 top이라 하고, 스택에 넣는 것을 push, 빼는 것을 pop이라 한다.
* Que : FIFO방식으로, 큐의 마지막을 end, 처음을 start로 했을 때, end로 들어오고 start에서 나간다.
* 프로그램 구현도



**2.4 제작**

위에서 설계한 내용을 C언어를 사용해 구현한다. 구현 후 프로그램의 각 구성요소를 상세히 분석하여 구현방법을 프로그램 기법과 정리한 이론 등과 연관 지어 정리한다.

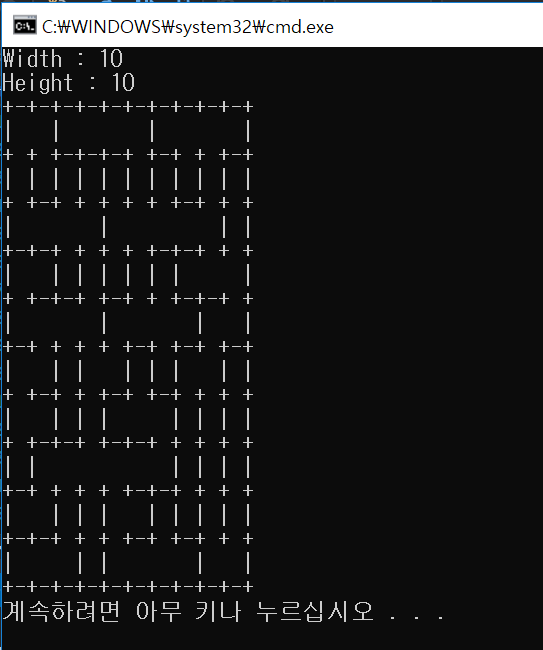
* 그래프로 구현하려 하였지만, 미로의 크기가 커질 경우 터질 위험이 매우 커 그저 갈 수 있는 곳을 push, pop 하는 방식으로 구현하였다.

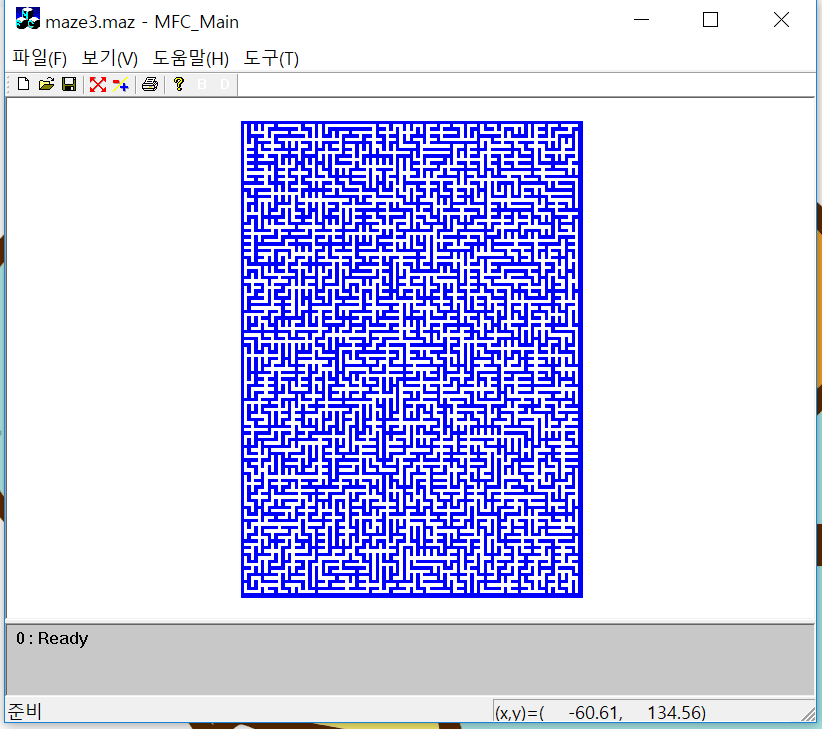
Dfs의 경우 우선 mark를 초기화해주고 시작한다. Stack에 먼저 시작점인 0,0을 넣은 후 0,0에서 갈 수 있는 곳 하나를 stack\_push 해주고, pop해준다. 이제 top에 있는 것을 이용해 계속해서 갈 수 있는 곳 중 하나를 push하고, 더 이상 갈 곳이 없으면, pop을 해준다. 이 때 갈 곳은 벽이 뚫려있고, mark가 0인 부분으로 하며, 방문하였다면 mark를 1로 설정해준다. 또한 갈 곳을 탐색할 때 if-elseif문을 사용하여, 하나라도 갈 곳을 발견하면 더 이상 탐색하지 않고 바로 다음 top 으로 넘어가도록 한다. Top이 null이 되면 길이 없는 것이므로 에러메세지를 출력하고, 도착점에 도달했다면, stack에 있는 것을 하나씩 pop하며 경로를 그려준다.

