기초인공지능

Assignment1 보고서

20171646 박태윤

(Stage 1)

(1) BFS탐색

search.py의 Node클래스를 담는 큐( queue = deque() )를 사용하였다. 큐에서 빠진 노드의 location에서 갈 수 있는 방향에 따라 해당 노드의 loaction을 parent로 가질 수 있도록 노드를 생성한 뒤에 이 노드의 obj에 해당 노드의 좌표를 넣어주고 큐에 추가(append)하였다. 큐의 선입선출 특성에 의해 (출발점에서부터 1칸 떨어진 칸들), (출발점에서부터 2칸 떨어진 칸들), …… , (출발점에서부터 n칸 떨어진 칸들), ….. 와 같은 순서대로 탐색을 진행할 수 있기 때문에 너비 우선 탐색이 가능하였다. 이 때, 이동한 칸의 개수는 visit이라는 이차원 배열에 저장하였다. visit[x][y]과 같은 형태이며, x와 y은 각각 해당 지점에서의 maze좌표값을 의미한다. 출발 지점에서의 visit값은 1이다. 만일 탐색을 하던 도중 목표 지점에 도착하게 되면 탐색을 중단하고 path 가장 마지막으로 큐에서 빠진 노드의 obj를 copy해준다.

- small

텍스트이(가) 표시된 사진

자동 생성된 설명

- medium

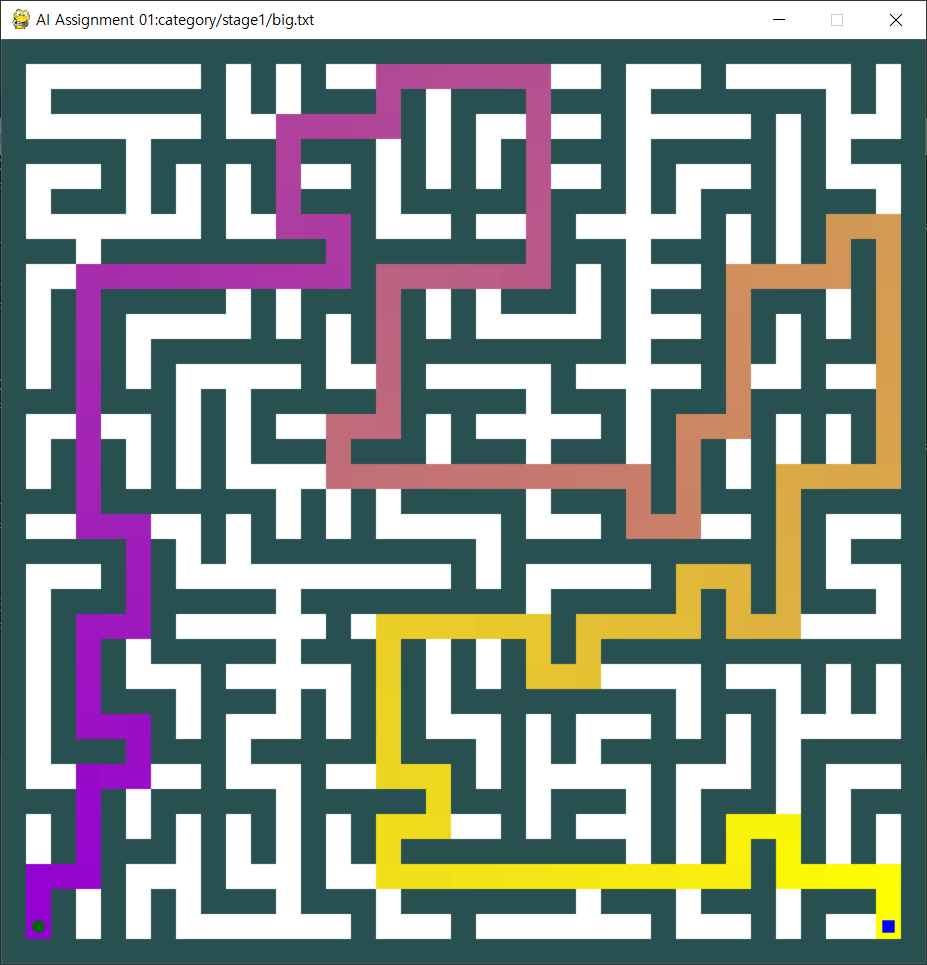
텍스트, 실외, 칠한, 밝은이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- big



