2017170307 정승원

1. capture the result of pacman.py with layout test71.lay

```
(base) C:₩Users₩redli₩OneDrive₩바탕 화면₩고려대학교₩2021-1학기₩인공지능₩minicontest1>python pacman.py --agent MyAgent --
layout test71.lay
Pacman emerges victorious! Score: 795
Average Score: 795.2000000000019
Scores:        795.2000000000019
Win Rate:      1/1 (1.00)
Record:        Win
```

2. Description of your agents.

ClosestDotAgent를 참고하여 MyAgent를 작성하였다. ClosestDotAgent는 매 순간 가장 가까운 목적지(Food)로 가는 Agent이다. 이 Agent는 목적지가 같은 두 개 이상의 pacman agent가 있는 경우에 대해 효율적이지 않았다. pacman agent A와 B가 같은 Food를 향해 가고 있으며, A가 B보다 Food에 먼저 도착한다고 가정해보자. A가 Food를 먹기 전까지 B는 그 Food로 계속 향할 것이다. A가 Food를 먹고 나서야 B는 다른 Food를 먹기 위해 다시 계획을 세울 것이다. 이런 단점을 보완하여 MyAgent를 작성하였다.

MyAgent에서는 각 agent가 어떠한 목적지(Food)로 향하는지를 저장한 GoingList, 목적지까지 가기 위해 해야 할 action들을 저장한 actionList, actionList의 길이를 저장한 actionLength를 이용하여 탐색을 진행하였다. agent의 actionList가 비어있을 경우 경로 탐색을 시작한다. 목적지를 찾기 전까지의 탐색 방식은 BFS와 유사하다. 목적지를 찾은 다음 GoingList에 현재 찾은 목적지와 같은 목적지가 있는지 없는지 검사한다. 만약 같은 목적지가 없다면 GoingList[self.index]에 해당 목적지의 위치를, actionList[self.index]에는 결과로 나온 action들의 집합을, actionLength[self.index]에는 actionList의 길이를 저장한다. 만약 같은 목적지(G)가 GoingList에 이미 있다면 agent(A)가 찾은 경로가 현재 G로 가고 있는 다른 agent(B)의 경로보다 짧은지 확인해야 한다. 만약 더 짧다면 B의 GoingList를 (-1,-1)로, actionList를 []로, actionLength를 0으로 초기화하고 A의 GoingList에 목적지를, 목적지까지의 경로를 actionList에, 그 경로의 길이를 actionLength에 저장해준다. 만약 A의 경로가 B의 경로보다 더 길다면, 현재 방문한 목적지(Food)의 위치를 visitedPosition에 추가하고 continue를 사용해 탐색 재개하여 다음 목적지(Food)를 찾는다.

위의 알고리즘을 사용하여 게임을 해보니, Food가 몇 개 안 남았을 때 점수가 빠르게 떨어지는 것을 확인할 수 있었다. 목적지가 G 하나 남았고, agent(A)가 G로 향하는 경로를 찾았지만, 이미 G로 가고 있던 agent(B)의 남은 경로보다 actionLength가 긴 경우를 생각해보자. 이때 A는 다른 목적지(Food)를 찾을 것이다. 하지만 다른 목적지는 없다. 따라서 A는 map 모든 곳을 확인할 것이고, 결과적으로 탐색시간이 늘어나게 되어 점수가 빠르게 떨어지게 된다.

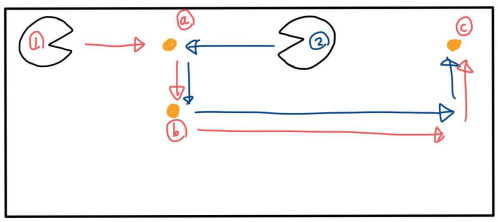
이를 방지하기 위해 음식(Food)이 agent의 개수(agentNum)의 절반(홀수의 경우 절반+1)정도 남았을 때, 탐색을 통해 찾은 경로는 별도의 확인과정을 거치지 않고 바로 그 agent의 경로로 인정해주는 코드를 넣었다(line 87~94). 코드를 넣고 게임을 실행해보니 음식이 몇 개 남지 않았을 때 점수가 이전보다 적게 떨어지는 것을 볼 수 있었다.

