**최종 프로젝트 보고서**

20190963 한다현

1. **실험 환경과 프로젝트 목표**

최종 프로젝트의 실험 환경은 다음과 같다.

* visual studio 2022
* OpenFramework
* Debug x64

프로젝트의 목표는 기존 실험에서 작성한 maze 프로젝트에 사용자가 미로를 탈출하는 기능을 추가로 구현하고자 하였다. 메뉴바에 Play라는 메뉴를 추가하고 Play 메뉴를 선택하였을 때 사용자가 시작점부터 도착점까지 키보드의 방향키를 사용해 조작하는 기능을 추가하였다.

1. **각 변수에 대한 설명**

|  |  |
| --- | --- |
| 변수 | 설명 |
| Int HEIGHT | 미로의 높이 |
| Int WIDTH | 미로의 너비 |
| int maze\_col | 미로칸의 열의 개수(방의 열의 개수) |
| Int maze\_row | 미로칸의 행의 개수(방의 행의 개수) |
| int isOpen | 파일이 열렸는지 판단하는 변수 |
| bool isdfs | DFS함수가 실행되었는지 판단하는 변수 |
| Bool isbfs | BFS함수가 실행되었는지 판단하는 변수 |
| Int isplay | Play\_draw와 success\_draw 함수의 실행 여부를 판단하는 변수 |
| Int\* visited | 방문 여부를 저장하는 포인터 |
| Int\* play\_visit | 사용자가 play할 때 방문 여부를 저장하는 포인터 |
| Int\*\* maze | .maz 파일의 데이터를 저장하는 2차원 배열 |
| Int\* path | DFS 탐색에서 모든 경로를 저장하는 포인터 |
| Int num | Path에 저장된 노드의 개수 |
| Int\* best | DFS 탐색에서 최적 경로를 저장하는 포인터 |
| Int best\_num | Best에 저장된 노드의 개수 |
| Int\* q\_path | BFS 탐색에서 모든 경로를 저장하는 포인터 |
| Int num2 | q\_path에 저장된 노드의 개수 |
| Vector<int>\* graph | 노드들의 인접 노드를 저장하는 vector 배열 |
| Vector<int>\* parent | 노드들의 부모 노드를 저장하는 vector 배열 |
| Vector<int> p\_path | 사용자가 play할 때 지나가는 경로를 저장하는 vector |
| Dot\*\* dots | ‘+’의 x, y 좌표를 저장하는 2차원 구조체 배열 |
| Dot\* rooms | 미로의 방의 x, y 좌표를 저장하는 구조체 배열 |

1. **각 함수에 대한 설명**

|  |  |
| --- | --- |
| 함수 | 설명 |
| Void setup() | 프로그램의 초기 설정을 다루는 함수이다. 초기 화면을 설정하고, 메뉴바와 서브 메뉴를 생성한다. |
| Void appMenuFunction(string title, bool bChecked) | 사용자가 선택한 메뉴에 따라 적절한 함수를 실행한다. |
| Void draw() | 미로와 경로 등을 그리는 함수이다. |
| Void doFullScreen(bool bFull) | 전체 화면을 실행하는 함수이다. |
| Void keyPressed(int key) | 특정 키가 press 되었을 때 키에 맞는 적절한 동작을 실행하는 함수이다. |
| Void keyReleased(int key) | 특정 키가 release 되었을 때 키에 맞는 적절한 동작을 실행하는 함수이다. |
| Bool readFile() | 사용자가 open 메뉴를 선택했을 때 실행되는 함수로, .maz 파일을 열고, 파일의 데이터를 적절한 자료구조에 저장한다. |
| Void freeMemory() | 프로그램을 실행하면서 사용한 자료구조의 메모리를 해제하는 함수이다. |
| Bool DFS() | 사용자가 Show DFS 메뉴를 선택하였을 때 실행되는 함수로, DFS 탐색 알고리즘을 통해 미로를 탈출하기 위해 방문한 모든 경로와 최적 경로를 적절한 자료구조에 저장하는 함수이다. |
| Void dfsdraw() | isdfs가 true일 때 draw() 함수에서 호출되는 함수로, DFS() 함수에서 구한 모든 경로와 최적 경로를 화면에 그리는 함수이다. |
| Bool BFS() | 사용자가 Show BFS 메뉴를 선택하였을 때 실행되는 함수로, BFS 탐색 알고리즘을 통해 미로를 탈출하기 위해 방문한 모든 경로와 최적 경로를 적절한 자료구조에 저장하는 함수이다. |
| Void bfsdraw() | isbfs가 true일 때 draw() 함수에서 호출되는 함수로, BFS() 함수에서 구한 모든 경로와 최적 경로를 화면에 그리는 함수이다. |
| Void play\_draw() | isplay가 1일 때 실행되는 함수로, 사용자의 키 입력으로부터 얻은 경로를 화면에 그리는 함수이다. 도착점까지 입력이 완료되면 isplay를 2로 변경한다. |
| Void success\_draw() | isplay가 2일 때 실행되는 함수로, 사용자가 도착점에 도착했을 때 성공적으로 미로를 탈출했다는 것을 화면에 표시하기 위한 함수이다. 미로와 사용자의 경로를 화면에 그린다. |
| Void movement(int next) | keyReleased() 함수에서 키를 입력 받은 후 계산한 다음 노드의 번호를 인자로 받아 다음 노드가 현재 노드와 인접한 노드인지 검사하고 이동 여부를 결정하는 함수이다. |

