AI\_assignmnet01 – Maze Game

BFS와 A\* Algorithm을 활용한 최단 경로 찾기

학번 : 20171669

이름 : 이재영

1. STAGE1
   1. BFS

* small.txt

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

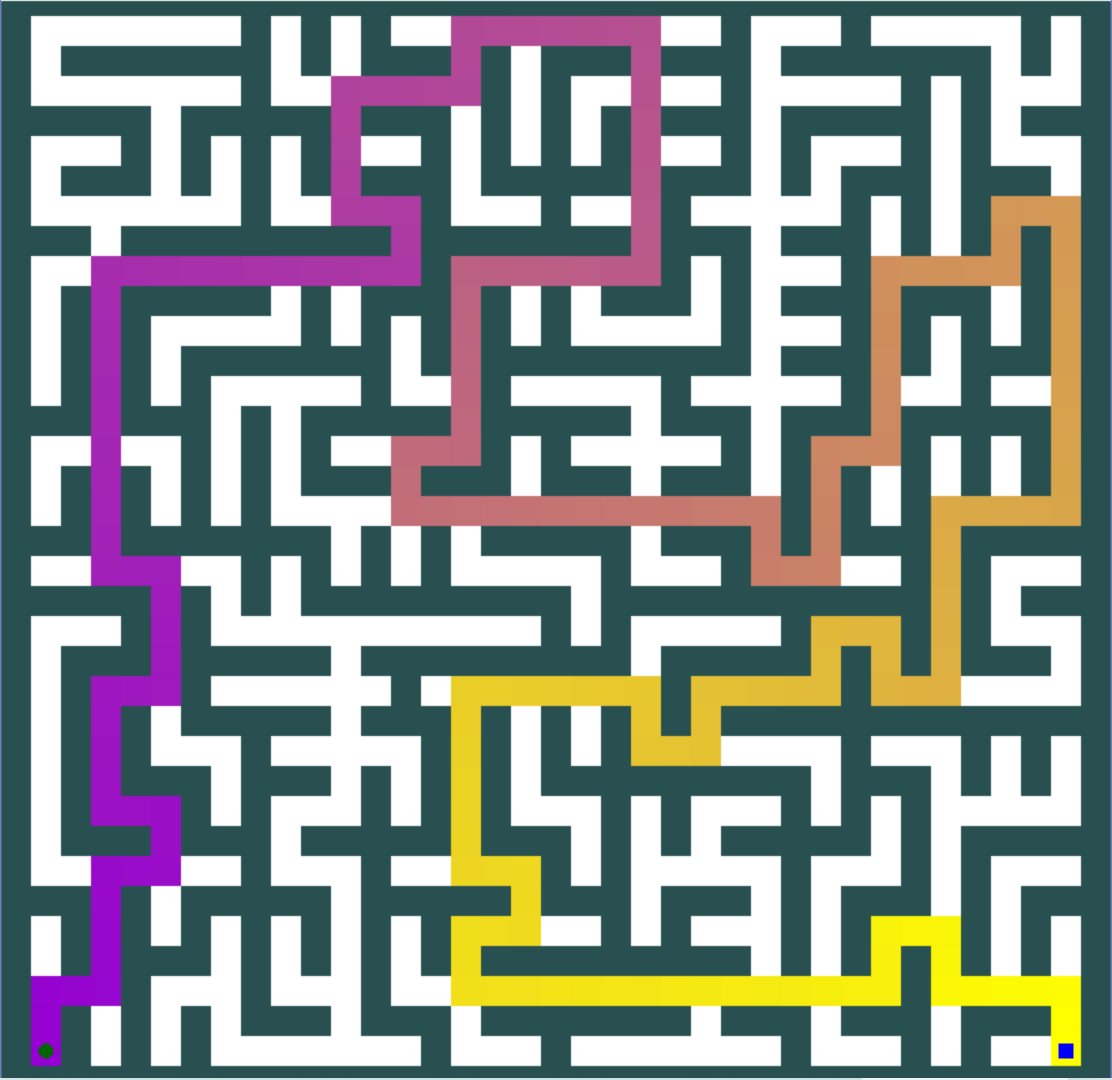
* medium.txt

텍스트, 실외, 녹색, 칠한이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

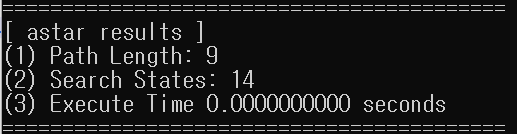
* big.txt

텍스트이(가) 표시된 사진

자동 생성된 설명

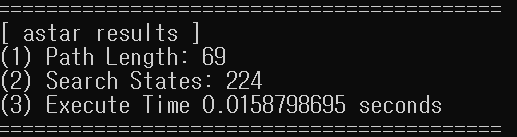
* 1. A-star algorithm
* small.txt

텍스트, 점수판이(가) 표시된 사진

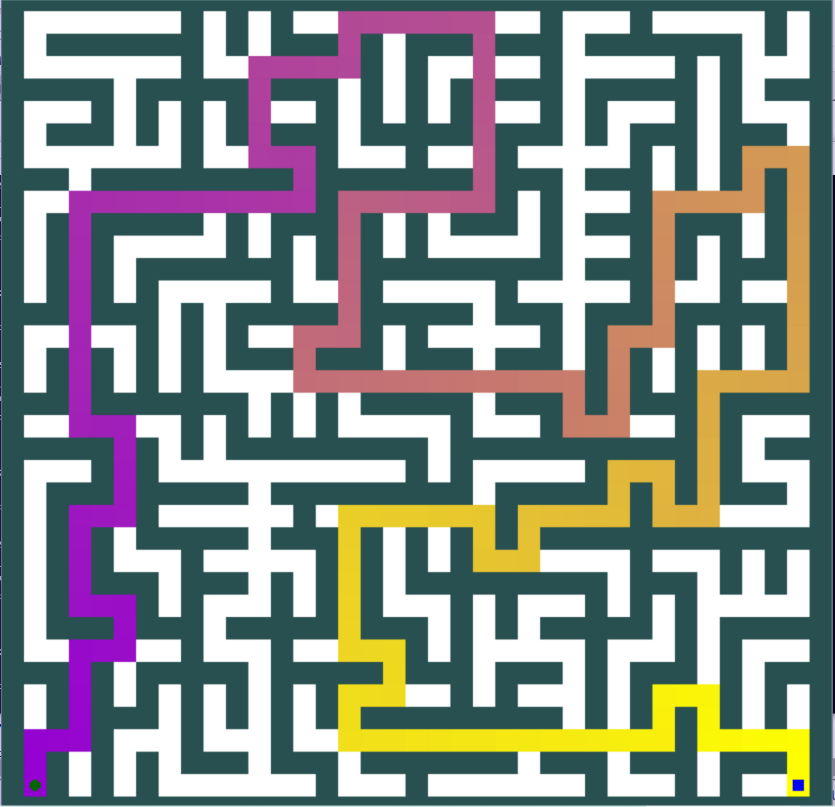
자동 생성된 설명

* medium.txt

텍스트, 실외, 밝은, 칠한이(가) 표시된 사진

자동 생성된 설명

* big.txt

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. Stage1 분석