각 agent의 actionList를 구한 다음에는 agent가 다음에 할 action들을 return 해주어야 한

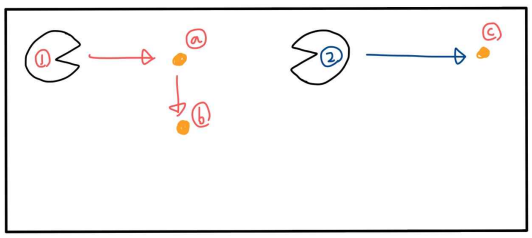
다. 만약 actionList의 길이가 0보다 작거나 같다면(목적지를 찾지 못한 경우) agent를 STOP 시키고, actionList의 길이가 0보다 크다면 첫 번째 원소를 꺼내 제거하고 return한다. 이때 원소를 제거함으로써 actionList가 비어있게 된다면, agent의 GoingList를 (-1,-1)로 초기화 한다.

3. Three discussions when playing Pacman.

• Discuss cases where the agent implemented by yourself is better than the baseline.



<ClosestDotAgent>



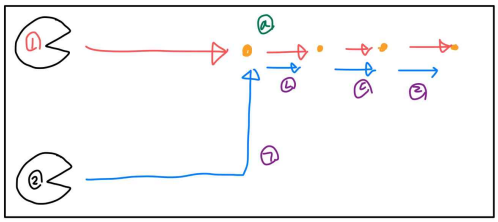
<MyAgent>

먼저 ClosestDotAgent(baseline) 쪽을 보자. Food(a)는 pacman1과 pacman2에게 가장 가깝기 때문에 두 agent 모두 a를 먹으려고 할 것이다. 1이 먼저 a를 먹었지만 2는 똑같이 a로 향할 것이고 마찬가지로 1이 먼저 b를 먹었지만 2는 b로 향할 것이다. 이후 1과 2는 가장 가까이 있는 Food(c)를 향해 간다. 심각한 동선 낭비를 확인할 수 있다.

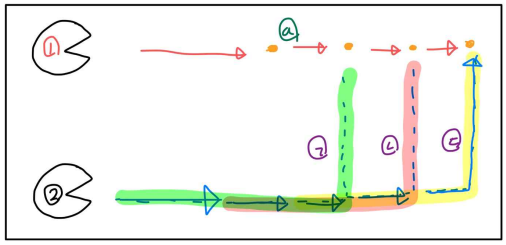
MyAgent에서는 이러한 문제를 해결하였다. 1은 가장 가까이 있는 a를 먹으러 간다. 이때 2는 1이 a로 향하는 것을 알고 가장 가까운 c로 향할 것이다. 1은 a를 먹고 가장 가까이 있는 b로 향한다.

ClosetDotAgent에서는 두 pacman agent가 만났을 때 문제가 발생한다. 두 agent가 만나면 두 agent로부터 가장 가까운 먹이는 하나이므로 그 먹이를 향해 두 agent가 같이 움직인다. 이는 심각한 동선 낭비를 초래한다. 반면 MyAgent는 같은 Food를 향해 두 agent가 같이 움직이지 않는다. 이는 pacman agent의 동선 문제가 개선되었음을 보여준다.

• Discuss cases where the agent implemented by yourself is worse than the baseline.



<ClosestDotAgent>



<MyAgent>

먼저 ClosestDotAgent를 보자. 1과 2에게 a가 가장 가까우므로, 1과 2는 동시에 a를 향해 간다. 1은 a를 먹은 후 바로 오른쪽 먹이를 향해가고 2도 1의 동선을 그대로 따라간다(ㄱ->ㄴ->ㄷ->ㄹ).