1. **프로젝트 전체 플로우차트**

텍스트, 도표, 스케치, 기술 도면이(가) 표시된 사진

자동 생성된 설명

텍스트, 도표, 스케치, 기술 도면이(가) 표시된 사진

자동 생성된 설명

1. **자료구조 및 알고리즘과 시공간 복잡도**

* **readFile 함수**

readFile 함수에서는 파일을 열고, 그 내용을 적절한 자료구조에 저장하였다. 이 함수에서 사용된 자료구조는 int 형 2차원 배열과 int x1, int y1을 포함하는 구조체 배열, 그리고 vector 배열을 사용하였다.

먼저, 파일을 한 줄씩 읽으면서 미로의 HEIGHT, WIDTH, maze\_col, maze\_row를 구할 수 있었고, 미로 데이터에서 ‘+’의 개수는 (maze\_row+1)\*(maze\_col+1)의 개수와 같으므로 행의 maze\_row+1이고, 열이 maze\_col+1인 2차원 구조체 배열 dots를 동적할당 받는다. 가장 처음, 즉 좌측 상단에 위치한 ‘+’의 좌표가 (0,0)이므로 dots[0][0].x1과 dots[0][0].y1에 0을 저장한다. 그리고 첫 번째 행에 위치한 ‘+’는 왼쪽에 위치한 ‘+’의 위치보다 x축으로만 이동하므로 for 반복문은 사용해 x1, y1의 값을 저장한다. 첫 번째 행의 위치를 구했으면 두번째 행부터는 이전 행의 ‘+’ 좌표보다 y 값만 변화하므로 이중 for loop을 사용해 이전 ‘+’ y 값에 30을 더한 값을 저장한다. 이 자료구조는 draw 함수에서 미로를 그릴 때 사용된다.

rooms는 방의 위치를 저장하는 일차원 구조체 배열이다. dots와 마찬가지로 Dot 구조체를 사용하며 maze\_row\*maze\_col의 크기로 동적할당 받는다. 0번째 방의 위치는 첫번째 ‘+’과 두번째 ‘+’ 사이인 (15,15)이다. dots에서 했던 방식과 비슷하게, 첫 번째 행에 존재하는 방들은 0번째 방에서 x 값만 증가시키고, 두 번째 행 이후부터 존재하는 방들은 y 값만 증가시켜 전체 방의 위치를 구할 수 있다.

maze는 미로의 데이터를 저장하는 이차원 배열이다. maze를 HEIGHT만큼의 행과 WIDTH 만큼의 열로 동적할당 받는다. 이 후 .maz 파일을 한 줄씩 읽어오면서 maze 배열에 저장하는데, ‘+’, ‘-‘, ‘|’과 같이 미로의 벽을 표시하는 문자는 그대로 이차원 배열에 저장하고, ‘ ‘(space)를 저장할 때는 vtx를 1씩 증가시키면서 maze에 vtx를 저장한다. 이렇게 되면 maze라는 int형 이차원 배열에 방 번호와 ‘+’, ‘|’, ‘-‘를 저장할 수 있다.

graph는 특정 방과 인접한 방의 번호를 저장하는 벡터 배열이다. 이 배열의 크기는 앞서 구한 vtx+1만큼인데, 예를 들어 미로의 방의 개수가 25개라면 마지막 방을 나타내는 vtx는 24일 것이다. graph를 동적할당 받은 이후에는 3중 for loop이 실행된다. maze라는 이차원 배열을 검사해야하므로 2중 for loop이 실행되고, maze에서 찾은 방마다 상하좌우 4 방향에 대해 인접한 방이 있는지 검사해야하기 때문이다. maze에서 방의 번호가 저장되어 있는 곳은 홀수행과 홀수열인 경우이다. 방의 번호를 찾은 후에는 해당 방(w)이 상하좌우에 다른 방(u)과 연결되어 있는지 검사해야 하는데, 상하좌우에 벽이 존재하지 않는 경우에 graph[w]에 u를 추가한다. 이러한 방식을 반복문을 통해 반복하여 특정 방과 인접한 방을 모두 찾을 수 있다.

