인공지능 Mid Term Project

2018320196 컴퓨터학과 유지훈

이번 프로젝트에 제출한 4개의 모델의 큰 흐름은 다음과 같다.
첫번째 모델은 현재상황에 대해 반응하는 Reflex Agent를 먼저 구상하였다. 쓸모 있는 가치함수를 구상하는 것이 관건이었다. 두 agent는 각각 공격형, 방어형으로 행동양식을 나누었다. 첫번째 모델을 완성해 나가던 중 공격형 요원이 잡혔을 때 시작점에서 다시 상대방 진영으로 넘어가야하는 것을 개선해보고자, 공격형 요원이 잡혔을 때 기존의 방어형 요원과 행동양식을 바꾸는 방안을 고안하였고 이를 두번째 모델로 구현하였다.
세번째와 네번째 모델은 기존의 reflex 모델에서 발전하여 몇 수 앞을 내다보는 alpha-beta pruning, expectimax model을 구상하였다.
기존의 baseline모델에서 활용도가 높은 부분들은 이용하였고, 네 가지 모델들의 경우 getFeature함수가 거의 흡사하다. 네 모델들 간의 중요한 차이점은 행동양식의 유동성과, reflex 모델인지 혹은 몇 수 앞을 미리 계산하여 예상하는 모델인지로 볼 수 있다.

Yourbaseline1

Reflex 모델임에도 기대 이상의 성능을 보여준 모델이다. 이 모델은 baseline 모델의 뼈대를 최대한 활용하였으며, 주목할 점은 getFeatures 함수와 getWeights함수이다.
공격형 요원에 두 가지 행동양식을 도입하였다. onAttack은 상대방의 진영에 들어가 음식을 먹는 것이고, onReturn은 일정량 이상의 dot을 들고 있거나, 상대방 유령에게 급히 쫓기고있을 때, 자신의 진영으로 복귀하여 점수를 올리는 것이다.
Feature에 차용된 특징들을 살펴보면, score는 가치함수의 베이스가 되는 점수를 형성하여 더욱 완성도 높은 가치함수를 만드는데 도움을 준다.
Stop과 reverse는 액션에 대한 패널티를 부과한다. Stop액션의 경우 가치함수의 허점에 의해 움직이는 것이 유리한 상황임에도 분명한 원인의 파악을 못하고 멈춰 있는 경우가 있기에, 이를 방지하고자 멈춰 있는 상태에 대해 마이너스 가치를 부가함으로 패널티를 준다. Reverse의 경우는 뚜렷한 방향없이 의미 없는 반복적인 행동에 대해 패널티를 부과한다. 더불어 레이아웃이 대부분 시작지점에서 긴 통로를 지나 경쟁이 벌어지는 필드가 열리기에 긴 통로를 문제없이 통과하는데 좋은 역할을 해준다.
onAttack 요소는 아직 상대 진영에 진입하지 못한 agent가 상대 진영에 침입하여 공격활동을 벌이도록 독려하는 요소이다. 아직 유령상태인 agent에게는 마이너스 가치가 되어 진영의 경계에서 이를 벗어나는데 도움을 준다. 또한 복귀 상태인 agent에게도 마이너스 가치가 되도록 설계하여 마찬가지로 진영의 경계를 넘는데 추진력을 준다.
eatCapsule은 캡슐을 먹을 수 있다면, 먹음으로써 공격활동에 도움이 되도록 하였다.
numCarrying은 현재 agent가 들고있는 dot의 수를 나타내는 속성으로 dot을 먹는 것이 중요함을 알리기 위해 사용하였다.

