기초 인공지능 첫번째 과제

학번 20191583

이름 : 김태곤

※사용한 라이브러리와 알고리즘 별 구현 방법 및 실행 결과

1. Stage1

(1). BFS method

BFS는 시작 노드에서 인접한 노드를 먼저 탐색해 나가는 방법이다. 방문한 노드를 저장 후 다시 꺼내는 FIFO작업이 필요한데, 이는 Queue를 이용하여 구현하였다. Library는 deque를 사용했다. 먼저 미로 크기의 matrix를 0으로 초기화하고 시작점에 1을, Queue에 시작점을 넣었다. 이후 Queue를 모두 꺼낼 때까지 현 위치에서 neighborPoints를 모두 집어넣고 다음 위치로 이동하는 과정을 반복하였다. 목적지에 도착하면 Queue가 남아있어도 종료 시켰다.

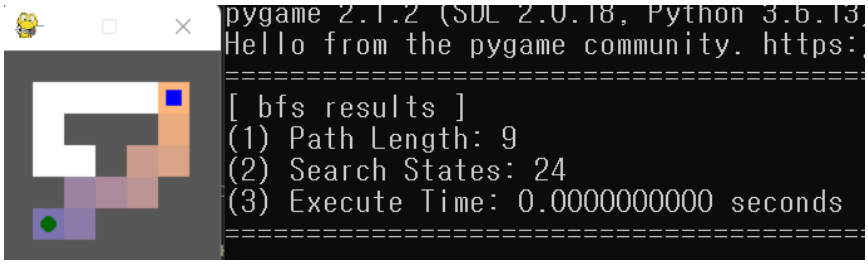

그림 1 BFS method를 사용하여 small.txt 실행

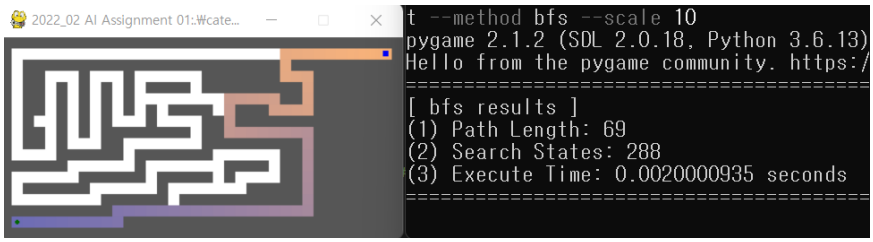

그림 2 BFS method를 사용하여 medium.txt 실행

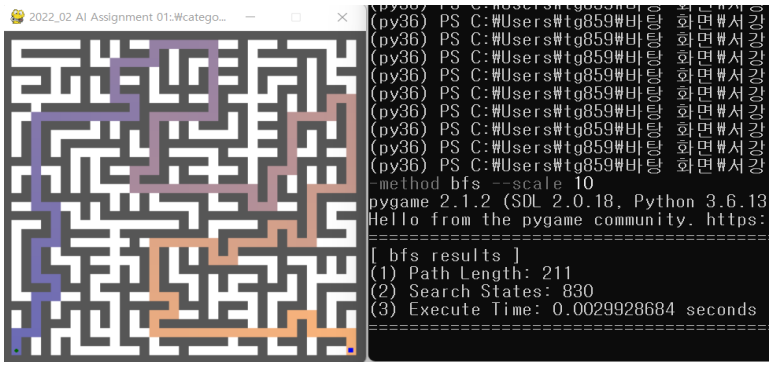

그림 3 BFS method를 사용하여 big.txt 실행

(2). DFS method

DFS는 그래프의 깊은 부분을 우선적으로 탐색하는 알고리즘이다. 방문한 노드를 저장 후 다시 꺼내는 LIFO작업이 필요한데, 이는 Stack을 이용하여 구현하였다. 먼저 BFS와 동일하게 matrix를 구성하고, stack에 시작점을 넣었다. 이후 stack을 모두 꺼낼 동안 LIFO를 통해 주변 노드를 추가해 나갔다. 목적지에 도착하면 종료시켰다. DFS는 깊이 우선으로 탐색하므로 운이 좋으면 BFS보다 빠르게 경로를 찾을 수 있지만, 최단 경로가 아닐 수 있다. 그림2와 그림5를 보면 Search State는 DFS가 더 좋지만 최단 경로는 아니다.

