기초 인공지능

Assignment01 – Maze Game

20191569 김서연

1. 사용한 라이브러리와 각 알고리즘마다 구현한 방법에 대한 간단한 설명

   사용한 라이브러리 : heapq

   Stage 1의 bfs는 visited와 path 리스트를 만든 후, 첫 시작점을 우선 path에 넣는다. 이후 반복문을 통해 path가 비어있지 않는 동안, path에서 첫 값을 빼오고, 그 값이 VISITED에 있는 점이라면 반복문의 첫 시작으로 돌아가고 없다면 추가를 해준다. 만약 그 점이 최종 골이라면 current 즉 경로를 출력한다. 골이 아닐때는, neighborPoints 함수를 통해 그 점의 주변 점들을 구해 visited에 없다면 path에 추가해주는 과정을 반복한다. Astar의 경우 heap를 사용했다. Openlist와 closedlist 리스트를 선언한다. 이때 openlist는 heap으로 사용한다. Heap에 첫 시작점부터 골까지의 맨하튼 거리 즉 h 값에 0인 g값을 더한 f와 시작 점을 넣는다. 이후 openList가 비어있지 않는 동안 반복문을 시작한다. Heap는 f 값이 작은 순서대로 정렬되어있다. F가 가장 작은 heap 노드 하나를 pop하여 그 노드의 마지막 값 즉 경로에 대한 정보를 path에 그리고 그 경로의 마지막 점 즉 현재 점을 currentpoint에 저장한다. Currentpoint가 closedlist에 있다면 반복문의 시작으로 돌아가고, end_point 즉 골과 같다면 path를 리턴한다. 그렇지 않다면 neighborPoints 함수를 통해 주변 점들을 구하고, 그 점들이 closedlist에 없을 때, path의 길이를 통해 g를 맨하튼 함수를 통해 h를 g와 h를 통해 f를 구하며 path에 선택한 주변 점을 더하여 heap에 push 한다.

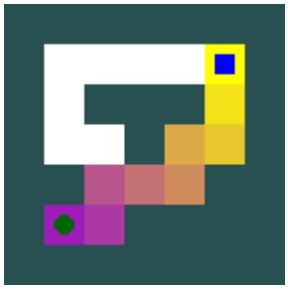
   Stage2의 경우 2가지의 astar를 구현했다. 하나는 stage1과 유사한 것이며 하나는 앞선 astar를 통해 각각 한 칸이 아닌 골과 골 사이에 관련된 astar이다. 앞으로 후자를 big astar라고 하겠다. 각 골들에 대하여 big astar를 각각 반복한다. 이때 big astar는 시작점과 각 골들 중 하나를 최종 목적지로 정한다. 이후 big astar함수에서, g는 startpoint에서 최종 골을 제외한 현재 위치의 골까지의 실 거리 즉 작은 astar로 구한 거리 값이며 h는 현재 골에서 최종 골까지의 맨하튼 거리이다. 즉 big astar는 각 골들을 한 칸 한 칸으로 보는 것이다. 따라서 모든 골들에 대해 big astar를 구하면, 각 최종골에 대한 최단 거리 길이가 나온다. 그 중에서도 최소 값이 모든 골들을 최단거리로 돌게되는 경로이다.

   Stage3의 경우 mst의 크루스칼 알고리즘에서 아이디어를 얻었다. 우선 stage1에서 구현

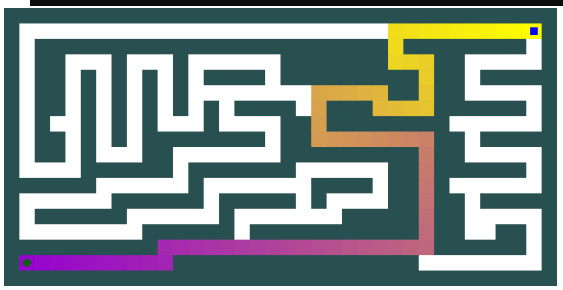
한 astar로 첫 시작에서 각 골들까지의 거리와 골과 골 사이의 모든 거리를 구하여 각각 start_edges와 myedges 리스트에 저장한다. 이후 이 값들을 갖고 mst함수로 들어가서, myedges를 작은 값부터 정렬하고, 작은 순서대로 pop하여 리스트에 저장한다. 이때 사이클을 형성하지 않는지, 한 골 당 경로가 두 경로 이상 연결되지 않는지 등을 검사한다. Edges를 pop하다보면 어느 순간 한 경로만 연결된 골이 2개가 남는데, 이때 시작점으로부터 각 골까지의 실 거리가 더 짧은 것을 구한다. 더 짧은 골이 시작점에서 바로 연결될 골이다. 이후는 pop하여 저장해뒀던 리스트를 검사하며 골들을 연결하여 경로를 구한다.

