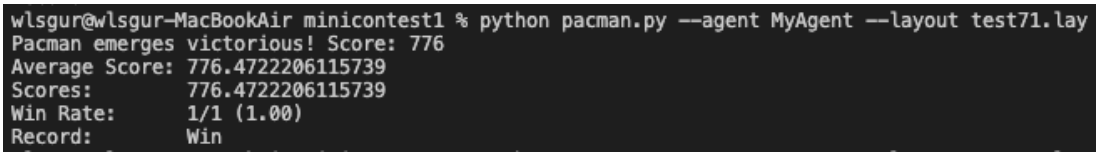
COSE361-03 인공지능
Assignment #3
컴퓨터학과 2022320006 최진혁

a. Capture the result of pacman.py with layout test71.lay



b. Description of your agents.

우선 MyAgent를 구현하는 데에 사용한 핵심적인 요소는 팩맨들끼리의 경로가 겹치지 않도록 하여 분산시키는 것, 그리고 Dynamic programing이라고 할 수 있다.

먼저 팩맨들의 탐색에는 조금 변형된 BFS 알고리즘을 사용해서 가장 가까운 food를 찾고 이를 향해 이동하도록 했다. 이 알고리즘은 일반적인 BFS와 비슷하지만, 팩맨들끼리의 경로가 겹치지 않도록 하였다. 우선 각 팩맨들마다 현재 향하고 있는 타겟 food, 그리고 해당 타겟까지의 거리를 다른 모든 팩맨들과 공유하도록 했다. 그래서 특정 팩맨이 BFS를 통해 food를 찾으면, 이 food가 다른 팩맨의 타겟인지부터 확인하여 다른 팩맨의 타겟이 아닌 경우에 큐에 push했다. Food를 pop했을 경우 지금까지의 모든 경로와 food의 좌표를 리턴한다.

원래는 모든 successor를 큐에 삽입하고, food가 다른 팩맨의 타겟인 경우 리턴하지 않는 식으로 했다. 그러나 이 경우엔 좁은 길목에서 한 팩맨이 길목의 food를 타겟으로 삼으면, 다른 팩맨들은 그 뒤의 food를 타겟으로 삼아 모든 팩맨이 이동해 연산량이 많아졌다. 즉 팩맨끼리의 타겟은 안 겹치지만, 동선은 겹치는 것이다. 그래서 큐에 삽입을 안 하는 방식으로 하여, 해당 좌표를 다른 팩맨들이 벽처럼 생각하도록 했다.

그렇지만 만약 거리가 먼 팩맨이 좁은 길목의 food를 먼저 타겟팅하면, 가까운 팩맨은 정지하여 효율성이 떨어지는 경우가 발생했다. 그래서 만약 팩맨이 찾은 food가 이미 다른 팩맨의 타겟이었던 경우, 이 두 팩맨이 food까지 얼마나 가까운지 계산하여, 더 가까운 팩맨에게 food를 할당하고 다른 팩맨은 새로 탐색하도록 했다.

이렇게 변형된 BFS로 팩맨의 타겟 좌표와 타겟까지의 액션 리스트를 저장한다. 만약 팩맨의 액션 리스트가 구해지지 않는다면 reachable food가 없으므로 정지하도록 했다. 그런데 만약 좁은 길목이 미로의 초반에 있고 팩맨들이 모여있다면, 초기에 하나의 팩맨을 제외한 나머지 팩맨들이 모두 영원히 정지해버리는 경우가 있었다. 그래서 정지한 팩맨들이 정지해있던 시간을 측정하고, 일정 시간이 지났을때 BFS 탐색을 한 번 더 해보고 reachable food가 있는지 확인하도록 했다.

그리고 getAction 함수는 다음 액션 하나만 리턴해야 하므로 BFS로 구한 팩맨의 액션 리스트 중 첫 번째를 삭제하면서 리턴하도록 했다. 또 만약 팩맨의 액션 리스트가 비어있지 않은 경우에는 BFS 탐색을 새로 하지 않고 저장해두었던 액션 리스트를 활용해 연산량을 줄일 수 있도록 하였다.

c. Three discussions

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

여러 팩맨이 한 지점에서 시작하는 경우 나의 알고리즘이 더 좋은 효율을 보일 수 있다. 팩맨이 여러마리이고 빠른 시간 안에 모든 food를 먹어야 하므로, 팩맨들을 분산시키는 것이 더 높은 점수를 받을 수 있다. 그런데 만약 baseline인 closestDotAgent의 경우, 여러 마리의 팩맨이 같은 food를 향해 겹쳐 이동하므로 사실상 다수의 팩맨의 이점을 얻지 못하고 효율성이 떨어진다.

또 food간 사이가 멀수록 나의 알고리즘이 더 효율적이다. Baseline은 액션을 취할 때마다 bfs를 수행한다. 하지만 나의 알고리즘은 새로운 food를 찾을 때만 bfs를 수행하고, food를 향해 가고 있던 경우 리스트에 저장된 경로를 활용하므로 연산량이 줄어들게 된다.

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

팩맨들의 경로가 이미 미로에 의해 나누어져 있고, food간 거리가 촘촘하거나 거의 없이 빽빽한 경우에는 baseline이 더 효율적일 수도 있다. Food가 촘촘하게 배치되어 있는 경우, BFS를 수행하는 것이 많은 노드를 expand 시키지 않는다. 또 계속해서 새로운 food를 탐색하게 되므로, 나의 알고리즘도 많은 BFS 탐색을 하게 된다. 따라서 BFS 연산량은 baseline과 나의 알고리즘이 비슷하게 된다. 게다가 이미 미로에 의해 팩맨들의 경로가 겹치지 않기 때문에, 오히려 팩맨들의 타겟이 겹치는지를 계속 계산해야 하는 나의 알고리즘이 더 많은 연산을 필요로 하여 비효율적일 수 있다.

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

Q: 만약 현재 상황에 유령(적)이 추가가 된다면 알고리즘을 어떻게 바꾸어야 할까?

A:

우선 opponent가 존재하기 때문에 minimax search를 고려해 볼 수 있다. 다만 많은 노드가 expand 되기 때문에 연산량이 문제가 될 수도 있다.

둘째로는 현재 나의 알고리즘을 활용하되, 유령까지의 거리를 추가적으로 계산하여 유령과 만나지 않는 것을 우선시 하도록 수정할 수도 있다. 그렇지만 이 경우에도 유령과 팩맨이 계속 움직이기 때문에, 계속해서 유령까지의 거리를 계산하는 것 역시 많은 노드를 expand 시키게 된다.

그래서 depth limited BFS를 이용해 팩맨의 일정 범위 내에 유령이 있는 경우에만 반대 방향으로 도망가도록 액션을 취하는 방식을 고려해볼 수 있을 것 같다. 이 외에도 다양한 접근방법이 존재할 것 같지만, 이번 과제처럼 직접 팩맨의 움직임을 살펴보며 최적의 방법을 찾아나가는 과정이 필요할 것 같다.