**컴퓨터공학실험1 기말 프로젝트 보고서**

**20150555 남민혁**

**1. 코드 설명**

**Diagram

Description automatically generated**

**2. 프로젝트 목표**

본 프로젝트는 마지막 프로젝트였던 미로 프로젝트를 기반으로 크게 다음 두가지 방향으로 확장하였다.

* 사용자가 사전에 미로 파일을 생성하지 않더라도, 계속해서 새로운 미로를 그릴 수 있다.
* 게임의 형식으로 사용자가 직접 미로의 경로를 찾아볼 수 있다.

**3. 각 변수에 대한 설명**

**3.1. 기본 변수**

float windowWidth, windowHeight;

윈도우 화면의 너비와 높이 정보를 가지고 있다.

int WIDTH, HEIGHT;

미로의 너비와 높이 정보를 가지고 있다.

char\*\* input;

사용자로 부터 입력받은 .maz 파일이나, 프로그램 내부에서 생성한 미로를 담는 이차원 배열이다.

**3.2. 프로그램 조작**

ofxWinMenu \* menu;

사용자가 보는 화면의 상단에 있는 메뉴를 담는 변수이다.

int isOpen;

파일이 열렸는지를 판단하는 변수. 0이면 열리지 않았고, 1이면 열렸다.

int isDFS, isBFS;

각각 DFS, BFS의 방식으로 미로를 찾는 기능을 실행시켰는지 여부를 표시함. 0이면 실행하지 않았고, 1이면 실행했다.

int isPlay;

사용자에 의하여 미로를 찾는 기능이 실행되었는지 여부를 표시함. 0이면 실행하지 않았고, 1이면 실행했다.

**3.3. 미로 찾기 게임**

point player = { 1, 1 };

사용자가 움직이는 캐릭터의 위치를 저장한다. 미로의 지도가 담긴 input array의 인덱스 값이다.

**3.4. 미로 경로 찾기**

int\*\* visited;

미로 경로를 찾기 위해 방문 여부와 그 결과를 저장하는 2차원 배열이다.

point\*\* backMaze;

BFS 방식으로 미로 경로를 찾는 과정에서 현재 노드를 방문하기 이전에 방문했던 노드 (부모 노드)를 저장하는 2차원 array이다.

stack<point> stck; queue<point> qu;

미로를 찾기 위해 사용하는 스택과 큐이다. C++에서 기본 제공하는 STL을 활용하였다.

int goalX; int goalY;

미로의 경로를 찾는 과정에서 저장된 목적지의 위치이다.

point dir[4];

미로의 경로를 찾는 과정에서 탐색 방향을 나타내는 배열이다. 상하좌우로 한 칸씩 이동하도록 인덱스를 저장해 두었다.

**4. 자료구조 및 알고리즘**

**4.1. 자료구조**

**- 미로를 풀기 위한 동적 할당된 배열**

mxn 크기의 미로를 만들기 위해서 각 칸의 소속을 나타내는 정수를 저장하고, 칸 사이의 벽의 유무를 저장하는 HEIGHT x WIDTH 크기의 배열을 동적으로 선언하였다. 미로를 풀거나, 미로를 생성하기 위해 필요한 다른 배열도 아래와 같이 비슷한 방식으로 생성되었다.

|  |
| --- |
| int\*\* input = (int\*\*)malloc(sizeof(int\*) \* HEIGHT);  input[0] = (int\*)malloc(sizeof(int) \* HEIGHT \* WIDTH);  for(i = 1; i < HEIGHT; i++)  input[i] = input[i-1] + WIDTH; |

**- 미로 경로를 찾기 위한 stack과 queue**

미로 경로를 찾기 위한 DFS, BFS 알고리즘 사용에 있어 각각 필요한 stack과 queue는 c++에서 기본적으로 제공하는 자료구조를 활용하였다.

|  |
| --- |
| stack<point> stck; queue<point> qu; |

**- 좌표 저장을 위한 point**

BFS 알고리즘 구현을 위하여 부모 노드의 위치를 Array에 효과적으로 저장하고, 사용자가 움직이는 미로 내 플레이어의 위치를 효과적으로 저장하기 위하여 x, y 성분을 동시에 저장하는 point라는 구조체를 선언하였다.

