[COSE-361] 인공지능: Assignment #3

2016130927 박준영

A. pacman.py (test71.lay) 결과

```
(py36) D:₩Study₩22_1st Semester₩인공지능₩Assignment₩minicontest1₩minicontest1>python pacman.py --agent MyAgent --layout
test71.lay
Pacman emerges victorious! Score: 738
Average Score: 738.936484146117
Scores:        738.936484146117
Win Rate:      1/1 (1.00)
Record:        Win
```

B. 에이전트 로직 설명

*** 코드 주석: '박준영_2016130927.py' 참조**

주어진 문제 상황에서 팩맨 에이전트는 (1). **음식이 '존재하고'** (2). 존재하는 음식 가운데 **가장 가까운 곳으로 이동**하는 경로를 모든 음식을 미로 상에서 모두 먹을 때까지 할당받게 설계하였다. 주요 로직은 다음과 같다.

(1). **findPathToMoreDot** 함수를 통해 주어진 팩맨 에이전트의 입장에서 "가장 많은 음식"을 먹는 경로를 찾는다.

(2). 현재 위치 및 후속 정보(successor)를 통해 다음 위치, 이동 방향, 이동비용뿐만 아니라 **'다음 위치에 먹이가 있는지'** 여부를 확인할 수 있다.

(3). 해당 팩맨 에이전트가 이 장소를 방문하지 않았고, 다른 에이전트 또한 이 먹이를 먹지 않았다면 이 위치에 **높은 우선순위를 준다(음의 정수로 표현)**.

이때 인덱스 값이 다른 에이전트의 접근 여부를 파악하기 위해 eaten이라는 변수를 **전역**으로 선언, visited라는 해당 팩맨의 방문 여부를 확인하는 **지역** 변수와 함께 관리했다. 즉 다른 팩맨이 '아직 먹지 않은' 음식 정보에만 우선순위 정보를 할당했다.

음의 정수로 우선순위를 주었기 때문에 "먹이가 있는 곳"과 "없는 곳"은 뚜렷하게 분리할 수 있다. 하지만 "먹이가 있는 곳" 중에서도 특히 현재 팩맨의 입장에서 가까운 곳으로 향할 때 보다 "빠르게" 문제를 풀 수 있으리라 생각했다. 따라서 맨해튼 거리의 역(1/맨해튼)을 추가로 뺄셈함으로써 먹이 정보에 보정치를 주었다. 즉 **주어진 먹이 중 맨해튼 거리가 가장 가까운 곳이 우선순위가 가장 높다**. 이때 맨해튼 거리의 역을 사용한 까닭은 역을 사용하지 않을 때 음의 정수 리터럴보다 값이 커지는 경우를 경계했기 때문이다.

findPathToMoreDot 함수는 특정 경로 내 먹이가 얼마나(먹이 여부 누적 카운트), 어느 정도 가까이(맨해튼 거리 이용) 존재하는지 찾아내고, 이 값을 이동 경로 비용에 더한 정보를 우선순위로 사용한다.

C.1. 구현한 에이전트가 기존 에이전트(baseline)보다 성능이 좋은 경우

해당 문제에서 findPathToClosestDot 함수를 통해 얻어낸 경로보다 점수가 높았다. 우선순위

기준이 이동 경로의 최솟값보다도 '가장 많은 먹이'인 까닭에 **팩맨 에이전트의 이동 비용보다 빠른 먹이 습득에 점수 가중치를 더 많이 주는 경우** 성능이 뛰어나리라 생각된다. 또한 주어진 문제의 **먹이가 매우 많고**, **처음부터 팩맨이 먹이에 가깝게 접근하는 위치에 분포**해 있을 때 해당 정보에 더 빨리 우선순위를 줄 수 있다.

C.2. 구현한 에이전트가 기존 에이전트(baseline)보다 성능이 좋지 않은 경우

팩맨의 **초기 위치 근처에 먹이가 존재하지 않는다면** 기존 에이전트(baseline)의 우선순위인 이동 경로의 최솟값을 그대로 활용하기 때문에 이동 경로는 동일하다. 하지만 먹이 여부 파악, 전역 eaten을 통해 **추가 연산이 필요**하다는 점에서 시간 복잡도 O(N)(N은 미로의 전체 크기, food 변수를 통해 참/거짓을 확인하기 때문)이 추가되고, 성능이 좋지 않으리라 생각된다. 또한 본질적으로 **팩맨의 이동 비용을 감소시키는 게 보다 더 나은 성능을 보이는** 미로 구조라면, 이 '먹이 우선' 모델은 좋지 않은 성능을 보일 것이다.

C.3. 모델링 시 추가 고려할 점

(1). **불필요한 이동 금지**: 같은 먹이를 향해 두 개 이상의 팩맨이 이동할 필요가 없다. 팩맨의 이동 자체가 전체 점수를 깎는 요인이기 때문에 가능하다면 주어진 해당 먹이에서 가장 가까운 팩맨만을 그 방향으로 이동하게 한다. 가령 먹이가 1개 남아 있고 팩맨이 두 개 있을 때 팩맨 A가 두 칸, B가 한 칸 거리라면 B만 이동하고 A는 이동하지 않고 정지하는 게 가장 효율적인 경로다.

(2). **먹이의 위치 사전 파악**: 먹이의 분포를 사전에 알 수 있기 때문에 어느 곳에 집중적으로 분포되어 있는지 알 수 있다. 먹이가 많은 방향으로 팩맨을 많이, 적은 방향으로 적게 보낼 수 있을 것이다.

(3). **팩맨 별 먹이 할당**: 먹이의 개수를 사전에 알 수 있기 때문에 총 먹이의 개수를 균등히 나눌 수 있다. 한 타임 스탬프에 최대한 가장 많은 먹이를 팩맨이 각각 먹고, 동시에 가장 빠르게 문제를 풀 수 있을 것이다. 이때 각 팩맨이 먹을 먹이는 물론 서로 다른 먹이여야 한다.

(4). **미로의 벽 구조**: 맨해튼 거리가 같다 할지라도 벽의 유무에 따라 먹이를 먹을 수 있는지가 결정된다. 벽 정보를 사전에 알 수 있기 때문에 가중치를 통해 조절할 수 있다.

위와 같은 사전 정보 활용 연산이 오히려 총 점수를 깎지 않도록 주의.