2. 각 stage별로 출력되는 경로 그림과, 터미널 상에서의 path length와 search states, execute time 출력 캡처 화면 ([그림 03] 참고)

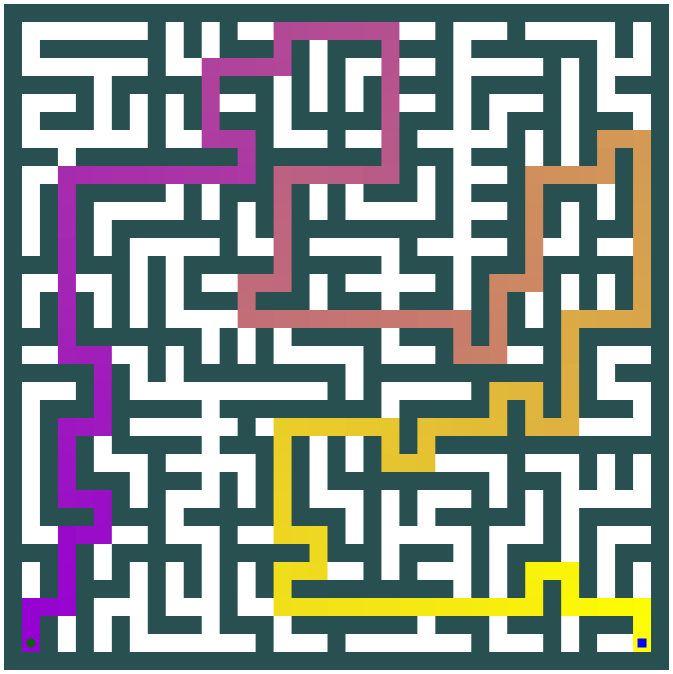
(1) bfs method로 실행한 stage1의 small.txt,medium.txt,big.txt 경로 그림과 출력 캡처 화면



```
==========================================
[ bfs results ]
(1) Path Length: 9
(2) Search States: 15
(3) Execute Time 0.0000000000 seconds
==========================================
```

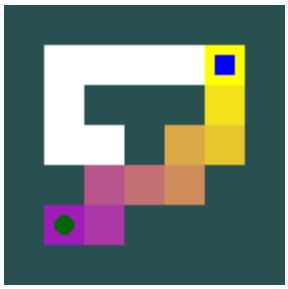


```
==========================================
[ bfs results ]
(1) Path Length: 69
(2) Search States: 269
(3) Execute Time 0.0039932728 seconds
==========================================
```

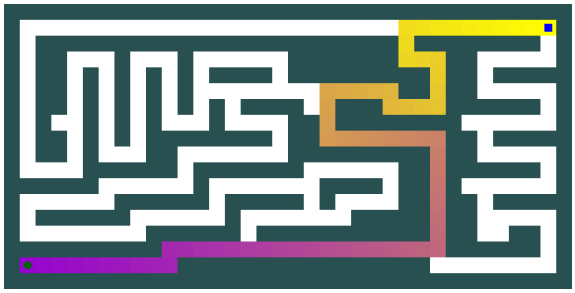


```
========================================
[ bfs results ]
(1) Path Length: 211
(2) Search States: 619
(3) Execute Time 0.0029904842 seconds
========================================
```

