ARTIFICIAL INTELLIGENCE MIDTERM

2016320187 홍성윤

1. Result Screenshot

		your_best(red)
		<Average Winning Rate>
	your_base1	0.8
	your_base2	0.8
	your_base3	1
	baseline	1
	Num_Win	4
	Avg_Winning_Rate	0.9
		<Average Scores>
	your_base1	5.5
	your_base2	6.7
	your_base3	6.9
	baesline	7.3
	Avg_Score	6.6

2. Discussion

1) Baseline1

- **설명** : Baseline과 기본적인 아이디어는 같다. 적극적으로 공격을 나가는 ActiveAgent와 수비를 적극적으로 하는 PassiveAgent가 있다. 두 agent 모두 reflex agent이다. 먼저 baseline에서의 offensiveagent가 단순히 가까운 음식을 향해 나아간 것에 비해 baseline1에서는 가까운 음식과의 거리 뿐만 아니라 상대 고스트와의 거리 또한 고려하도록 하였다. 이외에도 score, 캡슐 획득 여부, 남은 음식 개수 등을 사용했다. 상대 고스트가 scared 상태일 때는 고스트와의 거리에 상관없이 음식을 먹도록 했다. Passiveagent 또한 남은 지켜야 될 음식 개수, 상대 팩맨과의 거리 등을 평가함수에 사용했다. Activeagent는 음식을 두 개 이상 먹으면 안정적인 점수 획득을 위해서 다시 자신의 진영으로 복귀하도록 작성했다. 팩맨은 음식을 두 개 이상 먹으면 복귀하기 때문에 한 번에 다득점은 불가능하지만 비교적 득점 성공률이 높고 baseline에 비해 높은 점수를 기록한다.

- **평가** : Baseline1은 baseline에 비해 승률이 좋지만 여러 문제들이 남아있다. 먼저 팩맨이 공격을 할 때, agent가 reflex하기 때문에 현재상태만 보고 행동을 결정하게 된다. 따라서 당장의 음식 때문에 고스트에게 잡아 먹히는 등 reflex agent의 한계를 보여준

다. 또한 두 agent의 역할이 고정되어 있기 때문에 active agent가 상대 팩맨을 매우 쉽게 먹을 수 있음에도 공격을 우선시해 무시하는 경우, 의미 없는 수비만 계속 하는 경우 등이 다소 발생한다.

2) Baseline2

- **설명** : Baseline1의 activeagent, passiveagent와 기본적인 알고리즘은 같다. 하지만 baseline1의 가장 큰 문제점 중에 하나였던 고정적인 역할을 개선하기 위해 두 agent가 상황에 따라서 서로의 역할을 바꿀 수 있도록 작성했다. 각각의 게임 상태는 상대의 팩맨 수에 따라 크게 세 가지 경우로 나누어 진다. 상대가 모두 팩맨으로 공격을 하고 있는 경우, 상대 중 하나만 공격 중인 경우, 상대가 모두 수비 중인 경우이다. 상대가 모두 공격 중일 때는 실점을 막기위해 우리 팀 모두 수비를 하고, 상대가 모두 고스트 일 때는 막을 팩맨이 없으므로 우리 팀 모두 공격을 진행한다. 상대가 하나만 공격 중인 경우, 두 agent 중 더 가까이 있는 agent가 수비하도록 했다. 물론 한 agent는 공격을 집중적으로 하고 다른 하나는 수비를 집중적으로 하는 방식도 어느 정도 승률을 보장하지만 유동적인 역할 변경으로 승률 및 점수 향상을 기대할 수 있다.

- **평가** : Baseline2 또한 팩맨이 음식을 두 개 이상 먹으면 복귀하도록 작성되어 상대 입장에서 reflex agent임에도 비교적 수비하기가 까다롭다. 특히 두 agent가 모두 팩맨일 가능성이 존재하기 때문에 baseline1에 비해 평균점수가 높다. 실제로 bestline과의 게임에서 작은 확률로 이기는 경우가 존재한다.