BFS(Breath-First Search)는 너비 우선 탐색 방법으로 임의의 노드에서 시작해서 인접한 노드를 먼저 탐색하는 방법으로 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회방법으로 인접한 노드를 한번 검색하면 Search\_state가 1증가하는 방법으로 state 수를 확인할 수 있다.

* BFS 방법에서는 maze의 row와 col을 바탕으로 2차원 배열의 parent, dist 배열을 정의해서 미로의 시작지점인 start\_point로부터 시작했다. 시작지점으로부터 인접한 노드들은 maze.neighborPoints를 사용해 알 수 있다. 그렇게 dist 배열의 값이 1이상이면 방문한 것이므로 무시하고 아니라면 이전 지점의 dist의 값보다 1을 더하고 queue에 넣었다. 도착지점까지 도달했다면 현재지점을 저장하고 while 반복문을 빠져나왔다. 나와서는 현재 지점부터 출발 지점으로 역순으로 path에 저장한 후 path의 반대방향 (path[::-1])으로 return했다.

A-star algorithm은 출발지점으로부터 목표 노드까지의 최단 경로를 찾아내는 그래프 탐색 알고리즘으로 각 지점에 대해 그 지점을 통과하는 최상의 경로를 추정하는 순위 값의 heuristic 추정값을 이용한다. Stage1에서는 Heuristic 알고리즘으로 맨해튼 거리를 사용했다. 맨해튼 거리는 간단히 좌표에 표시된 두 점 사이의 거리의 차이로 볼 수 있다.