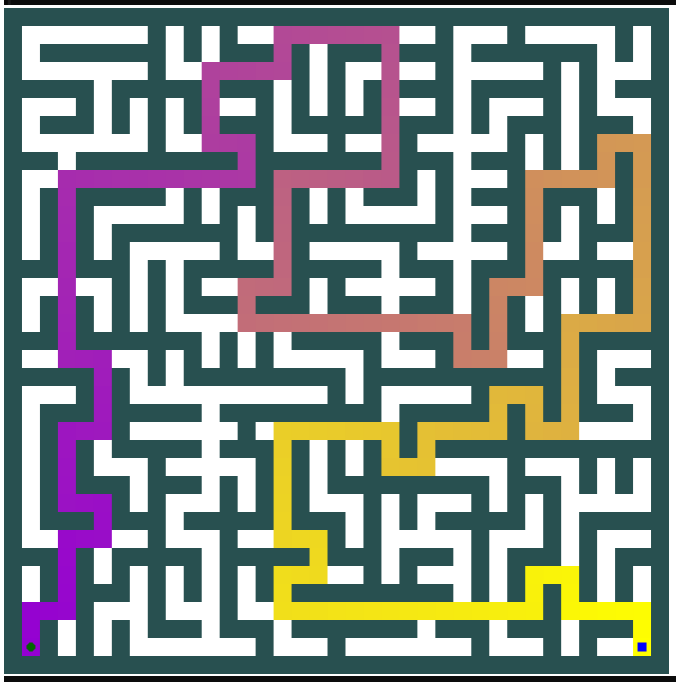
(2) astar method로 실행한 stage1의 small.txt,medium.txt,big.txt 경로 그림과 출력 캡처 화면



```
========================================
[ astar results ]
(1) Path Length: 9
(2) Search States: 14
(3) Execute Time 0.0000000000 seconds
========================================
```

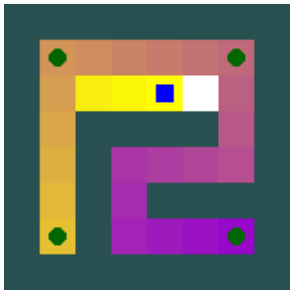


```
==========================================
[ astar results ]
(1) Path Length: 69
(2) Search States: 219
(3) Execute Time 0.0029911995 seconds
==========================================
```

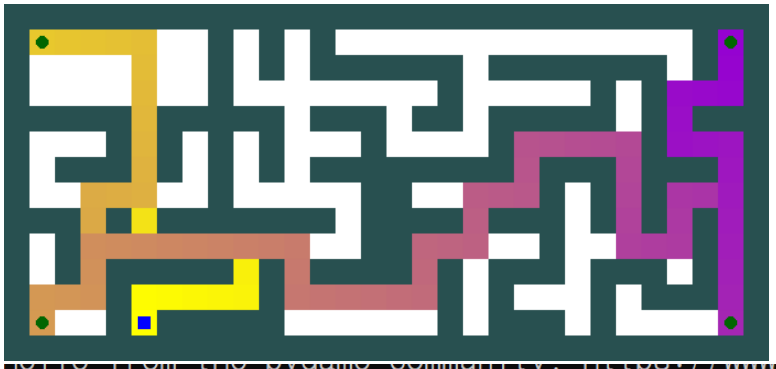


```
==========================================
[ astar results ]
(1) Path Length: 211
(2) Search States: 542
(3) Execute Time 0.0139727592 seconds
==========================================
```

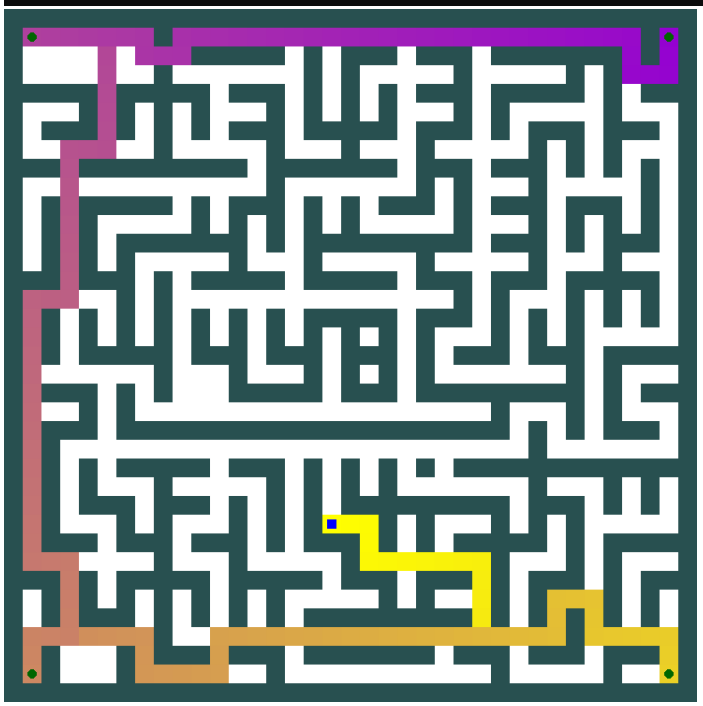
(3) astar_four_circles method로 실행한 stage2의 small.txt,medium.txt,big.txt 경로 그림과 출력 캡 처 화면



```
==========================================
[ astar_four_circles results ]
(1) Path Length: 29
(2) Search States: 1193
(3) Execute Time 0.0129642487 seconds
==========================================
```