|  |
| --- |
| typedef struct point {  int y;  int x;  }point; |

**4.2. 알고리즘**

**4.2.1. 미로 생성 알고리즘: Eller’s Algorithm**

다른 미로 생성 알고리즘보다 가장 메모리 공간 사용이 적고, 속도가 빠른 Eller’s Algorithm을 선택하였다. 특히 Recursive Backtracking 방식이 상당히 직관적이나, 메모리 공간을 많이 사용하며, 재귀적으로 함수를 사용함에 따라 미로의 크기가 커질 경우 상당히 큰 스택이 필요하다.

Eller’s Algorithm은 미로 생성알고리즘 중에서 가장 간단한 시간 복잡도를 가진 알고리즘으로 알려져있다. 앞서 살펴본 Recursive Backtracking과 달리 첫행부터 순차적으로 스캔하는 과정을 통해서 미로를 완성할 수 있어, 시간 복잡도는 O(n)이나, 앞서 설명한 기호를 사용하려면 nxm의 연산이 필요하다. 또한 공간복잡도 역시 별도의 스택 공간이 필요하지 않고, 미로를 저장할 메모리 공간이면 충분하며, 앞서 (2m-1)x(2n-1) 크기의 행렬로서 자료구조를 만들었으므로, (2m-1)x(2n-1) \* sizeof(int) 만큼의 메모리가 소요된다.

**4.2.2. 미로 경로 찾기 알고리즘: DFS**

DFS 알고리즘은 각각의 node에서 가장 깊이 이동할 수 있는 만큼 이동한 후 해당 지역이 미로의 끝이 아닐 경우 이전 정점으로 이동하는 방식으로 탐색한다. 개괄적으로 알고리즘은 다음과 같은 과정으로 진행된다. (1) 미로의 시작점으로 초기화한 후, 스택에 푸시한다. (2) 스택이 비어있지 않는한 현재 노드가 마지막이 아니고, 방문한적이 없는 주변 노드가 있으면, 스택에 푸시하면서 계속 깊이 들어간다.(3) 만약 더 이상 방문할 노드가 없다면 현재 스택 성분을 팝하여 다음 성분으로 넘어간다.

DFS 알고리즘은 각 node마다 최대 4번의 탐색이 일어나나, 매 탐색마다 갈 수 있는 가장 깊은 node까지 탐색하므로, 그 정도가 불규칙하다. 하지만 확실한 것은 한번 방문한 node는 다시 방문하지 않도록 하였기 때문에, 최대 미로의 방 개수 이상의 연산을 시행하지 않는다. 따라서, mxn 크기의 미로일 때, O(mxn)이다.

**4.2.3. 미로 경로 찾기 알고리즘: BFS**

BFS 알고리즘의 경우에는 현재 위치에서 갈 수 있는 모든 위치로 방문하며, 미로의 끝이 나타나는지를 확인하는 방식의 알고리즘이다. 이에 따라 현재 노드에서 갈 수 있는 모든 노드를 모두 큐에 넣은 후, 더 이상 현재 노드에서 방문할 수 있는 노드가 없다면, 큐에 들어있는 순서대로 다시 차례로 방문이 가능한 노드들을 살피는 방식으로 그래프를 탐색한다. 하지만 큐에 경로를 담지 못하고, 순차적이지 않게 탐색을 하므로, 별도로 순서를 저장하는 공간이 필요하다. 이에 따라 본 실습에서는 미로를 저장하는 input 과 비슷한 모양을 갖되, 자신을 부른 부모 노드를 저장할 수 있도록 앞서 설명하였던 새롭게 정의한 (x,y) 좌표를 저장하는 클래스로 구성된 행렬을 이용하였다.

