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

(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.

Pacman com Fantasmas

O código para esse trabalho é bastante parecido com o dos anteriores, mas comece com uma nova instalação. Você pode utilizar seus antigos search.py e searchAgents.py se quiser.

O código está disponível no Moodle como **multiagent.zip.**

Arquivos chave:

`multiAgents.py`	Seus agentes.
`pacman.py`	Arquivo principal que roda os jogos. O tipo `GameState` que será intensamente utilizado no trabalho está lá.
`game.py`	Lógica subjacente ao mundo PacMan. Tipos de suporte como AgentState, Agent, Direction, and Grid.
`util.py`	Estrutura de dados e algoritmos auxiliares.

O que submeter: Você vai preencher funções do arquivo `multiAgents.py`. Logo, o arquivo, com comentários, deve ser submetido. Por favor, NÃO altere arquivos auxiliares.

Avaliação: Por favor, não altere nome de classes ou métodos, já que o corretor automático será utilizado para auxiliar na verificação de erros.

Comece tentando um jogo clássico de PacMan:

```
pacman.py
```

Agora execute o `ReflexAgent` em `multiAgents.py`:

```
pacman.py -p ReflexAgent
```

Perceba que o desempenho dele é sofrível mesmo em cenários simples:

```
pacman.py -p ReflexAgent -l testClassic
```

Dê uma olhada no código (`multiAgents.py`) e certifique-se de que entendeu.

Questão 1 (1 ponto)

Melhore o `ReflexAgent` em `multiAgents.py`. O código apresenta alguns exemplos de como interagir com o tipo `GameState` para obter informações. Um bom agente reflexivo deverá considerar a posição da comida e dos fantasmas para ter um desempenho aceitável. Seu agente deve limpar (comer tudo) com facilidade o cenário `testClassic`:

```
pacman.py -p ReflexAgent -l testClassic
```

Exercite seu agente reflexivo com o cenário `mediumClassic` com um ou dois fantasmas:

```
pacman.py --frameTime 0 -p ReflexAgent -k 1

pacman.py --frameTime 0 -p ReflexAgent -k 2
```

Atenção: Os fantasmas têm comportamento aleatório. Você pode usar `-f` para rodar sempre com a mesma semente aleatória para testar o mesmo cenário. Desligue a parte gráfica com `-q` para executar mais rapidamente e `-n` para executar várias vezes.

Cheque seu desempenho usando o cenário `openClassic`. Um bom desempenho é morrer no máximo duas vezes em dez tentativas e comer toda comida.

```
pacman.py -p ReflexAgent -l openClassic -n 10 -q
```

Questão 2 (2 pontos)

Agora você vai escrever um agente de busca competitiva preenchendo a classe `MinimaxAgent` em `multiAgents.py`. Seu agente minimax deverá funcionar com qualquer número de agentes, então o algoritmo deve ser um pouco mais genérico do que o disponível no livro, mas comece com um fantasma... Em particular, sua árvore minimax terá várias camadas min (uma por fantasma) para cada camada max.
Seu código deve expandir a árvore até uma altura arbitrária. As folhas devem ser avaliadas com a função `self.evaluationFunction`, que usa `scoreEvaluationFunction`. `MinimaxAgent` extende `MultiAgentAgent`, que dá acesso a `self.depth` e `self.evaluationFunction`. Garanta que seu código minimax use essas duas variáveis já que as mesmas são inicializadas apropriadamente por linha de comando.

Importante*:* Uma jogada é considerada como o movimento do PacMan e de todos os fantasmas, logo profundidade 2 envolve dois movimentos do PacMan e de cada fantasma.

Dicas e Observações

- ✓ A função de avaliação já está fornecida (`self.evaluationFunction`). Você não deve alterá-la, mas deve ser capaz de perceber que avaliamos agora “estados” e não ações, como era para o agente reflexivo. Lembre-se que enquanto agentes reflexivos avaliam ações em relação ao estado corrente, agentes com planejamento avaliam estados futuros.

- ✓ Para testes, o valor de minimax do nó raiz usando o cenário `minimaxClassic` é 9, 8, 7, -492 para alturas 1, 2, 3 e 4 respectivamente. Nesse exemplo espera-se que o PacMan ganhe com alguma frequência.

  ```
  pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
  ```

- ✓ Em cenários maiores como `openClassic` e `mediumClassic` você perceberá que o PacMan é bom em não morrer, mas geralmente com desempenho medíocre quanto à comida. É normal que ele fique "enrolando" perto da comida, pois não sabe o que fazer.

- ✓ Quando o PacMan acredita que a morte é inevitável, ele tentará morrer o mais rápido possível:

  ```
  pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
  ```

Questão 3 (3 pontos)

Seu novo agente vai explorar a árvore usando poda alpha-beta, em `AlphaBetaAgent`. De novo seu algoritmo será um pouco mais genérico que o do livro e, de novo, comece com um único fantasma. Você deverá notar uma boa melhora de desempenho, talvez limite 3 alpha-beta funcione mais rápido que 2 normal. Idealmente, altura 3 no `smallClassic` deve rodar em poucos segundos por movimento.

```
pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

O valor de minimax é o mesmo da questão anterior.

Questão 4 (2 pontos)

Fantasmas aleatórios não são agentes minimax ótimos, logo modelá-los como tal não é apropriado. Preencha o método `ExpectimaxAgent`, onde você deve colocar um modelo do comportamento do fantasma. Para simplificar, suponha que você jogará contra `RandomGhost` que escolhem suas ações `getLegalActions` uniformemente aleatórias.
Você notará que seu PacMan ficará mais ousado, não ficando andando a deriva até que os fantasmas se aproximem. Em particular, se ele perceber que apesar de haver chances de ser encurralado ele ainda pode pegar mais comida, ele pelo menos tentará. Teste os seguintes cenários:

```
pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10

pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

Você deve ganhar em metade das vezes com `ExpectimaxAgent`, enquanto que o seu `AlphaBetaAgent` vai sempre perder. Entenda porque isso acontece.

Questão 5 (2 pontos)

Escreva uma nova função de avaliação para seu agente em `betterEvaluationFunction`. Agora você irá avaliar estados e não ações como no agente reflexivo. Você pode usar toda e qualquer informação disponível para escrever sua função de avaliação, inclusive seu código para busca ótima do projeto passado. Com profundidade de busca 2, sua função deve limpar o labirinto `smallClassic` com dois fantasmas em mais do que metade das vezes, num tempo razoável (para conseguir total, o PacMan deve fazer em torno de 1000 pontos quando ganha).

```
pacman.py -l smallClassic -p ExpectimaxAgent -a evalFn=better -q -n 10
```

Documente sua função de avaliação!

Dicas e observações

- ✓ Use diferentes informações (como a distância para comida) e não o valor do estado, diretamente.
- ✓ Uma abordagem interessante para a função de avaliação é a combinação linear de características. Ou seja, leve em conta tudo a respeito do estado que pode ser considerado importante e depois combine linearmente essas informações, multiplicando por pesos e somando. O peso reflete o quanto você acredita que aquela informação seja importante.