MyAgent에서는 1이 a를 먹고 바로 오른쪽에 있는 먹이로 목적지를 정한 순간, 1의 경로가 2의 남은 경로보다 짧아지게 된다. 따라서 이때 2의 동선이 초기화된다. 2는 탐색을 통해 a에서 두 번째로 오른쪽에 있는 먹이를 먹기 위한 경로를 설정한다(ㄴ). 하지만 1이 a의 바로 오른쪽에 있는 먹이를 먹고 그 옆에 있는 먹이를 목적지로 정한 순간, 다시 2의 동선은 초기화된다.

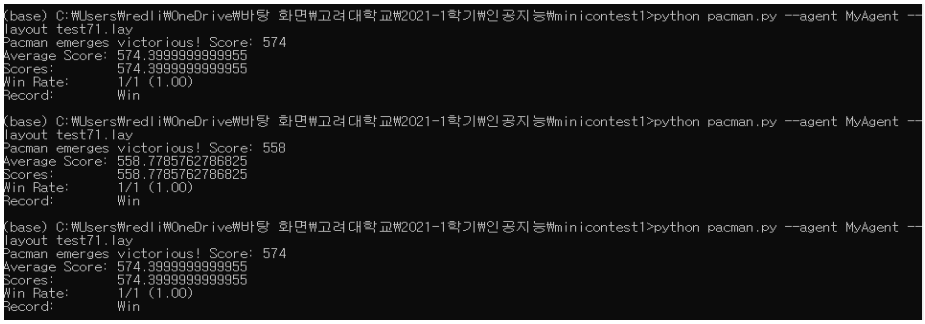
위의 상황에서는 ClosestDotAgent에서 탐색한 동선의 길이의 합보다 MyAgent에서 탐색한 동선의 길이의 합이 더 크다. 이는 MyAgent가 ClosestDotAgent보다 특정 상황에 대해 탐색시간이 긴 것을 의미한다. Food가 연달아 있을 때, MyAgent의 경우 경로를 재탐색하는 경우가 많아져 탐색시간이 길어진다. 하지만 ClosestDotAgent의 경우 그저 가까운 Food로만 목적지를 설정하기 때문에 이동 중 경로를 다시 탐색하는 경우는 없어 탐색시간이 상대적으로 짧아진다.

• Ask&Answer your own question about the above discussion.

Q) 이 게임에서는 '다음 action을 계산하기 위한 탐색시간 x 1000' 씩 점수가 깎이고, 한 번 action을 취할 때마다 0.4점씩 획득하게 된다. 그렇다면 몇 개의 agent는 탐색시간을 최소화하기 위해 단순히 갈 수 있는 곳으로만 이동하고, 나머지 agent는 경로를 탐색한 다음 action을 취한다면 시간이 지날수록 점수가 깎이는 것이 아니라 오를 수도 있지 않을까? 탐색 후 행동하는 agent와 탐색하지 않고 행동하는 agent의 수를 변화시키면서 게임을 진행해 보아라.

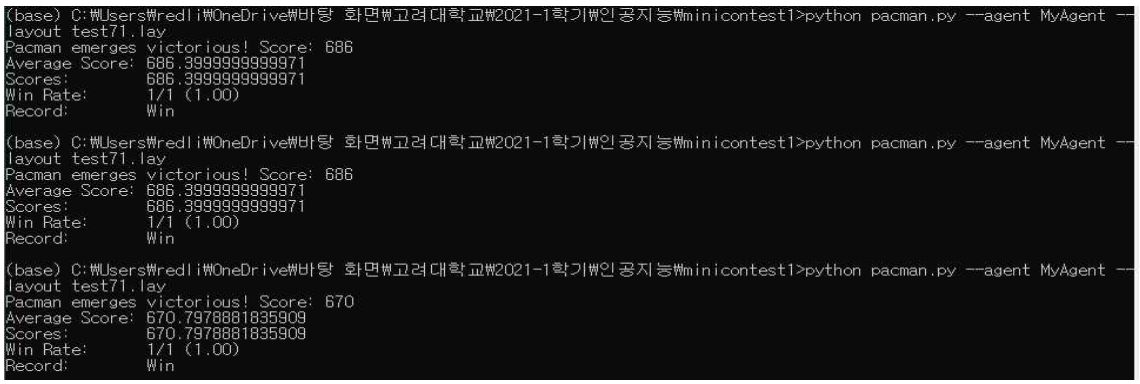
A)

