인공지능 hw #1
2014005123 정두진

1. 코드설명

1-1 Class

- Node : Priority Queue 에 담아야 하는 노드의 정보 (node의 좌표, 목표좌표까지의 weight 등)
- Priority Queue : 원래 힙 구조로 만들어야 하지만 그냥 정렬된 배열에서 가장 우선순위가 빠른것을 뽑게 구현
- Maze : 현재 미로에 대한 정보와 미로 explore

1-2 사용한 라이브러리

- import math : euclidean 구현 시 sqrt를 위해 사용

1-3 Flow

- 1. 소스가 실행 될 때 Maze Class 를 instance하고 run 함수를 호출한다.
- 2. run 함수에서 maze를 파일에서 read 해와서 메모리에 저장하고 explore 함수를 호출한다.
- 3. explore 함수에서 현재 미로의 시작, key, 목적지 좌표를 얻은 후 지정한 층의 길찾기 함수를 호출한다
  ex) fifth_floor(arg…)
- 4. 길찾기 함수가 끝나면 미로의 최적 탐색, 탐색 노드, 미로 정보를 파일에 write한다

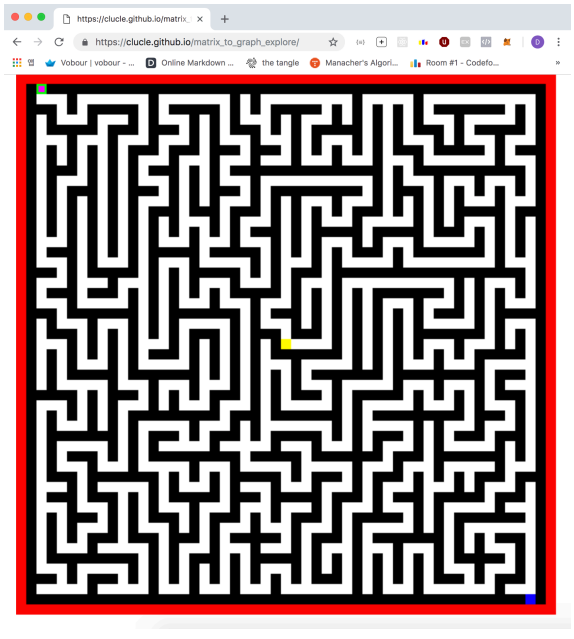
1-4 Algorithm

first_floor : greedy euclidean
second_floor : greedy manhattan
third_floor : greedy euclidean
fourth_floor : a star euclidean cube
fifth_floor : greedy euclidean

각 층마다 알고리즘을 왜 선택했는지
대부분 step보다는 결과가 가까운 쪽으로 가는 것이 더 유리한 것을 눈에 보이게 만들어서 파악했다.
https://clucle.github.io/matrix_to_graph_explore/

대체로 지금까지온 거리와 별개로 멀리 빵빵 돌아서 도착하는 경우가 많아서 그렇다고 생각한다.

특이한 점으로는 4층에서 A star를 euclidean의 제곱으로 하였는데, 이는 step보다는 heuristic에 더 많은 가중치를 주기 위해서 시도하였다.



1-5 각 층마다 코드 실행방법

Maze를 instance할 때 탐색하고 싶은 층의 문자열을 배열로 넣어준다.

```
maze = Maze(maze_names=["first", "second", "third", "fourth", "fifth"])
```

이 상태에서 maze.run()을 하면 first, second, third, fourth, fifth의 결과가 파일로 저장된다.

1-6 코드 동작 방법

- 1-3 에서 ex) fifth_floor(arg…) 이런식으로 층의 함수를 호출한다고 하였는데, 층마다 다른 정보 (좌표, 알고리즘) 등을 따로 설정하기 위함이었고, 구현으로는 이런 정보를 find_path()함수에 넣으면 최적 탐색 경로와 탐색 노드의 수를 반환한다.

```
self.find_path(here_y, here_x, key_y, key_x, 1, 1, 3)
```

find_path (시작 위치의 y, 시작 위치의 x, 목적지의 y, 목적지의 x, g 함수의 weight, f 함수의 weight, heuristic type)
heuristic type : 1. manhattan, 2. euclidean, 3. euclidean cube

g 함수의 weight 를 0으로 주는 것으로 greedy 탐색을 구현하였고,
greedy는 최적의 경로만을 반환하지 않고 돌아서 갈 수 있다.

1-7 length와 time

	length	time
first	3850	5809
second	758	1008
third	554	649
fourth	334	416
fifth	106	120

1-8 실험 결과

기존에 알고리즘 문제를 풀거나, 탐색을 할 때 bfs 나 dfs 를 사용하였는데, a*나 greedy가 bfs, dfs 보다 일반적으로 더 좋은 결과를 얻을 수 있다고 생각했다. 최악의 경우에 어떻게 될지 모르지만 그런 예시는 a* greedy bfs dfs 모두 다 안좋은 결과가 나오는 것 밖에 생각이 되지 않아서 확신 할 수는 없다.

그리고 예제는 가는 길들이 하나로 정해져있어서 무조건 최단 경로만으로 가게 되어서, 다른 예제를 돌리면 틀릴까봐서 6번째 미로를 만들어서 길을 찾아보았다.