numReturned는 반환하는데 성공한 dot의 수로 onReturn인 agent가 자신의 진영으로 돌아오도록 하는 강력한 동기가 되어준다.
상대방 진영의 유령들과의 거리 정보를 이용하여 길잡이로 이용하는 속성들로 ghostISclose와 eatGhost가 있다. ghostISclose는 일정 거리 이내로 유령이 있다면 이를 agent에게 경고해주고, 만약 바로 근접해 있다면 즉시 탈출을 강력히 권한다. eatGhost의 경우 상대방 유령이 캡슐에 의해 겁에 질린 상태라면 오히려 잡아먹어 공격활동에 도움이 되도록 한다.
dist는 agent가 복귀할 때 자신의 진영을 찾도록 해주는 역할을 한다. 시작점을 기준으로 찾는 방법과 방어형 팀원을 기준으로 찾는 방법 중 후자를 택하였으나, 상황에 따라 서로의 장단점이 있을 뿐 우열을 가릴 수는 없었다. 두 방법 외에 항상 자신의 위치에서 가장 가까운 자신의 진영의 위치를 알아내는 알고리즘을 적용한다면 훌륭한 개선책이 될 것이다.
distanceToFood는 자신과 dot들의 위치를 계산하고, 여기에 각 dot과 가장 이웃한 유령과의 거리를 계산하여 이를 전자에 빼주는 값을 이용하였다. 이는 오직 가장 가까운 dot만을 고려하는 것이 아니라, 유령과의 거리가 어느정도 있는 dot을 더 선호함을 알려준다.
더불어 getFeature함수내에서는 해당 agent의 공격형 행동양식 중 사냥과 복귀 둘을 전환하는 역할도 한다. 일정량의 dot을 모았을 때, 또는 상대방 유령이 매우 근접하면 복귀하도록 하였다. 사냥과 복귀 두 행동양식은 getWeight함수에서 서로 다른 가중치를 반환받아 가치함수를 계산하도록 하였다.
방어형 agent의 경우 baseline과 비슷한 점이 많은데, 여러 테스트 결과 생각보다 훌륭히 방어를 성공해주었기에 약간의 정보들을 첨가하여 getFeature함수를 구성하였다.
방어형 agent는 공격을 하지 않기에 유령상태를 유지하기 위해 onDefence를 이용하였고, 만약 침입자가 발생했다면, 즉시 경보가 울리듯 방어태세에 돌입한다. numInvaders와 invaderDistance에 큰 숫자를 넣어 침입자를 제거하는 것을 최우선에 두도록 한다. 만약 침입자가 아직 없는 상황이라면, 상대방의 공격 agent는 유령의 모습으로 공격하기위해 우리 진영에 들어올 것이다. 따라서 상대방의 유령과 가까운 위치를 정찰하다가 가까운 위치에 침입한 적을 즉시 잡기 위해 nearestGhost를 활용하였다.
이렇게 구성한 가치함수는 꽤나 유용하였고, 다른 모델들과 대결하였을 때 승리를 거두는 모습도 보였다. 이는 일정량의 dot을 채우면 복귀하는데 집중하는 면모가 승리로 이어지는 경우로 보인다.

Yourbaseline2

사실상 가치함수는 알고리즘의 적용을 위한 코딩부분에서 약간의 차이가 있을 뿐, 거의 동일하다. 두번째 모델이 첫번째 모델과의 차별점은 공수교대를 통해 신속한 공격이 가능해 진다는 점이다.

첫번째 모델에서 단점이라 생각한 것은 공격형 agent가 공격에 실패하고 부활하면 시작점에서 다시 상대 진영으로 가야 한다는 것이다. 이는 제한된 시간에서 패널티가 될 수 있는 부분이기에 같은 팀의 공격이 실패했을 때, 방어형 agent가 즉시 공격에 투입되는 전략을 구상하였다.

각 agent는 act_ver라는 변수에 팀원이 각각 맡고 있는 역할에 대한 정보를 관리한다. 0은 방어, 1은 공격, 2는 반환이다. 각 역할에 대한 가치함수와 행동양식은 첫번째 모델과 동일하다. 두번째 모델에서 중요한 알고리즘은 언제 어떻게 행동양식을 교체하는 것인지이다.

공수를 교대하는 시점은 무조건 우리팀 진영에 침입자가 없는 때이다. 만약 침입자를 쫓는 상황에서 공격에 투입되면 부활한 팀원은 침입자와 거리가 멀기에 방어에 취약해 진다. 따라서 침입자가 없는 상황에서 공격형 agent의 위치가 시작지점에 위치하게 되면 자신이 공격 해야 할 차례임을 알고 공격에 나서게 된다.

이 모델은 가장 큰 장점은 신속한 공격에 있다. 앞서 공격에 나간 agent가 다른 곳에서 공격에 실패하더라도, 방어형 agent는 앞에서 서술했던 것처럼 상대방의 예상 침입장소에서 대기하고 있기에 빠른 공격투입이 가능하다. 이는 상대방의 방어에 부담을 주고, 그럼에도 상대방의 공격에 충분히 강력한 방어를 제공할 수 있다는 장점이 있다.

