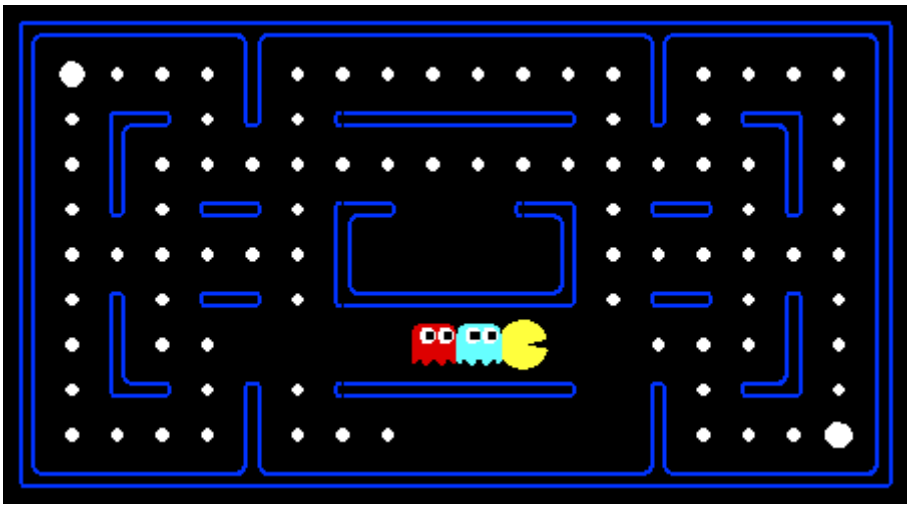
Trabalho 2: Pac-Man Multi-Agente

Este trabalho é parte do Pacman Project desenvolvido na UC Berkeley disciplina CS188 – Artificial Intelligence. Tradução realizada pela prof. Bianca Zadrozny, UFF. Adaptação realizada pelo prof. Eduardo Bezerra, CEFET/RJ).



Pac-Man, agora com fantasmas.
Minimax, Expectimax,
Avaliação.

Introdução

Neste projeto, você irá projetar agentes para a versão clássica do Pac-Man, incluindo fantasmas. Ao longo do caminho, você vai utilizar as buscas minimax e expectimax e projetar uma função de avaliação.

A base de código não mudou muito em relação ao trabalho anterior, mas, por favor, faça uma nova instalação, ao invés de utilizar os arquivos do primeiro trabalho. O código para este projeto contém os seguintes arquivos, disponíveis no arquivo denominado `multiagent.zip`.

Arquivos que devem ser lidos:

`multiAgents.py`	Onde todos os seus agentes de busca multi-agente vão ficar.
`pacman.py`	O arquivo principal que executa jogos Pac-Man. Este arquivo também descreve um tipo `GameState` para o Pac-Man, que você vai usar bastante neste trabalho.
`game.py`	A lógica por trás de como o mundo de Pac-Man funciona. Este arquivo descreve vários tipos de suporte como `AgentState`, `Agent`, `Direction` e `Grid`.

`util.py`	Estruturas de dados úteis para a implementação de algoritmos de busca.

Arquivos que você pode ignorar:

`graphicsDisplay.py`	Gráficos para o Pac-Man
`graphicsUtils.py`	Suporte gráfico para o Pac-Man
`textDisplay.py`	Gráficos ASCII para o Pac-Man
`ghostAgents.py`	Agentes de controle de fantasmas
`keyboardAgents.py`	Interfaces de teclado para controle do Pac-Man
`layout.py`	Código para leitura de arquivos de layout e armazenamento de seu conteúdo

O que entregar: Você irá preencher o arquivo `multiAgents.py` durante o trabalho. Você deve me enviar esse arquivo juntamente com um relatório contendo o resultados das execuções e as respostas das perguntas abaixo. Você também pode me enviar qualquer outro arquivo que tenha sido modificado por você (como `search.py`, etc.).