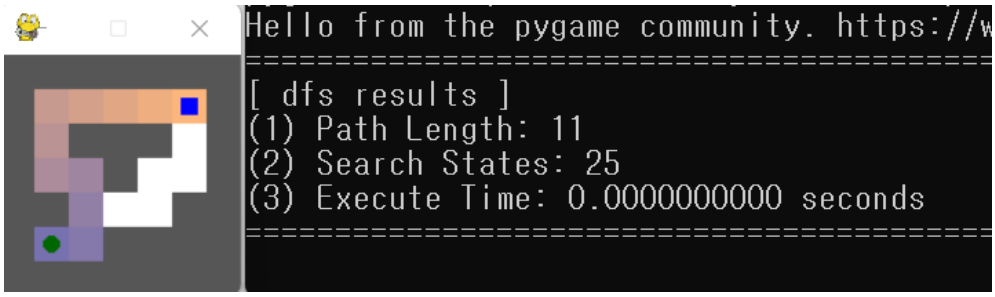

그림 4 DFS method를 사용하여 small.txt 실행

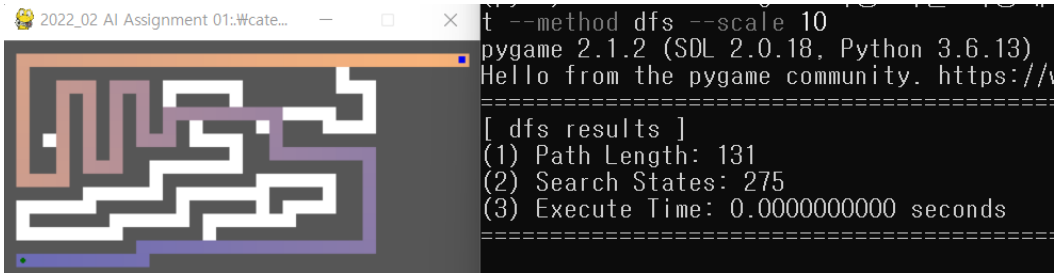

그림 5 DFS method를 사용하여 medium.txt 실행

그림 6 DFS method를 사용하여 big.txt 실행

(3) A*

A*알고리즘은 목적지 노드까지 최단경로를 찾는 알고리즘이다. F=G+H로 다음 경로는 그 경로까지의 coast + 휴리스틱 추정값이다. 1번 문제에서 휴리스틱 함수로는 Manhattan distance 방식을 사용하였다. 먼저 open List에 시작점의 위치, f,g,h값, parent 기본값을 넣어준다. 이후 open List가 Null이 아닌 동안 open List를 pop한다. 만약 목표 노드에 도착하면 반복문을 종료한다. 현 위치에서 이동할 수 있는 다음 노드를 neighborPoints를 통해 찾고, f,g,h값을 넣어 기존에 close List에 있는지, open List에 더 큰 g값이 있는지 확인하고 open List에 넣는다. 위 과정을 통해 시작점부터 목표지점 까지 최단경로를 구할 수 있다.