3) Baseline3

- **설명** : Baseline3는 baseline1과 baseline2에서의 reflex agent를 개선하기 위해 minimax 알고리즘을 사용하였다. Baseline1에서와 같이 두 agent를 각각 activeagent, passiveagent 로 설정했고 activeagent가 팩맨이 되어 공격을 나갈 때, minimax 알고리즘을 통해 보다 안전하고 효율적으로 득점을 할 수 있도록 했다. 안정적인 득점을 위해 팩맨이 음식을 6개 이상 먹었다면 복귀하도록 하였다. 이전에 baseline1에서처럼 똑같이 복귀할 기준 음식 수를 두 개로 하면 효율이 떨어진다고 판단해서 baseline3에서는 기준을 6개로 설정했다. Activeagent는 상대가 모두 공격 중일때는 passiveagent와 같이 수비를 하지만 이 외에 다른 상황에서는 가까운 음식을 향해 가도록 작성했다. 단, 가까운 음식 근처에 상대가 있다면 단순히 음식만 쫓아가지 않도록 작성했다.

- **계산속도** : Minimax 알고리즘을 사용하니 보다 효율적인 선택이 가능했지만 이전보다 훨씬 느린 계산속도를 보여줬다. 특히 상대 고스트가 둘일 때는 탐색 속도가 매우 느려 거의 정지하다시피 했고 기본적으로 탐색 깊이를 4 이상으로 하면 계산시간이 최소 1초 이상 걸렸다. 이를 해결하기 위해 알파베타 프루닝을 사용했다. 프루닝 코드를

추가하고 나니 탐색 시간이 눈에 띄게 감소한 것을 확인할 수 있었다.

- **평가1** : Minimax agent는 reflex agent보다 훨씬 더 안전하고 빠르게 많은 음식을 획득한다. 또한 몇 수 앞을 보기 때문에 미리 최악의 상황을 피할 수 있다. 그러나 minimax agent 또한 좋지 못한 행동을 보일 때가 많았고, leaf node 에서의 점수를 어떻게 결정하는지에 따라 성능이 매우 달라질 수 있음을 알았다.

- **평가2** : Minimax agent가 reflex agent보다 효율적으로 음식을 먹긴 하지만 baseline3의 baseline2에 대한 승률이 높은 것은 아니다. Baseline2의 agent들은 음식을 두 개만 먹으면 바로 복귀해 점수를 얻기 때문에 baseline3의 agent가 혼자서 두 팩맨 모두 수비하기에는 매우 까다롭다.

4) Bestline

- **설명** : baseline3의 minimax agent와 baseline2의 flexible agent의 장점을 합쳐 만든 알고리즘이다. 기본적으로 두 agent는 baseline2와 같이 상대의 공격 상태에 따라 자신의 역할을 결정한다. 공격을 하는 agent는 가지치기 된 minimax 알고리즘을 사용한다.

- **평가1** : 이 알고리즘은 baseline2, 3의 장점을 모두 가져 효율이 가장 좋다. 이전에 minimax agent의 아쉬운 점도 있었지만 bestline에서 두 agent가 모두 팩맥으로 공격을 할 경우 상대 입장에서 훨씬 수비하기가 어려워진다.

- **평가2** : minimax agent도 사용하고 역할도 유동적으로 바꾸는 bestline이 baseline1은 쉽게 이길 거 같지만 의외로 baseline1이 선전하는 모습을 보일 때가 있고 실제로 bestline의 승률도 1이 아니다. 이는 baseline1이 음식을 두 개만 먹고 복귀하기 때문이다. 따라서 bestline이 완벽하게 수비를 해내지 못하는 경우가 비교적 많이 발생한다. 이와 같은 설명은 baseline2와의 비교에서도 가능하다. Baseline2 또한 같은 이유로 bestline의 agent가 득점을 완전히 막아내지 못하는 경우가 발생한다. Baselin1, 2와의 평균점수가 가장 낮고 이 두 알고리즘만 승률이 1 미만인 것을 통해 reflex agent 임에도 평가함수에 따라 성능이 크게 달라질 수 있음을 느꼈다.

- **개선점(필요성을 느꼈으나 해결하지 못한 사항)** : bestline에서 또한 안정적인 득점을 위한 기준 음식 수를 설정했다. Reflex agent 보다 높게 기준을 설정해도 득점 성공률이 높긴 하지만, 특정 값으로 복귀 기준을 정하면 비효율적인 상황이 많이 발생할 수 있다. 따로 고정된 기준 값 없이 agent가 스스로 언제 복귀할 지를 판단할 수 있도록 bestline을 개선할 수 있을 것이다.

  대부분의 상황에서 minimax agent는 좋은 길을 찾아가지만 음식이 미로의 깊숙한 부분에 위치하는 경우 좋지 못한 성능을 보였다. Minimax 알고리즘의 평가함수를 개선해서 이와 같은 문제를 해결할 수 있을 것이다.