텍스트이(가) 표시된 사진

자동 생성된 설명

(2) A\*

bfs와는 다르게 자료구조로 우선순위 큐를 사용하였다. 우선순위 큐는 특정 값을 기준으로 정렬이 되는 큐를 의미하는데, 이 기준이 되는 특정 값을 A\*알고리즘에서의 f값( f = g + h )로 설정하였다. 즉, 코드에서 큐에는 q.put((visit[x][y] + manhatten\_dist((x, y), end\_point), node))와 같이 값이 들어가는데, visit[x][y]는 현재 maze에서 (x, y)위치에 해당하는 visit값(출발점에서부터 실제 이동한 거리)이 g로, manhatten\_dist((x, y), end\_point)는 현재 (x, y)위치와 도착지점(end\_point)간의 맨해튼 거리값이 휴리스틱 값 h로 들어간다. 우선순위 큐는 visit[x][y] + manhatten\_dist((x, y), end\_point)값이 작은 순대로 자료를 뽑아낼 수 있다.

- small

텍스트이(가) 표시된 사진

자동 생성된 설명

- medium

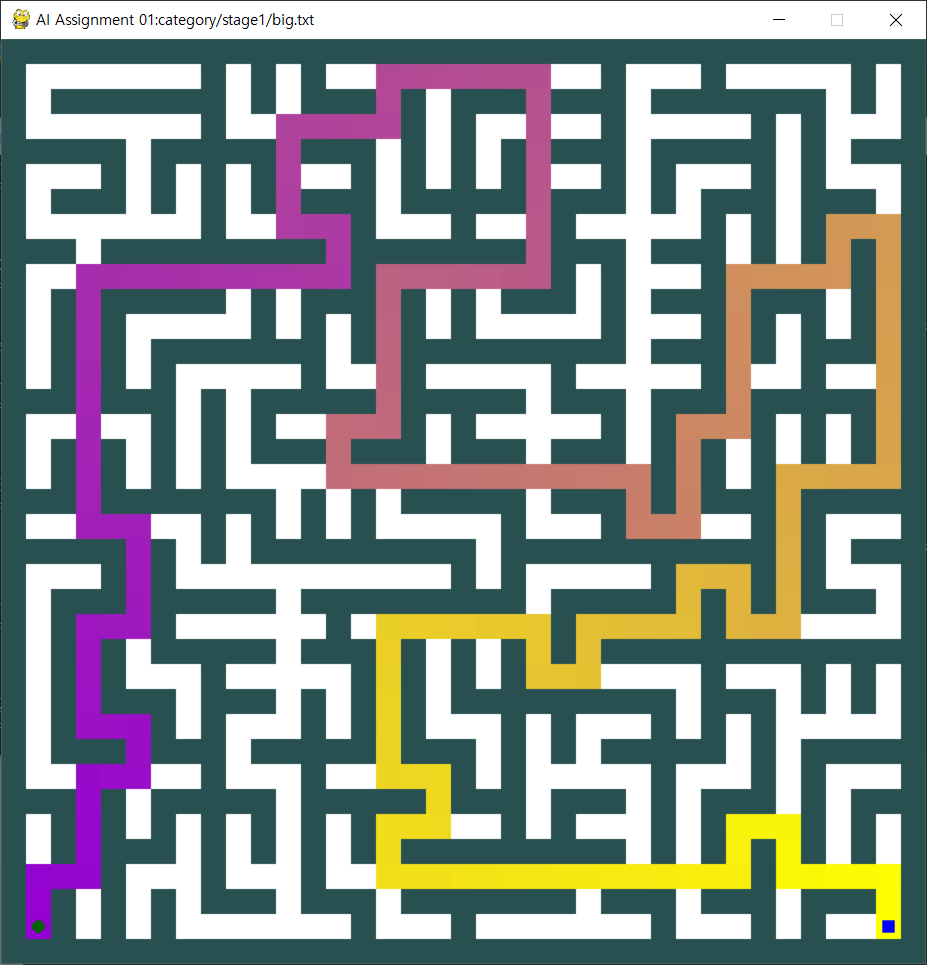
텍스트, 실외, 칠한, 밝은이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- big



