컴실1 기말 프로젝트

Teleportation maze

20201585 박준하

1. 프로젝트 목표

이번 프로젝트는 기존 실습 때 진행하였던 미로 생성 알고리즘 중 하나인 Eller’s algorithm을 변형하여 teleportation platform이 존재하는 미로 생성 알고리즘을 제작하는 것을 목표로 하였다. 구체적으로, 미로 내에 여러 쌍의 teleportation platform이 존재해 한 platform 위에 올라서면 그에 대응하는 다른 platform으로 순간이동하는 perfect maze를 생성하는 알고리즘이 목표이다.

나아가, Eller’s algorithm과 동일하게 변형한 알고리즘 또한 perfect maze의 성질을 가지기에 DFS로 미로의 경로를 탐색하는 기능을 구현하는 것 또한 추가 목표로 삼았다.

2. 자료구조 및 알고리즘에 대한 설명

프로그램을 실행시키고 메뉴에서 Make를 선택하면 미로가 생성됨과 동시에 화면에 그려진다.

미로가 생성되는 알고리즘은 makeTeleMaze() 함수에서 시작된다. 이 함수는 먼저 미로의 높이와 너비를 입력받은 후 미로 생성과 관련된 다양한 자료구조를 초기화하는 initializeMaze() 함수를 호출한다. 그리고 미로의 마지막 행에 도달하기 전까지 미로의 첫번째 행부터 각 행마다 일련의 과정을 수행한다.

Eller’s algorithm의 경우, 과정은 크게 3단계로 이루어진다.

1. 미로의 번호가 부여되지 않은 방마다 각각 새로운 집합의 번호를 부여한다.

2. 방의 번호가 오른쪽 방의 번호와 다르다면 확률적으로 두 방 사이의 벽을 허문다.

이때, 두 방이 각각 속한 두 집합을 하나로 합친다.

만약 방의 번호가 같을 경우 허물지 않는다.

3. 각 방의 아래쪽 벽을 확률적으로 허문다. 허문 벽의 아래쪽 방은 그 위쪽 방의 집합에 속하도록 한다.

이때, 한 집합에 속한 방이 모두 아래쪽 벽이 존재한다면 그 집합 중 하나의 방을 랜덤하게 선택해 아래쪽 벽을 허문다.

Teleportation platform이 존재하도록 하는, 변형한 알고리즘의 경우 이 과정이 달라진다.

변형된 Eller’s algorithm의 경우, 과정은 크게 5단계로 이루어진다.

1. 미로의 번호가 부여되지 않은 방마다 각각 새로운 집합을 부여한다.

2. 방의 번호가 오른쪽 방의 번호와 다르다면 확률적으로 두 방 사이의 벽을 허문다.

이때, 두 방이 각각 속한 두 집합을 하나로 합치고 두 집합에 속한 텔레포트 발판 후보군 역시 하나의 집합으로 합친다.

만약 방의 번호가 같을 경우 허물지 않는다.

3. 각 방의 아래쪽 벽을 확률적으로 허문다. 허문 벽의 아래쪽 방은 그 위쪽 방의 집합에 속하도록 한다.

이때, 한 집합에 속한 방이 모두 아래쪽 벽이 존재한다면 그 집합 중 하나의 방을 랜덤하게 선택해 아래쪽 벽을 허문다.

4. 만약 방이 3개의 벽으로 둘러싸여 있다면 그 좌표를 텔레포트 발판 후보군 집합에 추가시킨다.

5. 서로 다른 집합에서 두개의 텔레포트 발판 후보를 선택한 뒤, 이를 활성화시킨다.

이때, 두 텔레포트 발판이 속한 두 후보군 집합과 방의 집합을 합친다.

일반적인 Eller’s algorithm과 달라진 점은 2번 과정에서 오른쪽 벽을 없앴을 때 방의 집합이 합쳐짐에 따라 후보군 또한 하나의 집합으로 합쳐지는 과정이 추가되었다는 점, 그리고 텔레포트 발판의 후보를 생성하고 두 후보를 활성화시키는 4, 5번 과정이 추가되었다는 점이다.

이러한 미로 생성 알고리즘을 통해 미로를 생성하고 나면 미로를 DFS를 통해 탐색할 수 있는 기능 또한 존재한다. 이는 DFS과정 중 미로의 방의 상하좌우와 더불어 teleport platform을 통해 갈 수 있는 또 하나의 길을 추가로 고려함으로써 구현하였다.