BFS 알고리즘은 순차적으로 동일한 깊이를 유지하며, 탐색을 하므로, 각 노드 별로 최대 3번의 탐색이 일어날 수 있으나, 역시 탐색 도중 길이 없는 경우에는 한쪽 방향의 탐색이 중지될 수 있으므로 불규칙하다. 하지만 역시 한번 방문한 node는 다시 방문하지 않으므로, 같은 논리로 mxn 크기의 미로에 대해서 시간복잡도는 O(mxn)이다.

**5. 함수**

**5.1. 기본 함수**

void setup();

미로 프로그램의 초기 설정해준다.

void draw();

미로 프로그램의 기능에 따라 화면을 주기적으로 업데이트 해준다.

void doFullScreen(bool bFull);

full screen 기능 적용여부를 표시하는 bFull변수에 따라서, 화면의 배치를 수정하는 함수이다.

void doTopmost(bool bTop);

화면이 가장 앞에 위치하는지 여부를 표시하는 bTop 변수에 따라서, 화면이 어떻게 보일지를 수정하는 함수이다.

**5.2. 프로그램 조작**

bool readFile();

사용자가 사전에 생성한 .maz 확장자의 미로를 읽고, 프로그램 내부에 미로 형식을 생성한다. 미로가 성공적으로 생성되면 True, 아니면 False를 반환한다.

bool newMaze(int width, int height);

프로그램 내부의 알고리즘에 의하여 새로운 미로를 생성한다. 현재 프로그램 내에서는 3가지 단계에 따라서 10x10, 15x15, 20x20 크기의 미로를 생성할 수 있다. 미로가 성공적으로 생성되면 True, 아니면 False를 반환한다.

void appMenuFunction(string title, bool bChecked);

OpenFramework에 의해 생성되는 화면 상단에 사용자가 편리하게 프로그램의 기능을 사용할 수 있도록 상단 메뉴를 구현하였다. 즉 사용자가 상단 메뉴에서 특정 값 (title)을 누를 경우, 이 때 메뉴의 체크 여부 (bCkecked) 변수를 함께 활용하여 실제 기능이 불리도록 함수를 호출해주는 함수이다.

void keyPressed(int key);

키보드로 부터 입력을 받아 각 기능에 적합한 입력을 처리한다. 사용자로부터 특정 키보드 키(key)가 눌릴 경우, 어떤 기능이 실행되기 위해 함수를 호출해주거나, 변수의 값을 변경해준다.

**5.3. 미로 찾기 게임**

void play();

사용자가 직접 미로를 해결할 수 있도록 미로의 시작점인 좌측 상단에 스마일 캐릭터가 생긴다. 이후 좌우상하 방향키를 이용하여 캐릭터를 이동하여 미로 위에 그리고, 목적지인 우측 하단에 도달할 경우에 스마일 캐릭터의 색이 변하고, 크기가 커진다.

bool checkmove(int pointx, int pointy);

현재 위치에서 상하좌우 방향키를 눌러 움직이는 새로운 위치(pointx, pointy)를 입력받아 이동 가능 한지 여부 (벽인지 확인)를 판단한다. 이 때 이동가능하면 1, 아니면 0을 반환한다.

**5.4. 미로 경로 찾기**

void mazeInit();

DFS, BFS 알고리즘을 이용하여, 해를 찾기 이전에, 미로를 초기화 시켜서, 경로를 모두 없앤다.

bool DFS();

DFS 알고리즘을 이용하여, 경로를 찾는다.

void dfsdraw();

DFS( ) 함수에 의해 찾은 경로와, 이를 위해 방문한 모든 시도를 미로 위에 나타낸다.

bool BFS();

BFS 알고리즘을 이용하여, 경로를 찾는다.

void bfsdraw();

BFS( ) 함수에 의해 찾은 경로와 이를 위해 방문한 모든 시도를 미로 위에 나타낸다.

void freeMemory();

play( ), BFS( ), DFS( ) 함수 시행을 위해 할당한 모든 메모리 영역을 해제하고, 메뉴 설정을 초기화한다.

