Aplicação de técnicas de IA no domínio do Pacman - PARTE II

(Trabalho em triplas – entrega: ver DATAS)

O trabalho proposto é baseado no "Pacman Project" da Universidade de Berkeley, o qual visa aplicar conceitos de IA no domínio do Pacman.

Achando todos os cantos.

O novo problema de busca a ser resolvido pelo seu PacMan é encontrar o menor caminho no labirinto que atinge todos os quatro cantos (havendo ou não comida lá...). Note que dependendo da implementação, para alguns casos, como o *tinyCorners*, o menor caminho não vai para a comida mais próxima primeiro (o menor caminho tem 28 passos)!

Questão 1 (2 pontos)

Implemente o problema de busca `CornersProblem` no arquivo searchAgents.py. Agora você deve conseguir resolver:

```
pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

Dicas: Sua representação de estado deve codificar a informação de quais cantos já foram visitados. Não use a codificação de estado `gameState`, pois seu código vai ficar errado e muito lento. As informação relevantes do ambiente de jogo são a posição do Pac-Man e a situação dos cantos.

Questão 2 (3 pontos)

Implemente uma heurística admissível para o problema `CornersProblem` em `cornersHeuristic`.
Lembre-se: Para que uma função heurística seja admissível ele precisa retornar um valor menor ou igual ao menor caminho real para o objetivo mais próximo. Teste sua implementação com o problema abaixo:
`pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5`

(`AStarCornersAgent` é um atalho para `-p SearchAgent -a fn=aStarSearch, prob=CornersProblem,heuristic=cornersHeuristic`)

Você receberá 2 pontos pela heurística admissível. Além disso, receberá mais 0,5 ponto se a busca percorrer menos que 1.200 nós e mais 0,5 ponto se a busca percorrer menos que 1.000 nós (totalizando 3 pontos).

Comendo todos os pontos.

Agora vamos resolver um problema de busca mais difícil: fazer o Pac-Man comer tudo no menor número de passos. Para isso, vamos definir um novo problema de busca que formalize o problema da deglutição completa: `FoodSearchProblem` em searchAgents.py (que já está implementado). Uma solução é um caminho que coleta toda a comida disponível no mundo Pac-Man (você não precisa se preocupar com fantasmas ou outros elementos, somente com a comida e com as paredes). Se os seus métodos gerais de busca implementados no exercício anterior estiverem corretos, o `A*` com heurística nula deve achar rapidamente a solução ótima para o comando abaixo sem qualquer alteração com custo 7.

```
pacman.py -l testSearch -p AStarFoodSearchAgent
```

Nota:
`AStarFoodSearchAgent` é um atalho para `-p SearchAgent -a fn=astar, prob=FoodSearchProblem,heuristic=foodHeuristic`.

Contudo, você verá que o `A*` com heurística nula começa a ficar muito lento e é isso que você deve resolver...

Questão 3 (3 pontos)

Complete a função `foodHeuristic` em searchAgents.py com uma heurística admissível para o `FoodSearchProblem`. Teste seu agente com o tabuleiro `trickySearch`:

```
pacman.py -l trickySearch -p AStarFoodSearchAgent
```

Se sua heurística for admissível você ganha 2 pontos. Além disso, receberá mais 0,5 ponto se a busca percorrer menos que 12.000 nós e mais 0,5 ponto se a busca percorrer menos que 9.000 nós (totalizando 3 pontos).

Um bom teste para verificar a admissibilidade de sua heurística é testar seu desempenho com mediumSearch. Se você o resolve em pouco tempo, ou você é muito, muito perspicaz ou sua heurística não é admissível!

Busca Sub-ótima.

Pode ser difícil encontrar a solução ótima para o problema da comida, mesmo com A* e uma boa heurística. Nesse caso, uma solução razoavelmente boa é aceitável. Você deverá escrever um agente de busca local que sempre come o ponto mais próximo. A função `ClosestDotSearchAgent` está parcialmente implementada para você no arquivo searchAgents.py.

Questão 4 (2 pontos)

Implemente a função `findPathToClosestDot` no searchAgents.py. Resolva o problema abaixo (existe uma implementação sub-ótima de custo 350):

```
pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```