미로 생성 알고리즘의 대략적인 플로우 차트와 4, 5번 과정의 플로우 차트는 아래와 같다.

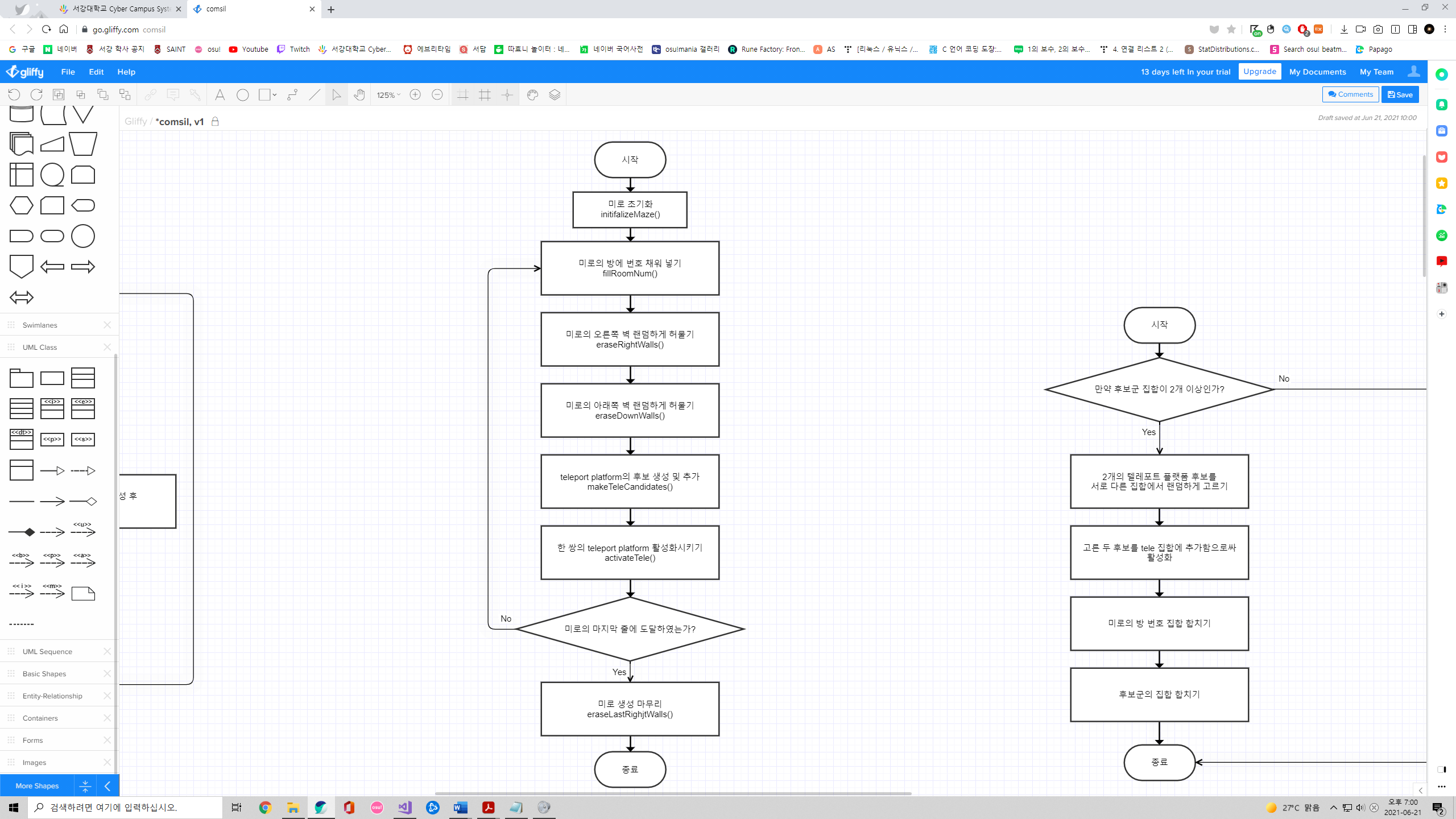


그림 1 – 대략적인 flow chart

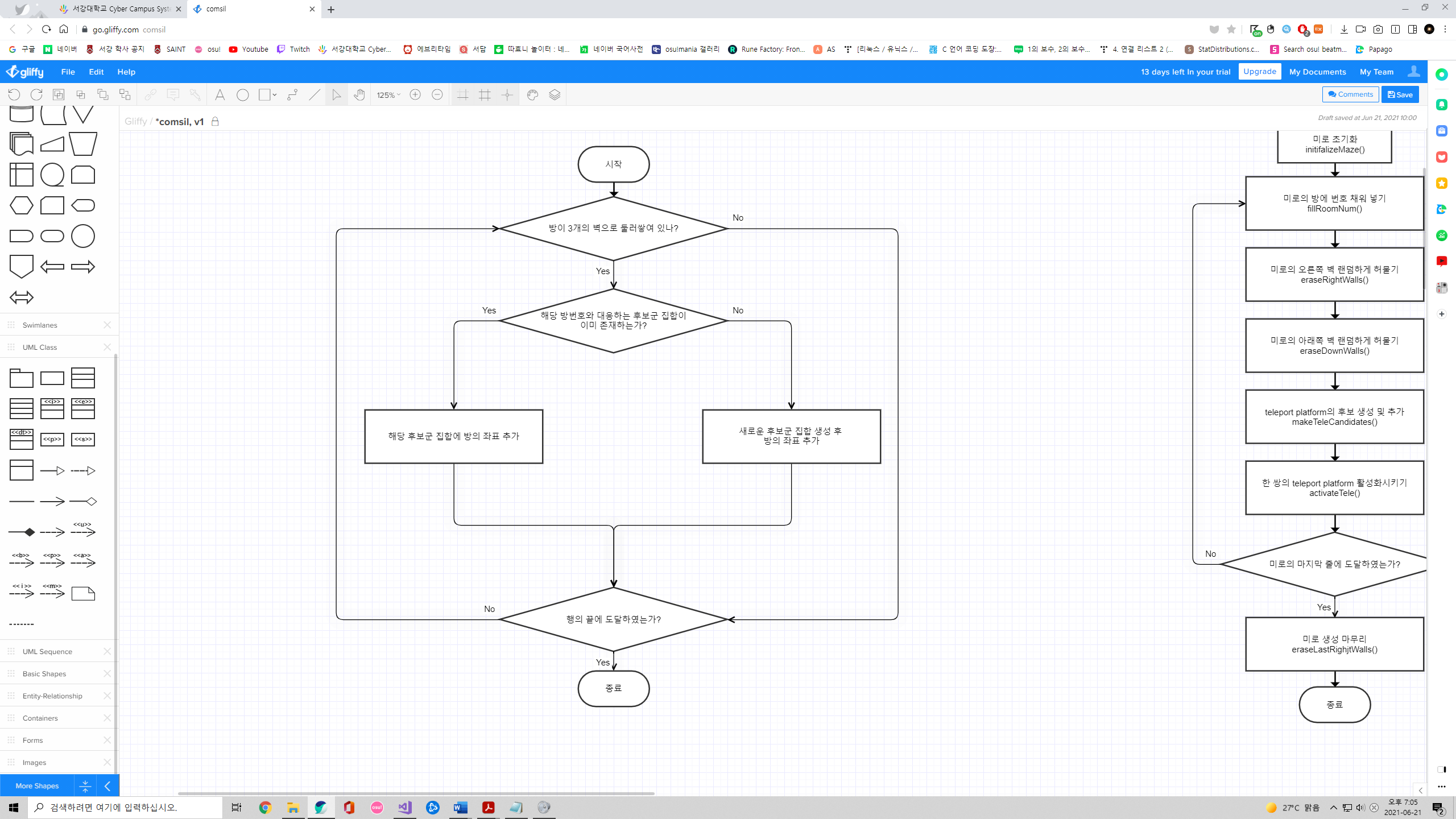


그림 2 – makeTeleCandidates() (4번 과정)의 flow chart

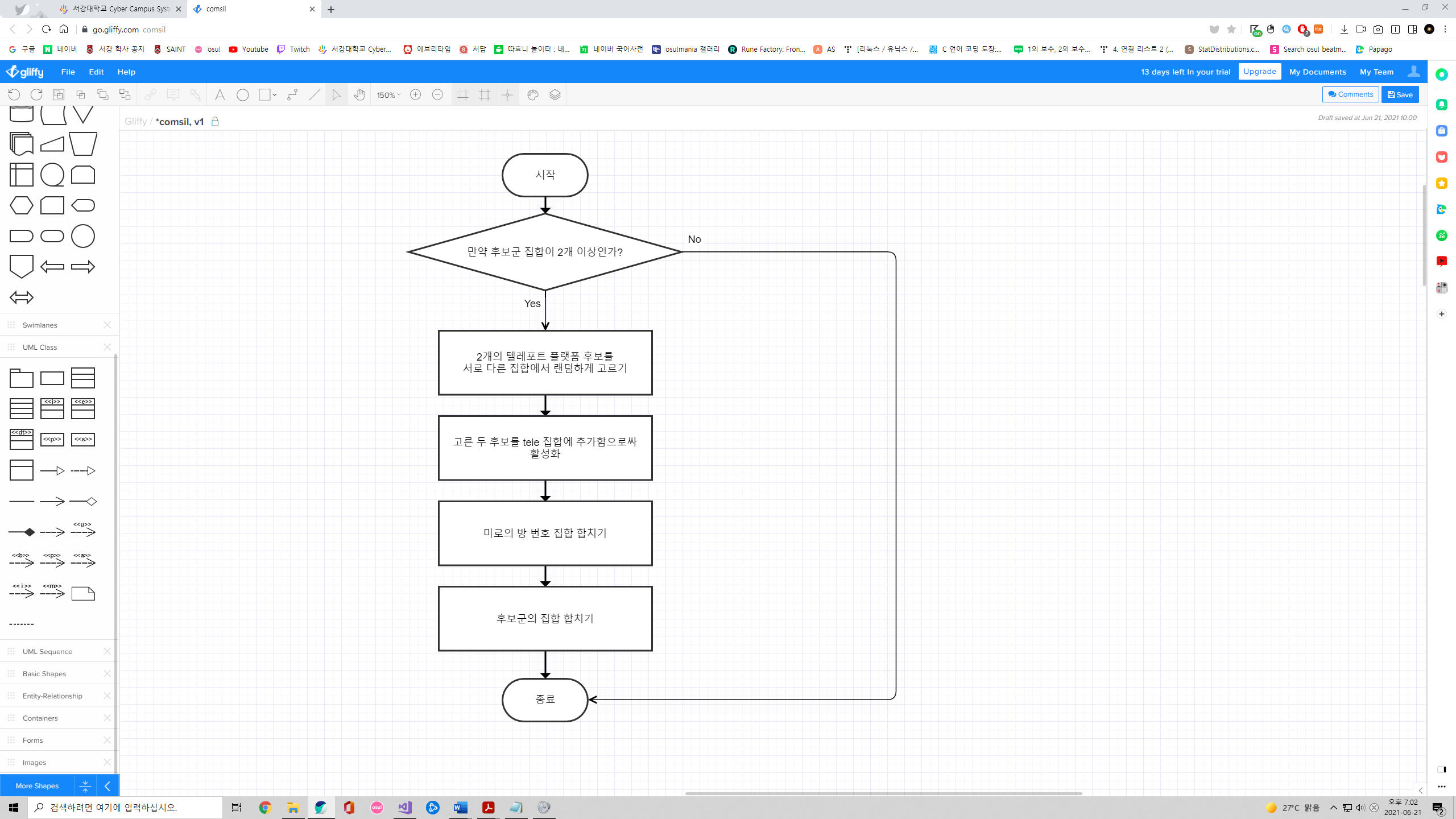


그림 3 – activateTele() (5번 과정)의 flow chart

이러한 알고리즘을 구현하기 위해 사용한 자료구조는 다음과 같다.

|  |  |
| --- | --- |
| MazeRoom \*\*maze | 미로의 벽 정보와 setNumber를 저장하기 위한 2차원 구조체 배열 |
| TeleRoom \*\*tele | 미로에 존재하는 teleportation platform의 정보를 저장하기 위한 2차원 구조체 배열 |
| Vector <TeleCandidateSets> candidateSets | teleportation platform의 후보들을 setNumber에 따라 저장하기 위한 구조체 백터 |
| Vector <Coordiate> s | Iterative DFS를 위한 스택 |

위 표에서 언급된 구조체는 다음과 같이 정의되어 있다.

typedef struct \_Coordinate {

int y; //방의 y좌표

int x; //방의 x좌표

} Coordinate;

typedef struct \_MazeRoom {

int roomNum; //방이 속한 집합의 번호

bool visited; //DFS 중 방문하였는가?

bool hasRightWall; //오른쪽 벽이 있는가?

bool hasDownWall; //아래 벽이 있는가?