텍스트이(가) 표시된 사진

자동 생성된 설명

(Stage 2)

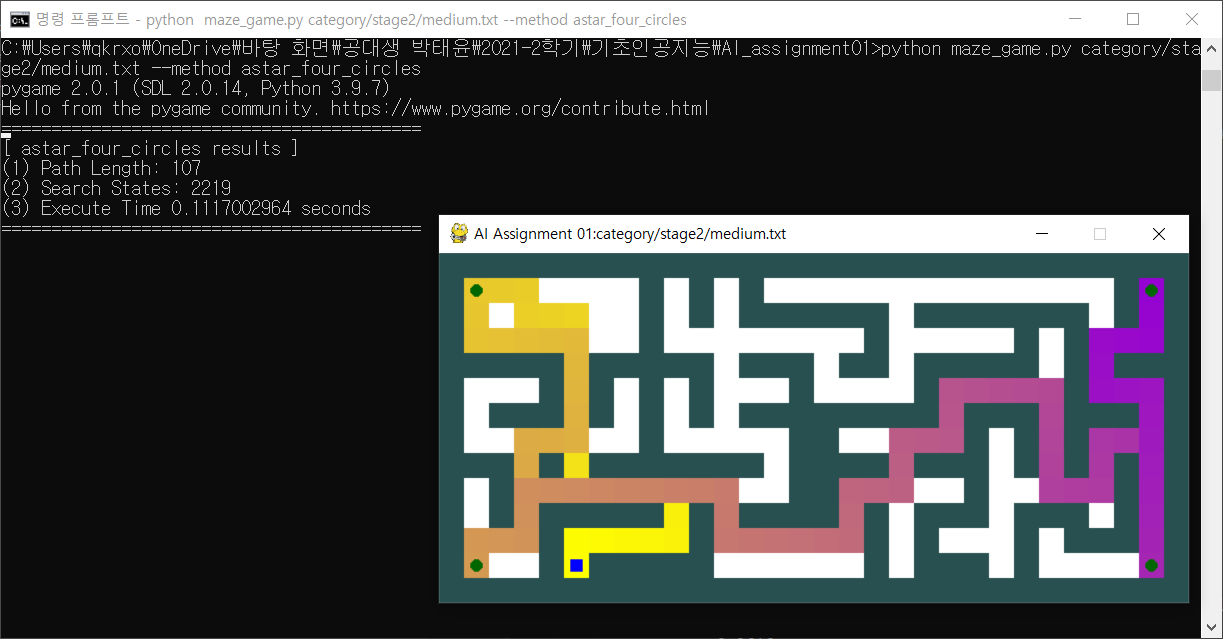
stage1에서 구현한 astar함수를 이용하여 astar\_distance(maze, start\_point, end\_point)라는 함수를 만들었다. 이는 astar알고리즘을 통해 미로(maze)내에서 start\_point에서부터 end\_point까지의 거리를 반환한다. 이 함수를 통해 dic\_obj라는 딕셔너리를 만든다. 이는 한 점에서부터 각 점들까지의 거리를 담고 있는 딕셔너리이다. 목적지 방문순서를 먼저 A\*알고리즘으로 정해주는데, 이 때 휴리스틱 값에 쓰이는 heu\_mat라는 이차원 행렬을 만들었다. 이는 한 점에서부터 각 점까지의 거리를 유클리드 거리로 구한 값을 가지고 있다. dic\_obj, heu\_mat 두 개의 자료구조를 가지고 목적지 방문 순서를 지정하는 A\*알고리즘을 돌린다. 역시 우선순위 큐를 이용하여 구현하였으며, 휴리스틱 값은 stage2\_heuristic에서 계산하여 res로 return하였다. stage2\_heuristic(mat, index)에서 mat는 5 X 5 이차원 행렬인데, 이미 앞에서 연결이 된 점이라면 mat에서 나타내는 거리의 값은 0이다. index는 앞으로 갈 점의 인덱스 번호를 나타낸다. mat을 res\_mat에 copy시킨다. 이후 res\_mat의 모든 값을 더해 res에 저장하고 res를 2로 나눈 뒤(1 -> 2, 2 -> 1과 같이 중복되는 값이 존재하기 때문) res = res \* (count – 1) / count 또는 res = res \* 0.5(count == 4)로 값을 바꾸고 res\_mat과 res를 리턴한다. count는 현재 연결이 안된 점들의 개수이다. res = res \* (count – 1) / count 또는 res = res \* 0.5는 나머지 노드들을 연결하는 간선들의 합들의 평균을 의미한다. 노드 n개를 연결하려면 최소 간선이 (n-1)개 필요하고 이를 현재 간선의 총 개수로 나누어 계산한 값이다. 예를 들어, 노드가 4개일 때 존재하는 간선의 개수는 6개이므로 (4-1)/6 = 0.5이므로 res = res \* 0.5(count == 4)인 것이다. 이를 통해 가장 최소의 비용을 가지는 목적지 방문 순서를 정했으면 이후는 stage1에서와 같이 astar알고리즘을 통해 한 점에서 다른 점으로 가는 최소 path를 구해준다. 이 때, 점이 4개이기 때문에 목적지를 한 번 방문하면 그 다음 경로에서는 이전에 방문한 목적지가 start\_point가 된다. 길을 찾는 과정에서 쓰이는 휴리스틱에서는 stage1에서는 맨해튼을 사용하였지만 stage2에서는 Euclidean distance를 사용하였다.