1) 탐색하는 agent : 1개, 탐색하지 않고 행동하는 agent : 3개



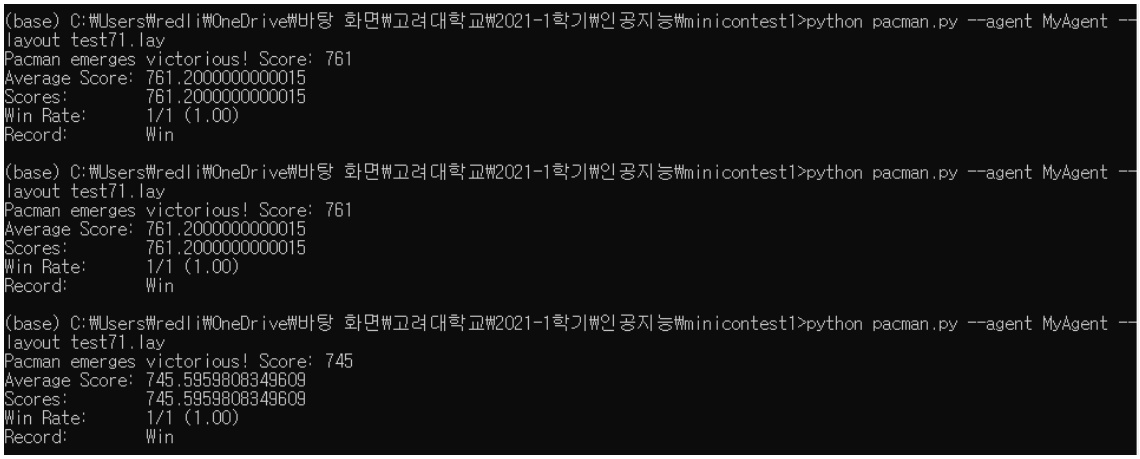
agent 3개가 탐색을 거치지 않고 제자리에서 왔다 갔다 반복을 하고, 나머지 하나의 agent는 탐색을 수행한 후 이동하였다. game에서 승리하기는 했지만, 점수는 낮았다. agent들이 action을 할 때마다 점수가 오를 것이라 예상했지만 천천히 깎였다. 한 번 action을 취해 얻는 점수보다 1개의 agent의 탐색시간에 의해 깎이는 점수가 더 컸기 때문에 발생한 결과였다.

2) 탐색하는 agent : 2개, 탐색하지 않고 행동하는 agent : 2개



탐색하지 않고 행동하는 agent가 1개 줄고, 탐색 후 행동하는 agent가 1개 늘었다. 첫 번째 경우보다 점수가 약 100점 정도 상승했다.

3) 탐색하는 agent : 3개, 탐색하지 않고 행동하는 agent : 1개



탐색하지 않고 행동하는 agent가 1개 줄고, 탐색 후 행동하는 agent가 1개 늘었다. 첫 번째 경우보다 점수가 약 100점 정도 상승했다.

위의 세 가지 경우를 보았을 때, 한 번 action을 취할 때 받는 0.4점은 탐색시간에 의해 깎이는 점수에 비하면 영향이 거의 없었다. 따라서 높은 점수를 받기 위해서는 최대한 많은 agent가 탐색에 참여하여 모든 먹이를 빠르게 먹는 것이 더 효율적인 전략임을 알 수 있다.