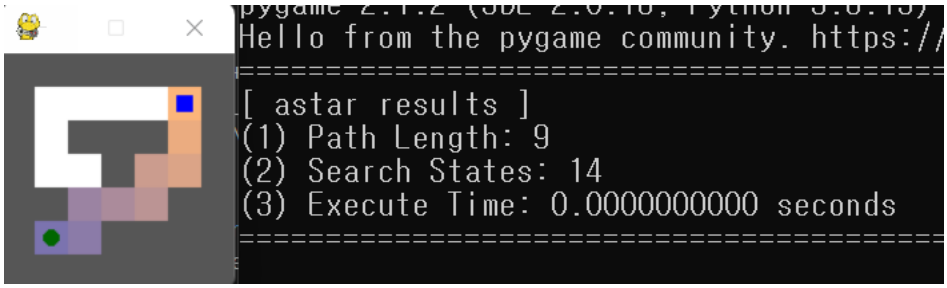

그림 7 A* method를 사용하여 small.txt 실행

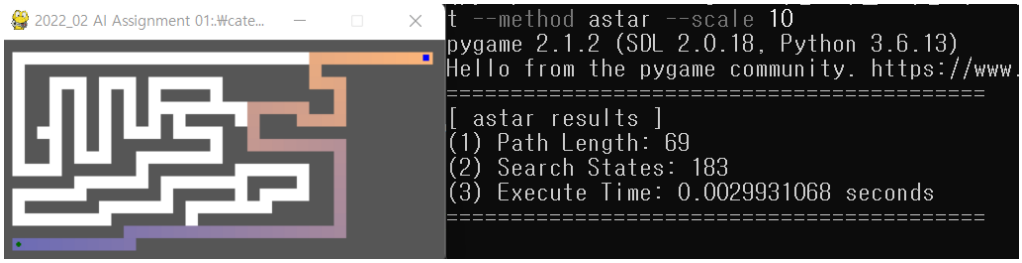

그림 8 A* method를 사용하여 medium.txt 실행

그림 9 A* method를 사용하여 big.txt 실행

2. Stage2

(1) A* four circles

Stage2 는 기존 Stage1 에서 사용한 A*알고리즘을 응용하여 작성하였다. 휴리스틱 함수는 Euclidean distance 를 이용하였다. 정확한 계산은 2*sqrt(dx^2+dy^2)이지만 값에 영향을 주는 dx^2+dy^2 만 사용하였다.

A* 알고리즘은 시작점과 목적지가 있을 때 최단경로를 찾는 알고리즘이다. 하지만 Stage2 는 목적지가 달라질 수 있으며 4 개의 점을 모두 지나가야 한다. 이를 해결하기 위해 Euclidean distance 를 사용한 A*알고리즘으로 모든 점 사이의 거리를 구한 후 점들을 연결해 주었다. 점이 4 개 뿐이므로 연결된 경로는 $\sum_{k=1}^{n} k$ = n(n+1)/2 = 10 개가 생성되고, 이를 거쳐가는 경우의 수는 n! = 24 가지가 나온다. 비효율적이지만 조합을 찾아 최단 경로를 path 에 추가하였다.

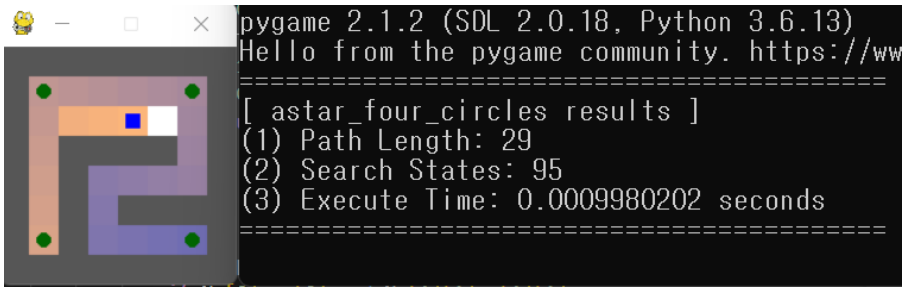

그림 10 astar_four_circles method를 사용하여 small.txt 실행

그림 11 astar_four_circles method를 사용하여 medium.txt 실행

그림 12 astar_four_circles method를 사용하여 big.txt 실행

3. Stage3

(1) Stage3는 n개의 점을 모두 지나는 최단경로를 구하는 문제이다. 이를 해결하기 위해 먼저 n개 점을 모두 연결하는 선들을 Euclidean distance를 사용한 A*로 구하였다. 이 방법은 기존 stage2에서 모든 연결선을 구할 때 사용한 방식과 동일하다. 이 문제의 핵심은 다음 목표 지점을 효율적으로 선택하는 것이다. Stage3에서 구현한 휴리스틱 함수는 다음 점으로의 목표지점을 구할 때 사용하였다.