**6. 본인이 어떻게 창의적 구현을 하였는지에 대해 구체적으로 설명**

수업시간에 2개 주에 걸쳐 생성된 미로 생성 프로그램과 미로 해결 프로그램이 중간고사 이전에 구현하였던 테트리스와 같이 하나의 유기적인 프로그램이 되지 않은 점이 아쉬운 것에서 처음 착안하여 프로젝트를 시작하였다. 이에 따라 본 프로그램은 크게 두 가지 부분으로 확장하였다. (1) 사용자로 부터 미로 파일을 받지 않더라도 지속적으로 새로운 미로를 시현할 수 있다. (2) 사용자가 직접 미로를 찾는 게임을 할 수 있다.

구체적으로1주차에 구현하였던 미로 생성 프로그램이 2주차에 미로를 화면에 그리기 위하여 생성하였던 자료구조에 일치시켜 내부화하였다. 기존에 .maz 확장자를 가진 파일로서 출력하던 방식을 이미 프로그램 내부에 구현되어 있는 동적인 배열으로 출력시켰다. 이에 따라 굳이 사용자가 사전에 다른 과정을 거쳐 미로 파일 (.maz)를 생성하여 준비하지 않더라도 다양한 미로를 시도해볼 수 있다.

또한 DFS, BFS 알고리즘에 의하여 미로의 경로를 찾아 보여주기만 하는 기능이 아쉬워서, 사용자가 직접 방향키를 이용하여 캐릭터를 이동시키는 방식으로 미로 찾기 게임을 해볼 수 있도록 새로운 기능을 추가하였다. 시작점에서 출발하여, 성공적으로 도착지에 도착할 경우에 캐릭터의 모양을 바뀌는 방식으로 미로를 성공적으로 해결하였음을 시각적으로 구현하였다.

끝으로, 기존 ‘comsil project’라는 안내 메시지를 보여주던 부분을 다양한 기능에 따라 사용자의 편의를 위하여, 적합한 메시지를 보여주는 방식으로 수정하였다. 이에 따라 2주 동안 단순히 자료구조와 다양한 알고리즘을 연습하던 목적으로 구현한 프로그램을 테트리스와 같이 실제로 사용자가 사용해볼 수 있는 응용프로그램으로 발전시켰다.

**7. 프로젝트 실행 결과 캡쳐**

Graphical user interface, application, Word

Description automatically generatedDiagram

Description automatically generated

<그림1> 초기화면 <그림2> 새롭게 생성한 미로의 모습

Diagram

Description automatically generatedDiagram

Description automatically generated

<그림3> 미로찾기 게임의 초기화면 <그림4> 미로찾기 게임의 실행화면

Diagram

Description automatically generatedDiagram

Description automatically generated

 <그림5> 미로찾기 게임을 성공적으로 마친 화면 <그림4> DFS 알고리즘에 의하여 미로의 해를 찾은 화면

**8. 느낀 점 및 개선 사항**

한학기 동안 OpenFramwork와 ncurse를 활용하여, 기존 컴퓨터 공학 전공 수업시간에서와 달리 일상생활에서 많이 접하던 GUI 환경에서 구현되는 프로그램을 간단하게 나마 만들어 볼 수 있어 즐거웠다. 특히나 선형적으로 진행되는 기존 프로그램과 더불어서, 사용자가 실제로 마주하는 화면에 계속해서 새로운 모습을 보여주기 위하여 화면의 주사 율에 맞추어 1/60초마다 계속해서 함수를 사용할 수 있어서, 상당히 새로운 기능들을 구현할 수 있음을 알 수 있었다.

앞으로는 미로를 생성하는 알고리즘을 더 깊이 공부하고, 다양한 사용자들이 개발한 Open Frameworks의 모듈을 이용하여 시중에 있을 법한 미로 찾기 게임을 구현해볼 수 있으면 좋을 것 같다. 예를 들어서 현재 단순히 크기로 난이도를 조절하는 것이 아닌, 실질적으로 더 어려운 미로를 제공하거나, 화면에 남은 시간을 표시하고, 좀더 다양한 캐릭터를 제공한다면, 좀더 시각적인 완성도를 높일 수 있을 것으로 생각한다.