} MazeRoom;

typedef struct \_TeleRoom {

bool isTele; //teleport platform인가?

int teleNum; //teleport platform의 번호

int gotoX; //teleport 되는 x좌표

int gotoY; //teleport 되는 y좌표

} TeleRoom;

typedef struct \_TeleCandidateSets {

int setNum; //후보군이 속한 집합의 번호

vector <pair<int, int>> coordinates; // 해당 집합에 속한 teleport platform의 후보군

} TeleCandidateSets;

3. 핵심 변수에 대한 설명

|  |  |
| --- | --- |
| MazeRoom \*\*maze | 미로의 벽 정보와 setNumber를 저장하기 위한 2차원 구조체 배열 |
| TeleRoom \*\*tele | 미로에 존재하는 teleportation platform의 정보를 저장하기 위한 2차원 구조체 배열 |
| Vector <TeleCandidateSets> candidateSets | teleportation platform의 후보들을 setNumber에 따라 저장하기 위한 구조체 백터 |
| Vector <Coordiate> s | Iterative DFS를 위한 스택 |
| Int height | 미로의 높이 |
| Int width | 미로의 너비 |
| Int lastRoomN | Maze 배열의 cell에서 가장 최근에 추가한 room의 번호 |
| Int lastTeleN | Tele 배열의 cell에서 가장 최근에 추가한 teleport platform의 번호 |
| Int isOpen | 미로가 생성되었는지 판단하는 변수 |
| Int isDFS | DFS함수를 실행시켰는지 판단하는 변수 |
| Int isFirst | 처음으로 미로를 생성한 것인지 판단하는 변수 |

4. 각 함수에 대한 설명

|  |  |
| --- | --- |
| Void setup(); | 프로그램이 시작된 후 전역 변수 초기화 및  윈도우 설정 |
| Void draw(); | 미로가 생성되었을 경우, 미로를 그린다. |
| Void freeMemory(); | 미로 생성 알고리즘 중 사용되었던 메모리를 해제시킨다. |
| Bool DFS(); | 미로가 생성되었을 경우, DFS를 진행한다. |
| Void DFSdraw(); | DFS가 성공적으로 이루어졌을 경우, DFS 과정과 결과를 화면에 그린다. |
| Void printMaze(); | 미로를 출력하고 maze.maz 파일로 내보낸다. |

|  |  |
| --- | --- |
| Void makeTeleMaze(); | 미로 생성 알고리즘을 시작하는 함수. 아래 함수들을 여러 번 호출한다. |
| Void initializeMaze(); | 알고리즘과 관련된 변수들을 초기화한다. |
| void fillRoomNum(int rowN); | rowN번째 행의 초기화되어있지 않은 room들을 초기화시킨다. |
| Void eraseRightWalls(int rowN); | rowN번째 행의 오른쪽 벽을 확률에 따라 허물고 이에 따라 room과 teleport platform 후보의 집합을 갱신한다. |
| void eraseDownWalls(int rowN); | rowN번째 행의 아래 벽을 확률에 따라 허문다. |
| void makeTeleCandidates(int rowN); | rowN번째 행의 teleport platform 후보들을 탐색하고 이를 집합에 추가한다. |
| void activateTele(int rowN); | 한 쌍의 Teleport platform을 활성화시키고 이에 따라 room과 teleport platform 후보의 집합을 갱신한다. |
| void eraseLastRightWalls(); | 미로의 마지막 행에서 오른쪽 벽을 규칙에 따라 허문다. |