(small)

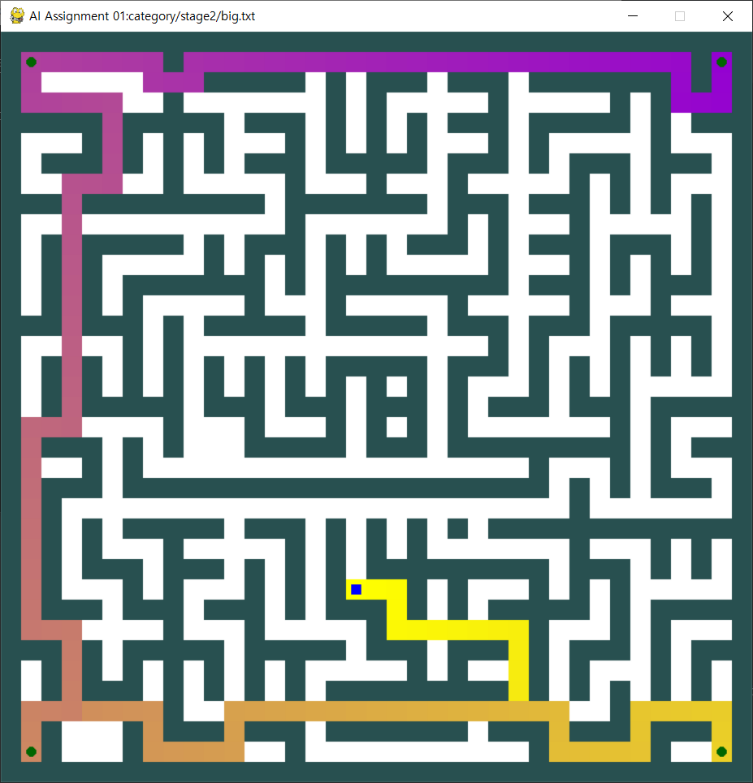
텍스트이(가) 표시된 사진

자동 생성된 설명

(medium)



(big)