Pac-Man Multi-Agente

Primeiro, Jogue um jogo do Pac-Man Clássico:

```
python pacman.py
```

Agora, execute o código do agente reflexivo `ReflexAgent` que já está implementado em `multiAgents.py`:

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

Note que ele joga muito mal mesmo em layouts simples:

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

Inspecione o código dele (em `multiAgents.py`) e certifique-se de compreender o que ele está fazendo.

Passo 1 (2 pontos)

Melhore o código do `ReflexAgent` em `multiAgents.py` para que ele jogue decentemente. O código atual dá alguns exemplos de métodos úteis que consultam o estado do jogo (`GameState`) para obter informações. Um bom agente reflexivo deve considerar tanto as posições das comidas quanto as localizações dos fantasmas. O seu agente deve limpar facilmente o layout `testClassic`:

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

Experimente seu agente reflexivo no layout default `mediumClassic` com um ou dois fantasmas (e desligue a animação para acelerar a exibição):

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

Como é o desempenho do seu agente? É provável que muitas vezes ele morra com 2 fantasmas no tabuleiro default, a não ser que a sua função de avaliação seja muito boa.

Nota: você não pode colocar mais fantasmas do que o layout permite.

Note: Como características, tente o inverso de valores importantes (como a distância para comida) ao invés dos próprios valores.

Note: A função de avaliação que você está escrevendo está avaliando pares estado-ação; na próxima parte do trabalho (com busca competitiva), a função de avaliação avaliará estados.

Opções: Os fantasmas default são aleatórios; você também pode jogar com fantasmas mais espertos usando `-g DirectionalGhost`. Se a aleatoriedade está impedindo você de perceber se o seu agente está melhorando ou não, você pode usar `-f` para executar com uma semente aleatória fixa (mesmas escolhas aleatórias a cada jogo). Você também pode jogar vários jogos em seguida com `-n`. A parte gráfica pode ser desligada com `-q` para executar muitos jogos rapidamente.

Passo 2 (3 pontos) Agora você vai escrever um agente de busca competitiva na class `MinimaxAgent`em `multiAgents.py`. O seu agente minimax deve funcionar com qualquer número de fantasmas, então você terá que escrever um algoritmo que seja um pouco mais geral do que o que aparece no livro. Em particular, a sua árvore minimax terá múltiplas camadas min (uma para cada fantasma) para cada camada max.

Seu código deve também expandir a árvore de jogo até uma profundidade arbitrária. A utilidade das folhas da árvore minimax deve ser obtida com a função `self.evaluationFunction`, que tem como default a `scoreEvaluationFunction`. A classe `MinimaxAgent` extende a classe `MultiAgentAgent`, que dá acesso às variáveis `self.depth` (profundidade da árvore) e `self.evaluationFunction` (função de avaliação). Verifique se o seu código minimax faz referência a essas duas variáveis quando necessário já que elas são preenchidas de acordo com a linha de comando.

Importante: Uma busca de profundidade 1 considera uma jogada do Pac-Man e todas as respostas do fantasmas, profundidade 2 considera o Pac-Man e cada fantasma se movendo duas vezes e assim por diante.

Dicas e Observações

- A função de avaliação desta parte já está feita (`self.evaluationFunction`). Você não deve alterar essa função, mas reconhecer que agora estamos avaliando *estados* ao invés de ações, como fizemos para o agente reflexivo. Agentes de busca avaliam estados futuros enquanto agentes reflexivos avaliam as ações do estado atual.
- Os valores minimax do estado inicial no layout `minimaxClassic` são 9, 8, 7, -492 para profundidades 1, 2, 3 e 4 respectivamente. Note que o seu agente minimax vai vencer muitas vezes, apesar do prognóstico sombrio do minimax de profundidade 4.
  `python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4`
- Para aumentar a profundidade da busca, retire a ação `Directions.STOP` da lista de ações possíveis do Pac-Man. A busca com profundidade 2 deve ser rápida, mas com profundidade 3 ou 4 deve ser lenta.
- O Pac-Man é sempre o agente 0 e os agentes se movem em ordem crescente de índice do agente.
- Todos os estados do minimax devem ser `GameStates`, sejam passados por `getAction` ou gerados por`GameState.generateSuccessor`.
- Em tabuleiros maiores como `openClassic` e `mediumClassic` (o default), o Pac-Man será bom em evitar a morte, mas não será capaz de ganhar facilmente. Muitas vezes ele vai vagar sem ter progresso. Ele pode até vagar próximo a um ponto sem comê-lo porque ele não sabe pra onde iria depois de comer o ponto. Não se preocupe se você perceber esse comportamento, no passo 5 isso será corrigido.
- Quando Pac-Man acredita que sua morte é inevitável, ele vai tentar terminar o jogo o mais rapidamente possível por causa da penalidade constante de viver. Às vezes, esta é a coisa errada a fazer com fantasmas aleatórios, mas os agentes minimax sempre supõem o pior:
  `python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3`
  Certifique-se de entender por que o Pac-Man corre para próximo do fantasma neste caso.