5. 프로젝트 실행 화면

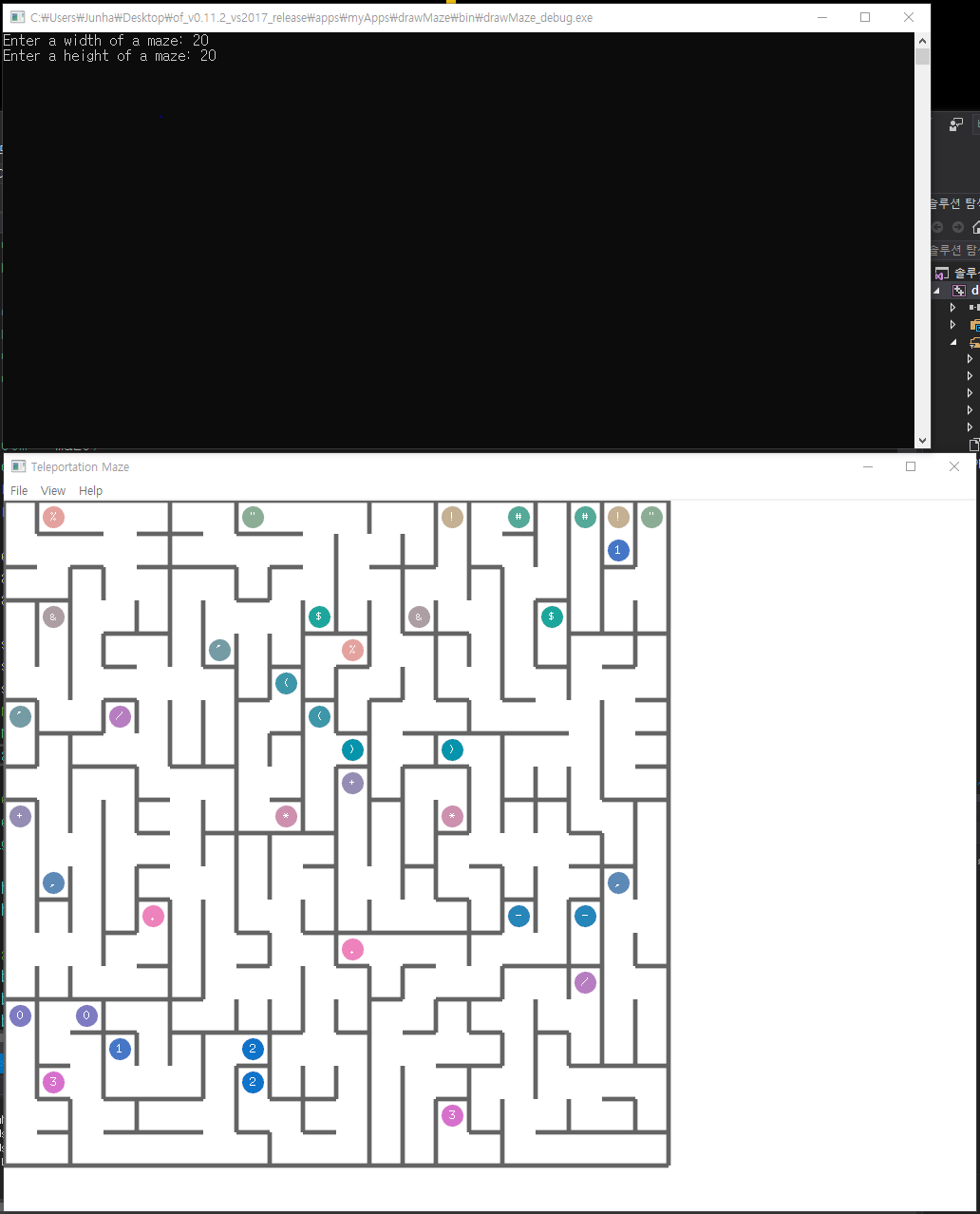


그림 4 – 20x20 미로를 생성한 모습

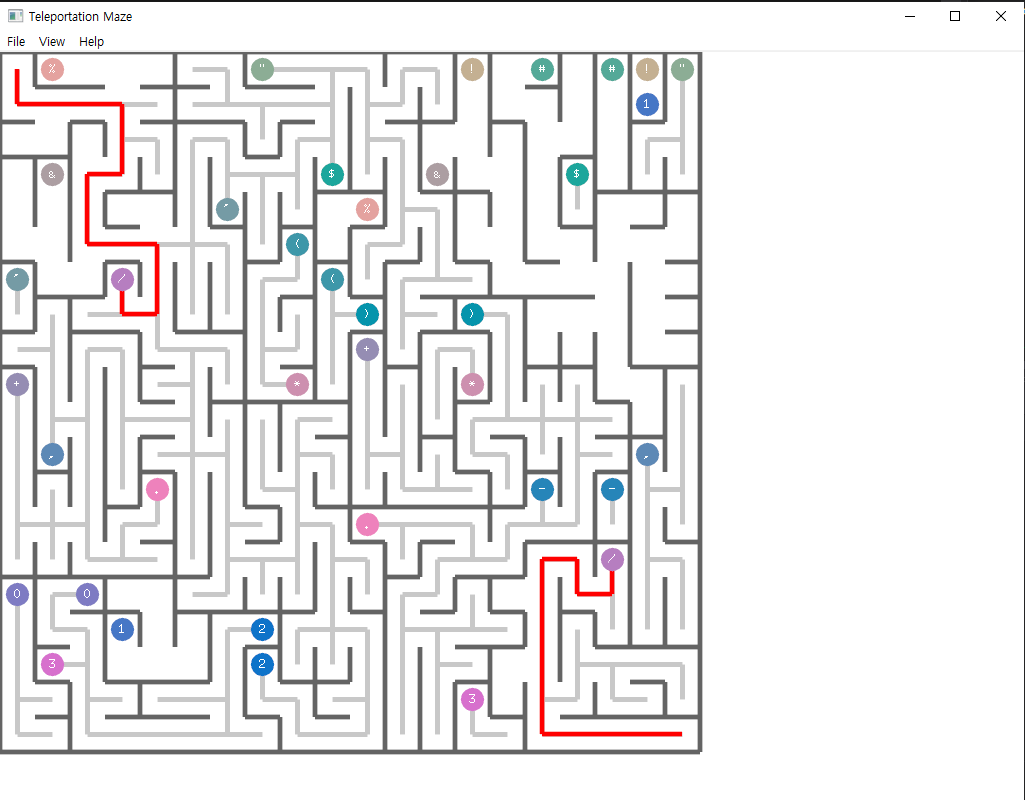