구현한 모델에서도 신속한 공격이 잘 이루어졌지만, 아쉬운 점은 공격 알고리즘이 실패하는 경우가 빈번하다면, 그리 큰 효과를 보지 못한다는 것이었다. 공격 알고리즘을 더욱 정교하게 하여 적용한다면 두번째 모델의 큰 개선점이 될 것이다.

Yourbaseline3

세번째 모델은 기존의 두 모델의 reflex agent방식에서 adversarial agent 방식으로 바꾼 것이다. 이 모델에는 alpha-beta pruning을 도입하여 몇 수 앞을 내다보아 더욱 지능적인 플레이를 도모하였다.

알고리즘은 가치함수로 계산을 수행하는 value와, max-layer, min-layer를 구성하여 alpha-beta pruning을 구현하였다.

다만 이번 프로젝트에서 agent가 자신의 액션을 계산 해내는 것에 시간제한이 걸려있었기에 탐색공간을 절약하는 것이 중요하였다. 한계 깊이는 2로 설정하였으며, 탐색하는 액션 중에 stop액션을 제외하고 탐색하였다. 앞서 서술한 것처럼 stop모션은 그 자체로 큰 메리트가 없기 때문이다. 이를 통해 제한된 시간내에 2깊이 탐색을 실행할 수 있었다.

getFeature 함수에도 변화가 생겼는데, reflex agent들은 getFeature함수내에 다음 게임상태를 구하여 가치 계산에 이용하였는데, 여기서는 가치를 구하는 시점에서 전달받은

gameState가 이미 미래의 상태이기 때문에 그 상태 자체의 가치를 판단해야 했다.
가장 큰 이슈는, 공격형 agent가 사냥과 복귀 행동양식의 전환에 대한 동기를 주는 것이었다. 만약 agent가 하나의 dot만 더 가지고 있으면 복귀모드로 전환되어 점수를 얻고자 할 텐데, 복귀모드에서의 가치함수의 결과값이 사냥에서보다 작다면 agent는 dot을 먹지 않고 피하게 된다. 이는 가치함수의 설계와 매우 밀접한 관련이 있었다. 가치함수는 행동양식의 변화가 있을 때마다 동기를 부여를 해야 했다. 즉, 가치함수는 행동양식의 변화에서 그 값이 계속 커져가야만 했다. 이를 위해 bonus 요소를 삽입하여 agent가 사냥에서 복귀로 전환될 때, 복귀에서 사냥으로 전환될 때 부가 가치 점수를 부여함으로써 agent의 공격활동이 활발히 이루어 지도록 동기를 부여하였다.

Mybest

alpha-beta pruning 모델을 구현하고 무엇이 최선일지 고민하였다. 상대방은 항상 자신의 이익을 좇을 것이기 때문에 minimax 방식의 모델이 가장 좋을 것이라 예상하였지만, 그럼에도 expectimax agent가 궁금하여 네번째 모델로 구현하게 되었다. 모든 알고리즘의 적용은 세번째 모델과 동일하였다. 가치함수 또한 동일하였으며, 유일한 차이점은 min-layer가 exp-layer가 된 것이다. 그럼에도 이 모델은 다른 모델과 유의미한 차이가 있었다. mybest로 뽑은 이유는 상대방의 행동양식에 대해 내가 알 수 없으므로 기댓값을 이용한 가치판단이 minimax를 이용한 가치판단의 불확실성을 조금이나마 덜어주지 않았나 생각이 든다.
다음은 다른 모델들과의 대결 결과이다. 승률이 그다지 높지 않게 나오는데, 이기지 못한 승부의 대부분이 무승부임을 고려하면 개인적으로 만족하는 결과이다.
점수가 높이 않는 점은 공격 알고리즘이 그리 정교하지 않아 공격을 빈번히 실패하기 때문에 이를 개선하면 높은 점수를 얻을 수 있을 것임을 기대한다.

	your_best(red)	
	<Average Winning Rate>	
your_base	0.7	
your_base	0.6	
your_base	0.4	
baseline	1	
Num_Win	4	
Avg_Winn	0.675	
	<Average Scores>	
your_base	1	
your_base	3.6	
your_base	-1.8	
baesline	4.9	
Avg_Score	1.925	