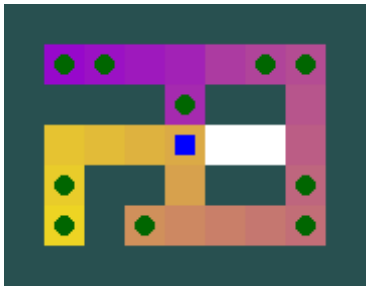


```
==========================================
[ astar_four_circles results ]
(1) Path Length: 107
(2) Search States: 7236
(3) Execute Time 0.1528766155 seconds
==========================================
```

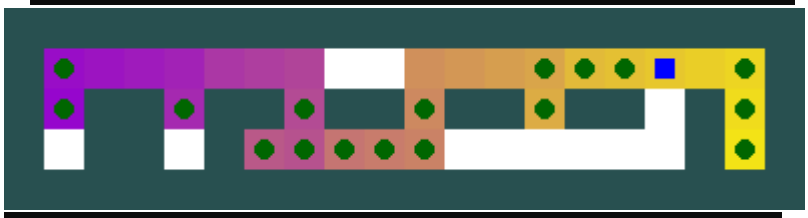


```
==========================================
 astar_four_circles results ]
1) Path Length: 163
2) Search States: 11785
3) Execute Time 0.2354040146 seconds
==========================================
```

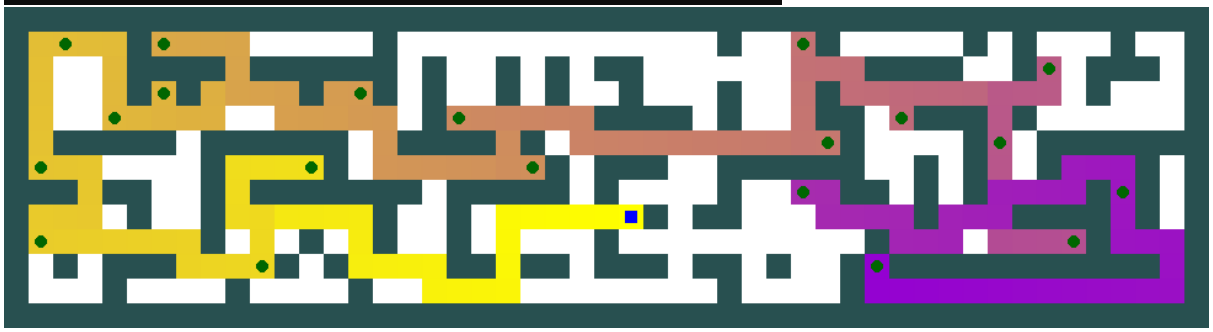
(4) astar_many_circles method로 실행한 stage3의 small.txt,medium.txt,big.txt 경로 그림과 출력 캡처 화면

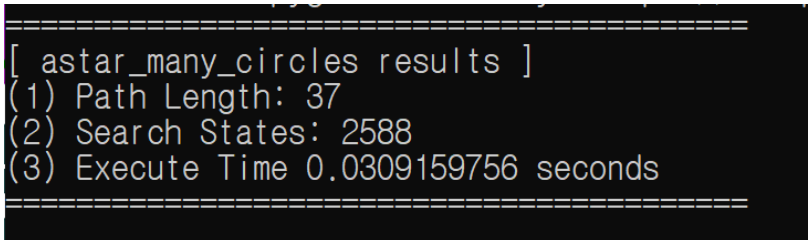


```
==========================================
[ astar_many_circles results ]
(1) Path Length: 30
(2) Search States: 672
(3) Execute Time 0.0099709034 seconds
==========================================
```



```
==========================================
[ astar_many_circles results ]
(1) Path Length: 225
(2) Search States: 33168
(3) Execute Time 0.5255904198 seconds
==========================================
```





4. Stage2와 Stage3에서 직접 정의한 Heuristic Function에 대한 설명

   Stage2의 경우 stage1에서 구현한 astar을 이용하여 골 사이의 거리에 대한 astar를 구현하였으나 h는 manhatten_dist를 사용했으며 stage3의 경우 mst중 크루스칼 알고리즘을 변형하였으나 마찬가지로 h는 manhatten_dist를 사용했다.