앞서 말했듯이 이 함수에서 사용된 자료구조는 int 형 2차원 배열과 int x1, int y1을 포함하는 구조체 배열, 그리고 vector 배열을 사용하였다. 따라서 이 함수에서 사용된 공간 복잡도는 O(n^2)이다. 또한 graph를 구하는 과정에서 3중 for loop을 사용하였다. 하지만 3중 for loop중 가장 안쪽에 있는 반복문은 상하좌우를 검사하기 위해 0부터 4까지 고정된 수만큼 반복한다. 따라서 이 함수의 시간 복잡도는 O(n^2)이다.

* **DFS 알고리즘과 dfsdraw 함수**

DFS에서 사용한 자료구조는 stack과 int형 배열인 visited, path, best이다. 먼저 시작 노드인 0을 stack에 추가하고, 0번 방의 방문 여부를 1로 변경하고 지나간 경로를 저장하는 path에도 0을 추가한다. 그리고 시작점부터 도착점까지 방문하게 되는 모든 경로를 찾는 while문을 stack이 비어있지 않은 동안 반복한다. while문 내에서는 먼저 stack의 top이 도착점과 동일하면 반복문을 종료한다. 그렇지 않은 경우에는 인접한 노드 중에 방문하지 않은 노드를 찾아야 하는데, 그러기 위해서 stack의 top과 인접한 노드가 1개 이상인지 확인하고, 그런 경우에는 인접한 노드마다 방문한 적이 있는지 검사해서 방문한 적이 없는 노드를 stack과 path에 추가한다. 방문하지 않은 인접한 노드가 없는 경우에는 stack의 마지막 원소를 반환한다. 이러한 과정을 while문이 종료될 때까지 반복하게 되면 방문한 모든 경로를 찾을 수 있다. while문이 종료되었을 때, 즉 stack의 top이 도착점인 경우에 stack에 남아있는 방의 번호들이 최적 경로가 된다. 따라서 다시 stack의 비어있는 동안 반복하는 while문을 실행하여 stack에 있는 노드들을 best에 추가함으로써 최적 경로를 얻을 수 있다. 이 과정까지 종료되면 isdfs를 true로 변경하고 함수를 종료한다.

dfsdraw 함수에서는 먼저 path의 길이만큼 for문을 실행하여 화면에 모든 경로를 그린다. 그 후 최적 경로를 표시하기 위해 선의 색깔과 두께를 변경한 후 best의 길이만큼 for 문을 실행하여 화면에 최적 경로를 그린다.

DFS 알고리즘에서는 stack과 1차원 배열을 사용하였기 때문에 DFS 알고리즘의 공간 복잡도는 O(n)이 된다. 또한 DFS 알고리즘의 첫 번째 while 반복문 내에서 인접한 노드의 방문 여부를 검사하기 위해 for 문을 사용하였으므로 이중 반복문이 사용되었다. 따라서 DFS 알고리즘의 시간 복잡도는 O(n^2)이다.

* **BFS 알고리즘과 bfsdraw 함수**

BFS 알고리즘에서는 queue와 1차원 배열인 visited, q\_path, 그리고 vector 배열인 parent를 사용하였다. 먼저 visited와 q\_path를 동적할당 및 초기화하고, queue와 q\_path에 시작점인 0을 추가한다. 그 후 BFS 알고리즘을 실행하면서 시작점부터 도착점까지 방문하게 되는 모든 경로를 찾는 while 반복문을 실행한다. 이 while 반복문은 queue가 비어있지 않은 동안 실행하고, queue의 rear가 도착점일 때 종료된다. while 문 내에서 w에 queue의 front를 저장하고, queue의 front 원소를 pop한다. 이 후 for loop을 사용하여 w와 인접한 노드 중에서 방문한 적이 없는 노드를 찾는다. 이 for loop은 w와 인접한 노드의 개수만큼 반복되고, 인접한 노드 중 방문한 적이 없는 노드는 queue와 q\_path에 추가하고, 인접한 노드의 parent에 w를 추가한다. while문이 종료되면 isbfs를 true로 변경하고 함수를 종료한다.