F = G + H에서 휴리스틱 함수를 n개의 점을 최단으로 연결하는 prim algorithm을 활용하여 기존 유클리드 값에 남은 노드의 MST 값을 더한 점을 목표점으로 잡았다. MST의 prim algorithm은 출발지 정점을 선택하여 빈 노드에 추가한 후, 포함이 안된 노드의 간선 중 최소인 것을 계속 연결해 주었다. 다음 목표를 찾는 휴리스틱 함수는 n개의 dots를 찾는 A*에서 open List에 새로운 이웃 노드를 추가하기 전 작업을 해주었다.

조금은 비효율 적이지만 점들 사이의 최단 경로를 구한 후, 다시 처음 시작점에서 출발하여 위에서 정한 휴리스틱을 이용해 다음 목표를 정했다. 이렇게 함으로써 그림 13처럼 오른쪽으로 먼저 출발하는 최단 경로를 잘 찾는 것을 확인할 수 있다. 그림 16 big.txt의 결과를 보면 실행시간이 60초를 넘지 않는 것을 확인할 수 있다.

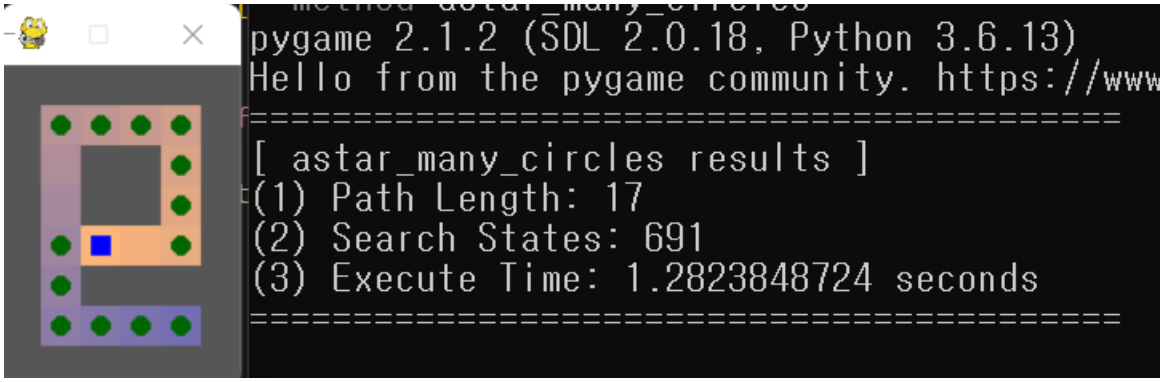

그림 13 astar_many_circles method를 사용하여 tiny.txt 실행

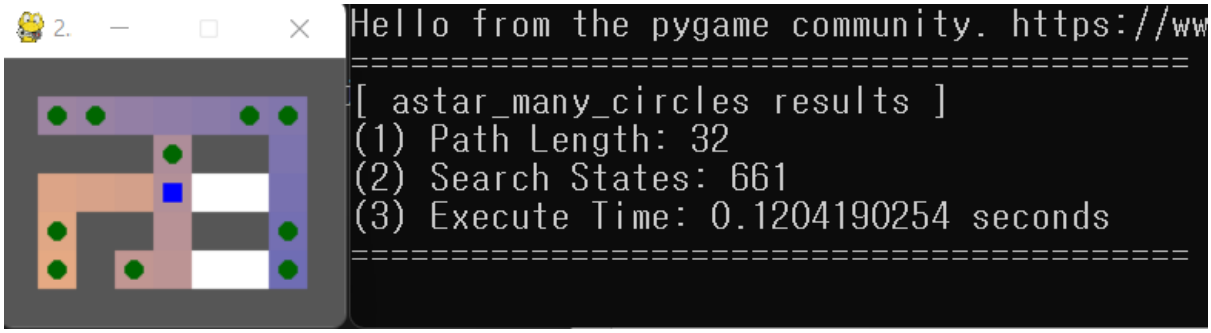

그림 14 astar_many_circles method를 사용하여 small.txt 실행

그림 15 astar_many_circles method를 사용하여 medium.txt 실행

그림 16 astar_many_circles method를 사용하여 big.txt 실행