인공지능 final project

김정우
고려대학교 컴퓨터학과

1 Introduction

기본적인 구현 방식은 baseline의 방식(feature - weight를 선형결합해서 evaluation하는 방식)을 차용했습니다. food가 멀리 있을 때는 가장 가까운 food 방향쪽으로 이동할 수 있도록 minFoodDistance를 이용한다거나, 침입한 상대와의 거리를 측정해 그 방향으로 이동하게 하는 등 기본적인 작동 원리는 같습니다. 그러나 음식을 먹는 것 자체에 가산점을 두어 공격성을 강화하고, 근처에 상대 agent가 오면 특정 수식에 그 거리를 대입해 민감하게 반응하도록 만들었습니다. 또한 food를 하나 먹으면 즉시 집으로 돌아가도록 하여 안정적으로 점수를 획득하도록 조작했습니다.

원래는 탐색의 depth를 설정하여 minimax 방식을 적용하고자 했으나, 제한시간 1초를 넘어가는 경우가 빈번하고 생각만큼 높은 지능을 가지지 않는 것으로 보여 수용하지 않았습니다.

2 Method

3가지 방식 모두 동일한 방식을 채택하지만, Offensive agent, Defensive agent의 수에서 차이가 있습니다.

```python
class OffensiveAgent(DummyAgent):
  """
  A reflex agent that seeks food. This is an agent
  we give you to get an idea of what an offensive agent might look like,
  but it is by no means the best or only way to build an offensive agent.
  """

  def getFeatures(self, gameState, action):
    global foodList
    features = util.Counter()
    successor = self.getSuccessor(gameState, action)
    curfoodList = self.getFood(gameState)
    foodList = self.getFood(successor).asList()
    myState = successor.getAgentState(self.index)
    myPos = successor.getAgentState(self.index).getPosition()
    wall = gameState.getWalls()
    score = 0

    if curfoodList[int(myPos[0])][int(myPos[1])]:
        score += 200

    enems = [successor.getAgentState(i) for i in self.getOpponents(successor)]
    invs = [a for a in enems if a.isPacman and a.getPosition() != None]
    non_invs = [a for a in enems if not a.isPacman and a.getPosition() != None]

    if len(foodList) > 0 and self.foodnum == 0 and len(invs) == 0:
      minFoodDistance = min([self.getMazeDistance(myPos, food) for food in foodList])
      features['mindistFood'] = minFoodDistance

    gdis_min = wall.height + wall.width
```

offensive agent는 적 pacman이 아군 영역에 들어온 것을 신경쓰지 않습니다.

if curfoodList[int .. 부분은 agent의 다음 행동이 food를 먹는 행동이면 200점을 가산한다는 의미입니다.

```
if myState.isPacman :
        sight = 6
else :
        sight = 3

if len(non_invs) > 0:
  for ghost in non_invs:
    if ghost.scaredTimer == 0:
      gdis_min = min(self.getMazeDistance(myPos, ghost.getPosition()), gdis_min)
  if gdis_min < sight:
        score -= 20 ** (sight + 1 - gdis_min)

features['successorScore'] = score

if self.foodnum > 0:
  minHomeDistance = min([self.getMazeDistance(myPos, self.start)])
  features['back'] = minHomeDistance

if action == Directions.STOP: features['stop'] = 1
return features
```

sight는 agent의 시야를 의미합니다. sight보다 ghost와의 distance가 적은 경우에만 영향을 미치도록 조건문을 제시했습니다.
지수함수를 차용하여 가까울수록 큰 영향을 받게 설정했습니다.

```
    if curfoodList[int(myPos[0])][int(myPos[1])]:
        score += 50
```

defensive agent는 offensive와 크게 다르지 않습니다. 다만 food를 먹었을 때의 score 증가치를 하향 조정했습니다.

```
gdis_min = wall.height + wall.width

#if myState.isPacman :
#      sight = 3
#else :
#      sight = 3

sight = 3

if len(non_invs) > 0:
  for ghost in non_invs:
    if ghost.scaredTimer == 0:
      gdis_min = min(self.getMazeDistance(myPos, ghost.getPosition()), gdis_min)
  if gdis_min < sight:
        score -= 50 ** (sight + 1 - gdis_min)


features['successorScore'] = int(score)

if self.foodnum > 0:
  minHomeDistance = min([self.getMazeDistance(myPos, self.start)])
  features['back'] = minHomeDistance

if len(invs) > 0:
  dists = [self.getMazeDistance(myPos, a.getPosition()) for a in invs]
  features['invaderDistance'] = min(dists)

if action == Directions.STOP: features['stop'] = 1
return features
```

또한 몇 번의 시행착오 끝에, sight를 고정하는 것이 defensive에 유리함을 경험적으로 알게 되었습니다. 이에 sight를 3으로 고정합니다.

feature['invaderDistance']를 통해, 상대 agent가 pacman이 되어 들어오면 즉시 돌아와 방어할 수 있도록 설계했습니다. 음의 가중치(weight)를 두어 ghost가 가까워질수록 점수를 얻게 됩니다.

baseline1은 offensive 2개, baseline2는 defensive 2개, baseline3는 각 하나씩 가지고 있는 코드입니다.

3 Result

	Column1	Column2
2		your_best(red)
3		<Average Winning Rate>
4	your_base1	0.8
5	your_base2	-0.0
6	your_base3	-0.2
7	baseline	1.0
8	Num_Win	2.0
9	Avg_Winning_Rate	0.4
10		<Average Scores>
11	your_base1	12.0
12	your_base2	0.0
13	your_base3	-0.2
14	baesline	5.1
15	Avg_Score	4.225

baseline1은 offensive한 특성상 변수가 너무 많아 좋지 않은 코드라 확신했습니다. 이에 baseline2와 baseline3중 best를 고민했는데, 안정적인 것은 역시 baseline2였습니다. 움직임이 소극적이고 안정적이라 움직이는 방식이 크게 변하지 않았습니다. 다만 baseline3에 약 열세인 모습을 보였습니다.

baseline3는 제 3가지 방식들끼리 겨룰 때에는 가장 뛰어났습니다. 그러나 잘 짜인 코드를 만나면 offensive agent가 무기력하게 봉쇄당하고 그 사이에 점수를 많이 뺏길 것 같다는 생각이 들었습니다. 그래서 best case를 baseline 2로 정했습니다.

4 Conclusion & Free Discusion

depth를 늘려서 탐색하는 방식을 차용하지 못하니, 상대 진영에 들어갔을때 충분히 빠져나올 수 있는데 그렇게 하지 못하는 경우가 빈번했습니다. 이를 개선하지 못해 몹시 아쉽습니다.

그리고 offensive agent 두 개가, 상대 ghost 하나에 막히는 경우가 있었는데 이런 경우에 목적지를 재설정하는 알고리즘을 짤 수 있다면 좋을 것 같습니다.

마지막으로, bfs를 통해 가장 가까운 food까지 가는 경로에 가산점을 부여하고 싶었으나 그것이 스스로 ghost 앞으로 가는 상황을 자주 만들 것 같아 수용하지 않았습니다.