bfsdraw 함수에서는 모든 경로를 그리기 위해서 q\_path만큼 반복하는 for loop을 실행한다. for 문 안에서는 다시 while문을 사용해야 하는데, q\_path에 존재하는 노드마다 부모 노드를 따라가며 경로를 출력해야 하기 때문이다. q\_path에 있는 모든 노드의 부모 노드를 따라가며 경로를 그린 후 for loop이 종료되면, 최적 경로를 그리기 위해 선의 색깔과 두께를 변경한다. 그 후 다시 while 문을 실행하는데, 이 때 while 문은 q\_path의 마지막 노드의 부모 노드를 따라가며 경로를 그리는 것이다. q\_path의 마지막 노드가 도착점을 의미하기 때문에 도착점의 부모 노드를 따라가며 경로를 그리는 것이 최적 경로를 그리는 것이다.

BFS 알고리즘을 사용하여 전체 경로와 최적 경로를 그릴 때 queue, 1차원 배열, vector 배열을 사용하였다. parent라는 vector 배열이 사용되었지만 한 노드의 부모 노드는 하나만 존재하므로 이 알고리즘의 공간 복잡도는 O(n)이다. 또한 BFS에서 모든 경로를 사용할 때 for loop과 while loop으로 구성된 이중 반복문을 사용하였고, dfsdraw에서도 전체 경로를 그릴 때 for loop과 while loop으로 구성된 이중 반복문을 사용하였으므로 시간 복잡도는 O(n^2)이다.

새로 구현한 기능의 알고리즘과 자료구조에 대해서는 6번 항목에서 설명하겠다.

1. **창의적 구현 항목**

최종 프로젝트에서 새롭게 추가한 항목은 사용자가 직접 방향키를 입력하여 미로를 탈출하는 기능이다. Play 메뉴를 선택하면 먼저 시작점과 도착점이 표시된다. 이 후 사용자가 방향키를 입력하여 탈출 경로를 찾는다. 이 때 인접하지 않은 방, 즉 벽이 있는 방향으로는 이동할 수 없으며 이동이 가능한 방향의 키를 입력하면 그 방향으로 경로가 표시된다. 또한 이미 지나온 경로로 다시 되돌아 가는 경우에는 그 방향으로 경로가 삭제된다.