Passo 3 (2 pontos) Faça um novo agente em `AlphaBetaAgent` que use a poda alfa-beta para explorar mais eficientemente a árvore minimax. Novamente, o algoritmo deve ser um pouco mais geral do que o pseudo-código no livro, então parte do desafio é estender a lógica da poda beta-alfa adequadamente ao caso de múltiplos agentes minimizadores. Você deverá ver um aumento de velocidade (a busca com poda com profundidade 3 deve rodar tão rápido quanto a busca sem poda com profundidade 2). Idealmente, a profundidade 3 em `smallClassic` deve rodar em poucos segundos por jogada ou mais rápido.

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

Os valores minimax do `AlphaBetaAgent` devem ser idênticos aos do `MinimaxAgent`, embora as ações selecionadas possam variar por causa de desempates diferentes. Novamente, os valores minimax do estado inicial no layout `minimaxClassic` são 9, 8, 7 e -492 para profundidades 1, 2, 3 e 4, respectivamente.

Passo 4 (2 pontos) Fantasmas aleatórios obviamente não são agentes minimax ótimos. Sendo assim, utilizar a busca minimax pode não ser apropriado. Implemente ExpectimaxAgent, onde o seu agente deixará de escolher o mínimo entre as possíveis ações dos fantasmas, e calculará o valor esperado supondo que os fantasmas escolhem as ações dentre as ações legais (`getLegalAction`) aleatoriamente de maneira uniforme (`RandomGhost`).

Agora você deve observar uma atitude mais cavalheiresca quando o Pac-Man se aproxima dos fantasmas. Em particular, se o Pac-Man percebe que ele pode ficar preso, mas tem a possibilidade de pegar mais algumas peças de comida, ele vai pelo menos tentar. Investigue os resultados destes dois cenários:

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

Você deve notar que o seu agente `ExpectimaxAgent` ganha metade das vezes, enquanto o`AlphaBetaAgent` sempre perde. Certifique-se de entender porque o comportamento do expectimax é diferente do minimax.

Passo 5 (1 ponto) Escreva uma função de avaliação melhor para o Pac-Man dentro da função`betterEvaluationFunction`. A função de avaliação deve avaliar os estados, ao invés de ações como a função de avaliação do agente reflexivo faz. Você pode usar todas as ferramentas à sua disposição para a avaliação, incluindo seu código do primeiro trabalho. Com a busca de profundidade 2, sua função de avaliação deve limpar o `smallClassic` layout com dois fantasmas aleatórios mais de metade das vezes e ainda executar a uma velocidade razoável (para obter crédito total, o Pac-Man deve atingir em média cerca de 1000 pontos quando estiver ganhando).

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

No relatório, inclua uma explicação sobre a sua função de avaliação.

Dicas e Observações

- Da mesma forma que na função de avaliação do agente reflexivo, você deve usar o inverso de valores importantes (como a distância para a comida) ao invés dos próprios valores
- Uma maneira de escrever sua função de avaliação é usar uma combinação linear de características. Ou seja, calcular valores para características do estado que você acha que são importantes, e então combinar as características, multiplicando os valores por pesos diferentes e somando os resultados. Você pode decidir os pesos com base na importância da característica.