그림 5 – 생성한 미로에서 DFS를 진행한 모습

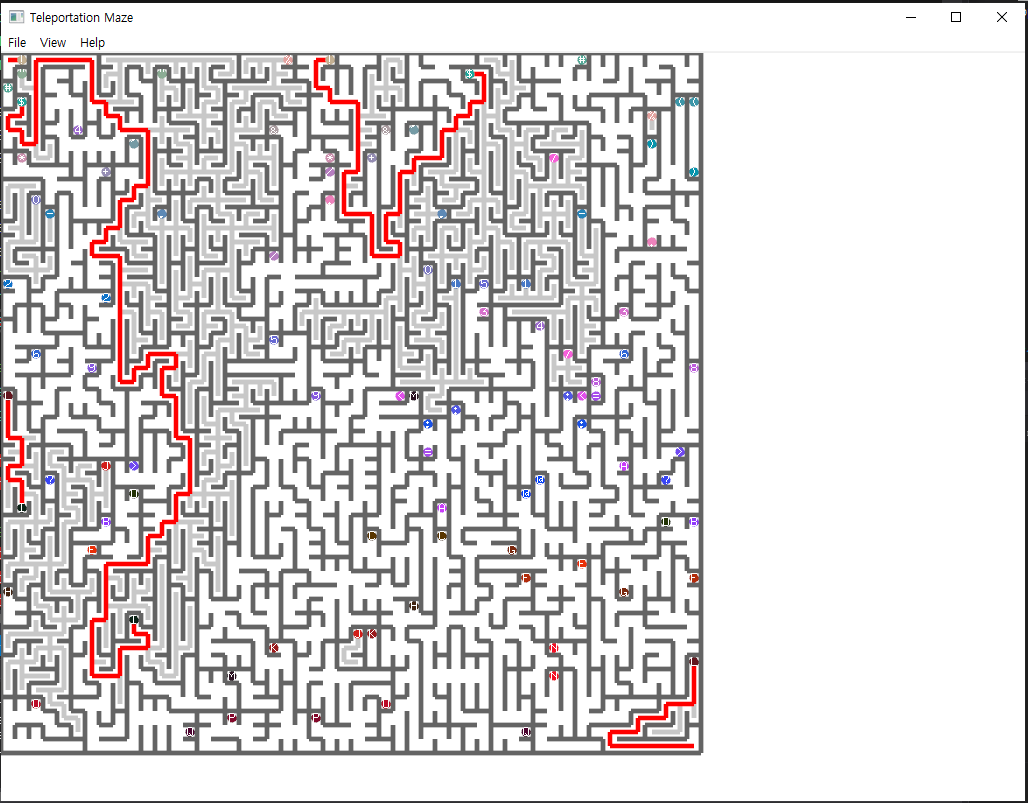


그림 6 – 50x50 미로 생성 후, DFS를 진행한 모습