이 기능을 구현하기 위해서는 먼저 setup() 함수에서 메뉴바에 Play 메뉴를 추가하고, Play 메뉴를 선택했을 때 다시 Play라는 서브 메뉴가 보이도록 추가하였다. appMenuFunction() 함수에서사용자가 Play 메뉴를 선택했을 때의 작업을 추가하였다. 우선 이전 play에 대한 데이터를 지우기 위해 isplay를 0으로 설정하고, 이전 play에서 사용했던 p\_path를 비운다. 그리고 이전에 Show BFS나 Show DFS를 사용하였을 수 있으므로 isdfs와 isbfs를 false로 설정한다. 이 후 isplay를 1로 변경하여 사용자가 play할 수 있도록 하고, play\_visit을 0으로 초기화한 후 p\_path에 0을 추가한다. keyReleased() 함수에서는 isplay가 1일 때 관련 작업이 실행된다. 사용자가 play 메뉴를 선택하면 isplay가 1이 되므로 사용자가 입력하는 방향키에 대한 작업을 실행할 수 있다. keyReleased 함수에서는 방향키에 따라 이동할 다음 방을 구한다. 아래 방향키를 입력하면 next는 p\_path의 마지막 노드에 maze\_col을 더한 것이 되고, 위 방향키를 입력하면 next는 p\_path의 마지막 노드에 maze\_col을 뺀 것이 된다. 오른쪽 방향키를 입력하면 p\_path의 마지막 노드에 1을 더한 것이 되고, 왼쪽 방향키를 입력하면 p\_path의 마지막 노드에 1을 뺀 것이 된다. 이렇게 next를 구한 후 movement() 함수를 호출한다. movement 함수에서는 keyReleased에서 구한 next를 인자로 받게 되고, 이 next를 사용해 next 노드가 현재 노드와 인접한 노드인지 확인한다. 확인하는 과정은 graph[현재노드]의 처음과 끝, 즉 현재 노드와 인접한 노드들이 저장된 graph[현재노드] 중에 next와 일치하는 번호가 있는지 찾고, graph[현재노드]에 존재하는 경우, next가 이미 방문한 적이 있는 노드인지 확인한다. 방문한 적이 없는 노드라면 p\_path에 next를 추가하고, 방문한 적이 있는 노드라면 현재 방의 방문 노드를 0으로 하고, 현재 노드를 p\_path에서 제거한다. p\_path에서 제거하는 이유는 next가 이미 방문한 노드라는 것이 현재 노드에서 next로 되돌아 가는 것을 의미하기 때문이다. draw 함수에서는 isplay가 1이면 시작점과 도착점에 각각 초록색과 빨간색 점을 그려 시작점과 도착점을 표시하고 play\_draw 함수를 호출한다. play\_draw 함수에서는 p\_path의 사이즈만큼 for loop을 반복하는데, for loop 안에서는 현재 노드와 다음 노드 사이의 선을 그린다. 이 과정을 반복하여 반복문이 종료되면 사용자가 현재까지 이동한 경로를 화면에 그릴 수 있다. 또한 p\_path의 마지막 노드가 도착점이 되면 isplay를 2로 변경한다. draw 함수에서 isplay가 2인 경우에는 success\_play를 호출한다. success\_play 함수는 성공적으로 미로를 탈출하였을 경우 초록색으로 미로를 그리고, 파란색으로 탈출 경로를 표시한다. 미로를 그리는 과정은 선의 색깔을 초록색으로 변경한 후 파일을 처음 열었을 때 미로를 그리는 과정과 동일하게 두 개의 ‘+’ 사이의 선을 이중 for loop을 사용해 그린다. 미로를 그린 후에는 선의 색깔을 파란색으로 변경하고 p\_path의 크기만큼 반복하는 for loop을 통해 현재 위치와 다음 위치 사이의 점을 그리며 경로를 화면에 표시한다.

이 과정에서 자료구조는 p\_path라는 vector와 play\_visit이라는 1차원 배열을 사용하였기 때문에 공간 복잡도는 O(n)이다. 또한 미로를 그리는 과정에서 이중 for loop을 사용하였으므로 시간 복잡도는 O(n^2)이다.

1. **프로젝트 실행 결과**

텍스트, 소프트웨어, 스크린샷, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명 Open 메뉴 선택했을 때

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명 Show DFS 메뉴 선택했을 때

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명 Show BFS 메뉴 선택했을 때

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명 Play 메뉴 선택했을 때

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명 Play에서 방향키를 사용하여 경로 이동

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명 위 상태에서 아래 방향키 입력했을 때

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명 위 상태에서 왼쪽 방향키 입력했을 때

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 미로 탈출에 성공했을 때

1. **느낀 점 및 개선 사항**

maze 프로젝트를 진행하면서 OpenFramework에 많이 익숙해질 수 있었다. 특히 선과 점을 화면에 그리는 ofDrawLine이나 ofDrawCircle, 색깔과 두께를 바꾸는 ofSetColor와 ofSetLineWidth를 많이 사용하면서 익숙해질 수 있었다. 최종 프로젝트를 구현하면서는 특히 keyReleased 함수에 대해 많이 학습하였다. 방향키를 입력 받고 그에 따라 적절한 작업을 keyReleased 함수를 통해 구현할 수 있었다. 자료구조에 대해서는, 기존에는 linked list를 많이 사용하였지만 c++ 언어를 사용하면서 vector나 vector 배열, stack, queue와 같이 다양한 자료구조를 사용하였다. 기존에 C언어를 사용할 때는 접하지 못했던 자료구조라서 사용하는 데에 어려움이 있었지만 사용하다보니 linked list를 사용할 때보다 더욱 편리하게 사용할 수 있었다.

개선 사항으로는 최종 프로젝트에서 구현한 play 기능에 추가적인 기능을 추가하는 것을 생각해볼 수 있을 것 같다. 사용자가 이동한 경로를 모두 표시하는 것이 아니라, 캐릭터를 생성하여 캐릭터가 미로를 탈출하는 기능을 구현하거나, 미로를 탈출했을 때 미로의 색깔을 바꿔서 그리는 것이 아니라, 조금 더 화려한 그림을 출력하는 것을 생각해 볼 수 있다. 또한 미로를 탈출하기 위해 사용자가 움직인 횟수를 통해 점수를 측정할 수 있을 것이다.