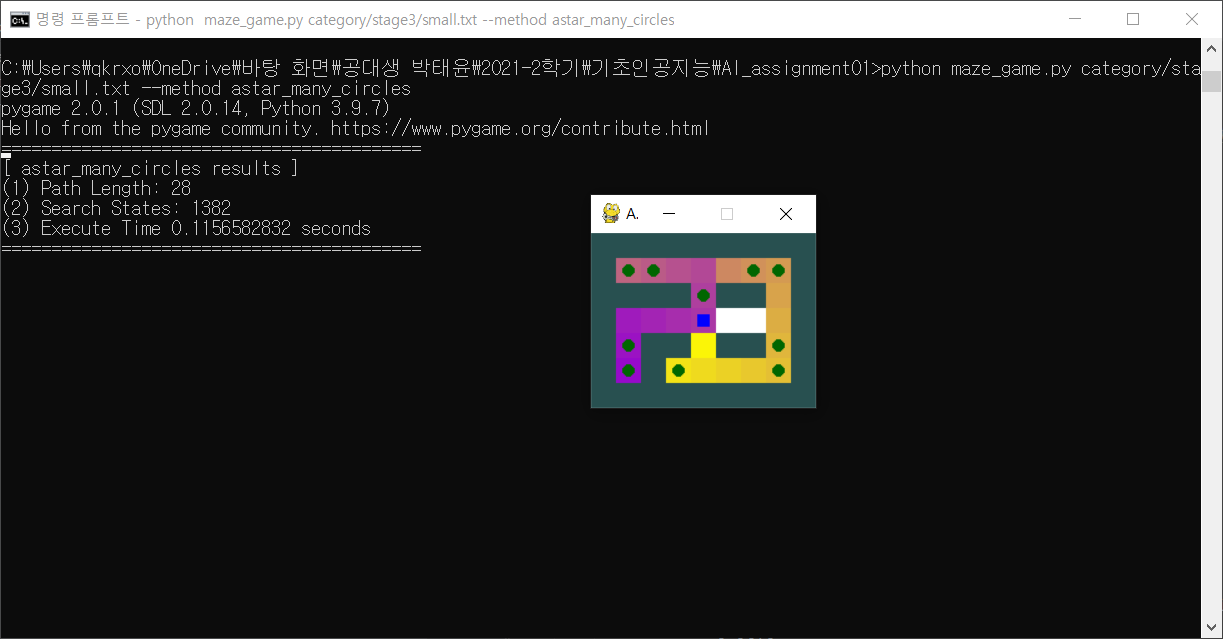
텍스트이(가) 표시된 사진

자동 생성된 설명

(Stage 3)

스테이지2에서와 마찬가지로 여러 목적지가 존재하기 때문에 목적지 방문 순서를 A\*로 먼저 정한 뒤에 path를 A\*로 만들어주었다. 목적지 방문 순서를 정할 때 휴리스틱은 목적지와 목적지 사이의 실제 거리를 간선 값으로 가지는 minimum spanning tree를 이용하였다. 각 점들간의 거리를 나타내는 이차원 행렬을 인자로 받아 이를 가지고 mst를 만들어 이차원 행렬로 리턴해주는 함수인 mst를 정의하였다. mst(len\_mat, isVisit)에서 len\_mat이 앞에서 언급한 각 점들간의 거리를 나타내주는 이차원 행렬이고, isVisit은 남은 점 개수만큼의 사이즈를 가지는 일차원 행렬로 해당 점을 방문했는지에 대한 여부를 나타낸다. mst함수는 stage3\_heuristic함수에서 호출이 된다. stage3\_heuristic(mat)에서 mat은 역시 각 점들간의 거리를 나타내는 이차원 행렬이다. stage3\_heuristic에서 mst함수를 통해 만든 최소신장트리 간선값들의 합을 리턴한다. 이 값을 휴리스틱 값으로 이용하였다. tmp\_node를 우선순위 큐인 obj\_queue에 추가하여 이를 통해 목적지 방문 순서를 정하는데, tmp\_node에 해당하는 인덱스를 부모 노드인 node의 점들간의 거리를 나타내는 node.road\_mat에서 행(index)과 열(index)을 잘라 tmp\_node.road\_mat으로 지정하였다. 또한 expand될 때마다 tmp\_node.obj에 현재 보고 있는 목적지의 좌표를 추가하였고, 최종적으로 큐에서 get한 node의 obj길이가 전체 목적지 점의 개수인 len(points)와 같아지는 순간 목적지 방문 순서를 정하는 반복문을 break하였다. 추가적으로 우선순위 큐에 자료를 넣을 때 num이라는 특정 값을 더해줬는데, 이는 자료들끼리 f값이 같은 경우 먼저 들어간 자료가 큐에서 get할 때 먼저 나올 수 있도록 더해주는 아주 작은 값이다.

(small)



(medium)

텍스트이(가) 표시된 사진

자동 생성된 설명

(big)

