Segundo Trabalho Prático – Laboratório de IA

Diego Ascânio Santos

1)

O código de ReflexAgent foi melhorado levando em conta as distâncias de Manhattan dos alimentos e dos fantasmas para a formulação dos scores. O código abaixo mostra a implementação:

```
    "*** YOUR CODE HERE ***"

    compare = lambda ghost_1, ghost_2, pacman: -1 if util.manhattanDistance(pacman, ghost_1) - util.manhattanD
istance(pacman, ghost_2) < 0 else 1 if util.manhattanDistance(pacman, ghost_1) - util.manhattanDistance(pacman
, ghost_2) > 0 else 0
    oldFoodList = oldFood.asList()
    oldFoodList.sort(lambda x, y: util.manhattanDistance(newPos, x)-util.manhattanDistance(newPos, y))
    foodScore = util.manhattanDistance(newPos, oldFoodList[0])
    ghostPositions = [g.getPosition() for g in newGhostStates]

    if len(ghostPositions) == 0:
        ghostScore = 0
    else:
        ghostPositions.sort(lambda x, y: compare(x,y,newPos))
        if util.manhattanDistance(newPos, ghostPositions[0]) == 0:
            return -99
        else:
            ghostScore = -10./util.manhattanDistance(newPos, ghostPositions[0])
    return 10. + ghostScore if foodScore == 0 else ghostScore + 1./float(foodScore)
```

O score para um ambiente sem fantasmas assumiria sempre o valor de 10 e ele sempre iria à comida mais próxima. Mas como o ambiente tem fantasmas, ele vai em direção à posição que dará mais chances do pacman sobreviver, aquela que o mantenha mais longe dos fantasmas e que ao mesmo tempo permite que ele alcance mais rapidamente seu objetivo.

A função de avaliação foi razoavelmente boa, pois, de cada 5 execuções, ele venceu 4 e morreu 1.

2)

O minimax foi construído e funcionou razoávelmente para o primeiro caso também, de cada 5 execuções, ele venceu 4. Respondendo à pergunta do enunciado, ele corre para o fantasma mais próximo, pois quando a derrota é inevitável, ele morre mais rápido (sendo menos penalizado por isso) indo em direção ao fantasma mais próximo.

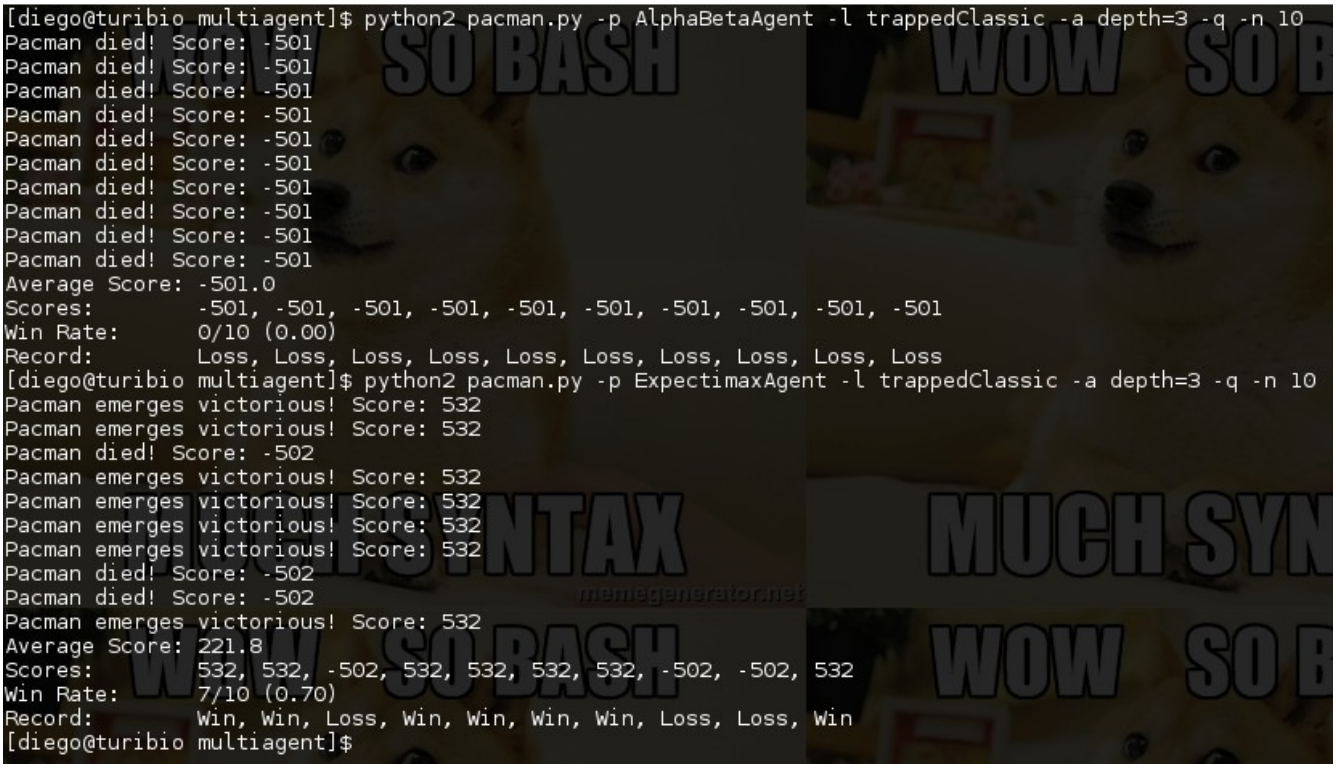
3)

O agente AlphaBetaAgent foi implementado e como esperado, a sua execução foi bem mais rápida do que a do minimax somente. Para uma profunidade 4 isso ficou evidente.

4)

Como o esperado, a poda AlfaBeta perdeu todas as vezes e o expectiminimax ganhou aproximadamente a metade das vezes.

Abaixo um print da execução mostrando o sucesso da implementação:


[diego@turibio multiagent]$ python2 pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores: -501, -501, -501, -501, -501, -501, -501, -501, -501, -501
Win Rate: 0/10 (0.00)
Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
[diego@turibio multiagent]$ python2 pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: 221.8
Scores: 532, 532, -502, 532, 532, 532, 532, -502, -502, 532
Win Rate: 7/10 (0.70)
Record: Win, Win, Loss, Win, Win, Win, Win, Loss, Loss, Win
[diego@turibio multiagent]$