A* (A star) 알고리즘 구현

출발 지점에서 목표 지점까지의 최단 경로를 찾아내는 그래프 탐색 알고리즘
그래프 상의 최단 경로를 찾는 Dijkstra 알고리즘에 거리 기반 휴리스틱을 결합한 방식

주요 변수

G : 지금까지 이동한 거리
H : 휴리스틱 (현재 노드에서 목표 노드까지의 거리)
F : G + H (경로의 총 이동 비용)
open 리스트 : 앞으로 탐색할 노드 목록
closed 리스트 : 탐색이 완료된 노드 목록
current 노드 : 현재 대상 노드

탐색 절차

1. 시작노드를 open 리스트 에 추가
    * 시작 노드의 G,H,F 계산

2. Open 리스트에서 F 값이 가장 낮은 노드 선택
    * 선택한 노드를 **current** 로 설정
    * 선택된 노드가 목표 지점이면 *경로 재구성 (reconstruct path)

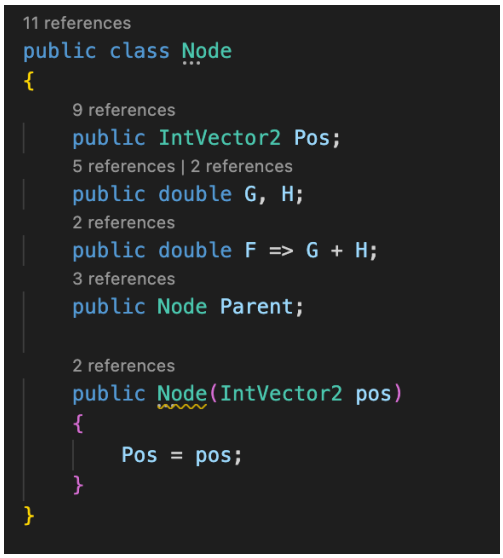
3. **current** 노드를 Open 리스트에서 제거하고 Closed 리스트에 추가
    * 이 노드는 다시 탐색하지 않도록

4. **current** 노드의 이웃 노드 (이동 방향의) 들을 검사
    * 각 이웃에 대해
        1) 맵 경계 안에 있는가?
        2) 이동가능한 칸인가?
        3) closed 리스트에 이미 있는가?
        4-1) open 리스트 에 들어있는가?
            * G 를 계산하여 기존의 G 보다 작다면 더 나은 경로임
                > G 값 업데이트
                > 해당 노드의 Parent 에 **current**를 대입

5. Open 리스트가 비어있지 않다면 2번부터 수행
    * 목표에 도달하거나 Open 리스트가 빌 때 까지 반복.
    * Open 리스트가 빌 때 까지 반복했는데 목표지점이 아니라면
        경로가 없는것

* 출발 지점으로부터 리스트가 시작되도록 정렬 해주는 과정

* 경로를 만들기 위한 노드 타입
  * 위치
  * G , H 값
  * Parent 노드 참조

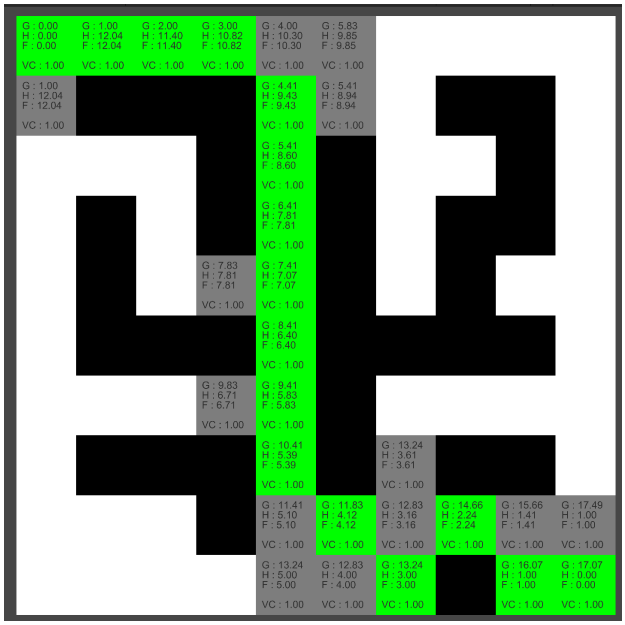


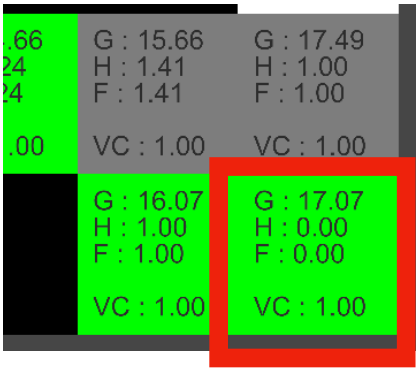
```
11 references
public class Node
{
    9 references
    public IntVector2 Pos;
    5 references | 2 references
    public double G, H;
    2 references
    public double F => G + H;
    3 references
    public Node Parent;

    2 references
    public Node(IntVector2 pos)
    {
        Pos = pos;
    }
}
```

휴리스틱에 관해..

노드 선택의 기준이 F = H 인 대신, F = G + H 인 이유 (why not Greedy Best-First Search)
-> 지금까지의 이동거리를 선택 기준에 합산하지 않으면, 전체 경로가 더 길어질 수 있다.
-> 목표와의 직선거리가 가까운 노드를 선택하는 경우, 경로 계산 횟수 자체는 더 적을 수 있지만 계산된 결과가 최적의 경로가 아닐 수 있음.

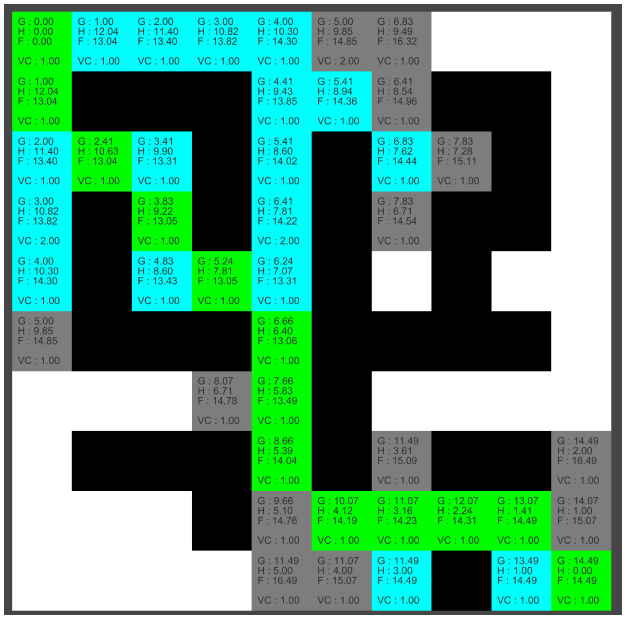
F = H 인 경우의 결과

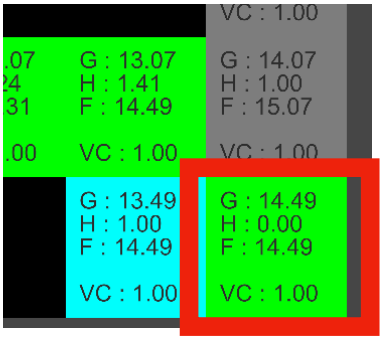




이동경로의 길이 G = 17.1

F = G + H 인 경우의 결과





이동경로의 길이 G = 14.5