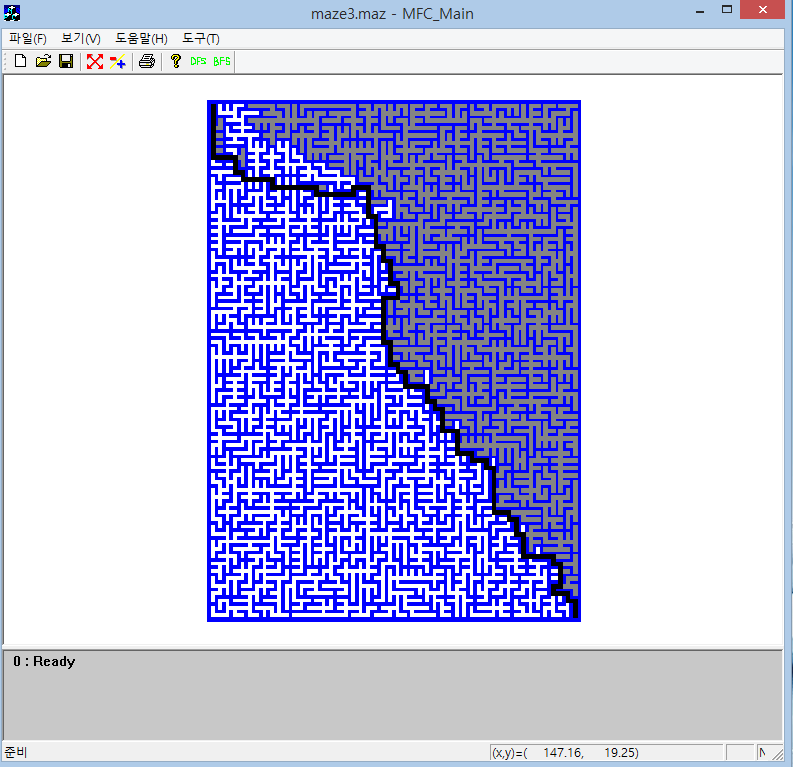
Bfs 또한 먼저 mark를 0으로 초기화 하고, que의 처음에 0,0을 넣고 시작한다. Dfs 와는 달리 if 문을 여러 개 돌려 갈 수 있는 모든 경로를 push 해주고 mark를 갱신해준다. 갈 수 있는 모든 곳을 탐색 했다면, pop 해주고 다음 start로 넘어간다. 또한 path 안에 이전 방의 정보를 넣어준다. Start가 null이 되면 길이 없는 것이므로 에러메세지를 출력하고, 도착점에 도달했다면 도착점에서부터 이 path 배열을 사용하여 거슬러 올라가며 출력해준다.