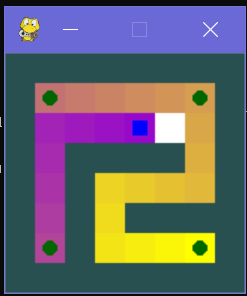
* Astar 알고리즘의 방법은 시작점을 startNode로 놓고 g,h,f 값을 초기화했다. 도착지점 또한 endNode로 놓고 g,h,f값을 초기화했다. openList 배열을 정의해서 startNode를 넣고 반복문을 시작했다. openList[0] 값을 currentNode에 넣고 f값이 가장 작은 것을 찾아내 pop을 한 후 closedList에 넣는다. 그 후에는 neighborPoints 로 인접 좌표들을 찾아낸 후 접근 가능한 노드들을 children에 넣고 g,h,f값을 넣고 반복문을 진행하고 endNode에 도착할 때까지 반복문을 진행한다.

Stage1의 실험을 통해 BFS의 방법을 사용할 때보다 A-star algorithm을 사용했을 때 공통적으로 Path는 최단 거리를 찾지만 Search State, 즉 주변 인접 노드를 더 적게 search 한다는 것을 확인할 수 있다.

1. Stage2

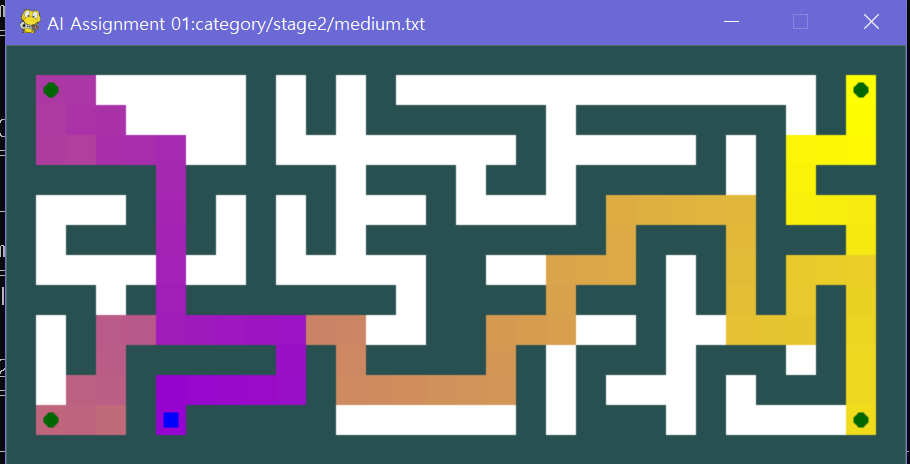
2-1. 사진

* small.txt

텍스트이(가) 표시된 사진

자동 생성된 설명

* medium.txt



텍스트이(가) 표시된 사진

자동 생성된 설명

* big.txt



텍스트이(가) 표시된 사진

자동 생성된 설명

2-2. Stage2 분석

Stage2의 경우는 4개의 도착지점을 거치는 최단거리를 구하는 방법으로 astar algorithm 을 사용한다. 4개의 지점을 어떤 순서로 갈지를 모두 정해서 최단 거리를 구한다. 4개의 지점의 순서는 4!, 즉 24가지의 경우가 있기 때문에 24가지의 순서 중에서 최단 거리를 구하는 알고리즘을 구현했다. 이를 위해서 from itertools import permutations 를 사용해서 0, 1, 2, 3의 순열을 list로 저장했다. 또한 도착지점이 4개이므로 각각이 출발점과 도착점이 되도록 astar 알고리즘을 4번 진행해서 path를 구현했다. path가 최소일 때, search\_state, path를 저장해놓고 최종 최단거리를 path에 넣는다. Search\_State의 경우에는 24가지 경우의 수 동안 탐색하기 때문에 높게 나온 편이다.

Stage2의 경우는 Heuristic 함수를 2초 이내에 사용하기 위해 유클리디안 제곱 Heuristic 함수를 사용했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Astar 알고리즘의 경우는 Stage1의 방법과 같으나 따로 추가한 점은 도착지점이 각각 4가지 지점 순서대로이다. 즉, 도착지점으로 도착했으면 그 지점이 출발지점이 되고 다른 지점에 도착지점으로 astar 알고리즘을 진행했다.

1. Stage3

3-1. Stage3 분석

Minimum spanning tree를 사용하기 위해 bfs\_edge\_check라는 함수를 선언해 실제 지점사이의 거리를 탐색해서 가중치로 삼았다. 그 값들을 각각 출발 지점, 도착 지점, 가중치 순으로 edge\_graph에 넣어서 저장한 후 mst 함수에 넣어서 Kruskal algorithm을 구현해서 도착지점들의 순서를 고르는 단계이다.