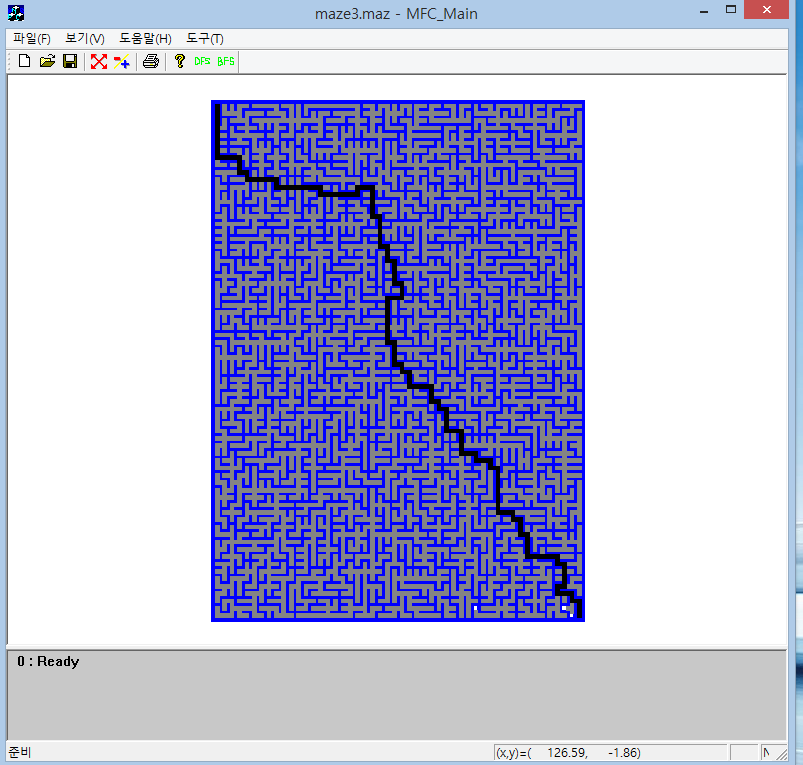
**2.5 시험**

위의 과정에서 수행한 문제 정의, 프로그램 순서도와 순서도 상의 각 부분 역할 및 구현 방법, 프로그램의 구현 방법, 구현 방법의 이론과의 연관성, 구현한 프로그램의 내용, 수행화면 등을 정리한다.

1주차 완전미로 : 

Mfc 미로 출력시 : 

Dfs : 

Bfs : 

**2.6 평가**

문제와 이론의 연관성이 적절한지, 순서도와 실제 구현 사이에 차이는 없는지 프로그램은 잘 동작하는지 등을 평가한다.

프로그램을 작성 하기 전, dfs와 bfs를 미리 조사하여 구현하려 노력하였고, 3주차에 쓸 dfs와 bfs를 위해 2주차에 미리 미로를 구조체의 배열로 구현하였다. 프로그램은 잘 동작하지만, 미로의 크기가 커질수록 시간이 오래 걸리고 메모리를 매우 많이 사용한다. 때문에 더 효율적인 방식이 있다면 그것으로 구현하는 것이 더 좋았을 것 같다.

**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 그려내야 한다.

1주차의 경우에는 완전미로를 만들기만 하는 실험이라 cmd창에 그대로 구현하였지만, 2주차부터는 mfc의 gui 프로그램 drawsolidbox\_I를 이용해 프로젝트에서 정의하고 있는 모습대로 그렸다.

**2.9 보건 및 안정**

프로젝트에서 정의하고 있지 않은 입력으로 인한 프로그램의 오동작이 없어야 한다. 또한 프로그램 동작 중에 일어날 수 있는 오류에 대처할 수 있어야 한다.

Dfs, bfs를 부를 때마다 drawbuffered()를 불러주어 dfs와 bfs 가 겹치지 않도록 하였고, mark를 계속 초기화해주어 이 또한 겹치지 않도록 구현하였다.

**3. 기 타**

**3.1 환경 구성**

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

* Mfc : [Microsoft Windows](https://namu.wiki/w/Microsoft%20Windows) 운영체제 환경에서 작동하는 [GUI](https://namu.wiki/w/GUI) 프로그램을 [C++](https://namu.wiki/w/C%2B%2B)를 사용하여 개발할 수 있도록 [Win32 API](https://namu.wiki/w/Win32%20API)의 핸들과 [C언어](https://namu.wiki/w/C(%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%20%EC%96%B8%EC%96%B4)) 함수들을 C++ 언어의 클래스화 한 [라이브러리](https://namu.wiki/w/%EB%9D%BC%EC%9D%B4%EB%B8%8C%EB%9F%AC%EB%A6%AC)이다. Microsoft [Visual Studio](https://namu.wiki/w/Visual%20Studio)의 유료 버전과 Community에는 기본으로 포함되어 있다.

**3.2 참고 사항**

프로젝트의 수행 시 참고 사항에 대해서 기술한다.

참고사항 없음

**3.3 팀 구성**

이예은 – 100%

**3.4 수행기간**

2018 – 06 - 19