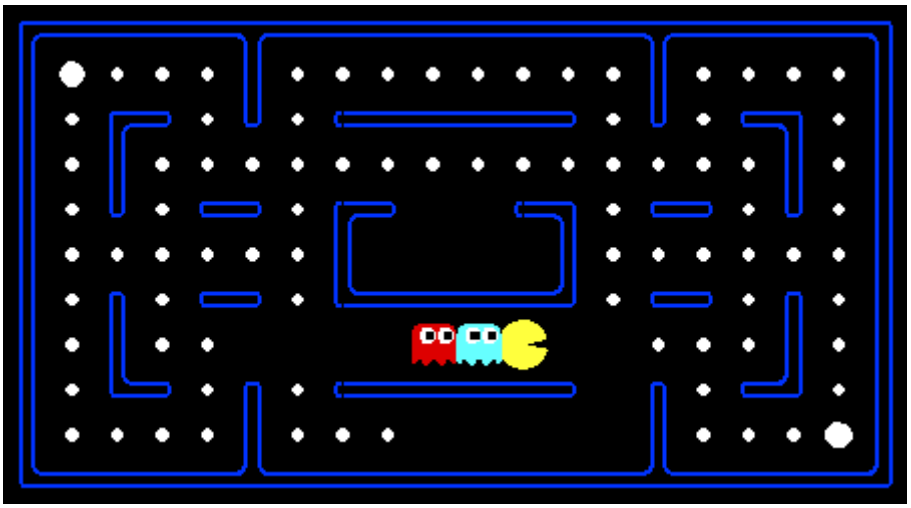
Trabalho 2: Pac-Man Multi-Agente

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



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

Introdução

Neste projeto, você projetará 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, em vez de utilizar os arquivos do primeiro trabalho.

Este projeto inclui um autoavaliador (*autograder*) para você avaliar suas respostas em sua máquina. Ele pode ser executado em todas as questões com o comando a seguir:

```
python autograder.py
```

A autoavaliação pode ser aplicada a uma questão em particular. Por exemplo, para aplicar a autoavaliação na 2ª questão, use o seguinte comando:

```
python autograder.py -q q2
```

A autoavaliação também pode ser aplicada para um teste em particular, conforme exemplo no seguinte comando:

```
python autograder.py -t test_cases/q2/0-small-tree
```

Por padrão, o autoavaliador exibe gráficos com a opção -t, mas não com a opção -q. Você pode forçar gráficos usando o sinalizador --graphics ou forçar nenhum gráfico usando o sinalizador --no-graphics.

O código para este projeto contém os seguintes arquivos, disponíveis no arquivo denominado `multiagent.zip`.

Arquivos que devem editados neste trabalho:

`multiAgents.py`	Onde todos os seus agentes de busca multiagente vão ser implementados.

Arquivos que devem ser lidos:

`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

Avaliação: Seu código será autoavaliado. Não altere os nomes das funções ou classes fornecidas dentro do código, porque isso causará problemas no autoavaliador. No entanto, a exatidão de sua implementação - não os julgamentos do autoavaliador - será o juiz final de sua pontuação. Se necessário, revisaremos e avaliaremos as tarefas individualmente para garantir que você receba o devido crédito por seu trabalho.

Arquivos para editar e enviar: Você preencherá partes de `multiAgents.py` durante a tarefa. Depois de concluir o trabalho, você enviará um token gerado por `submit autograder.py`. Não altere os outros arquivos nesta distribuição nem envie qualquer um dos arquivos originais além deste arquivo.

O que entregar: Você irá preencher o arquivo `multiAgents.py` durante o trabalho. Você deve enviar esse arquivo acompanhado de um relatório contendo os 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 MultiAgente

Primeiro, jogue o 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 desse agente (em `multiAgents.py`) e certifique-se de compreender o que ele está fazendo.

QUESTÃO 1 (4 PONTOS) – REFLEXAGENT

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 labirinto *default*, a não ser que a sua função de avaliação seja muito boa.

Notas:

- Lembre-se de que `newFood` contém o método `asList()`
- Você não pode colocar mais fantasmas do que o layout permite.
- Como características, tente o recíproco de valores importantes (tal como a distância para comida) em vez dos próprios valores.
- A função de avaliação que você está escrevendo está avaliando pares estado-ação; na próxima parte deste 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.

Avaliação: executaremos seu agente no layout openClassic 10 vezes. Você receberá 0 pontos se seu agente atingir o tempo limite ou nunca vencer. Você receberá 1 ponto se o seu agente vencer pelo menos 5 vezes, ou 2 pontos se o seu agente vencer todos os 10 jogos. Você receberá 1 ponto adicional se a pontuação média do seu agente for maior do que 500, ou 2 pontos se for maior que 1000. Você pode testar o seu agente nestas condições com:

```
python autograder.py -q q1
```

Para executar sem a parte gráfica, use:

```
python autograder.py -q q1 --no-graphics
```

QUESTÃO 2 (5 PONTOS) – MINIMAX

Agora você vai implementar um agente de busca competitiva na classe `MinimaxAgent` em `multiAgents.py`. O seu agente minimax deve funcionar com qualquer número de fantasmas. Sendo assim, 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` estende 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 dos fantasmas, profundidade 2 considera o Pac-Man e cada fantasma se movendo duas vezes, e assim por diante.

Avaliação: verificaremos seu código para determinar se ele explora o número correto de estados do jogo. Esta é a única maneira confiável de detectar alguns bugs muito sutis nas implementações do minimax. Como resultado, o autoavaliador será muito exigente sobre quantas vezes você chamar GameState.generateSuccessor. Se você chamar mais ou menos do que o necessário, o autoavaliador reclamará. Para testar e depurar seu código, execute

```
python autograder.py -q q2
```

Isso mostrará o que seu algoritmo faz em uma série de pequenas árvores, bem como em um jogo de pacman. Para executá-lo sem gráficos, use:

```
python autograder.py -q q2 --no-graphics
```

DICAS E OBSERVAÇÕES

- A implementação correta do minimax irá fazer com que o PacMan perca em alguns testes. Entretanto, isso não é um problema: já que esse é o comportamento esperado, ele irá passar nos testes.

- A função de avaliação desta parte já está feita (`self.evaluationFunction`). Você não deve alterar essa função, mas reconhecer que agora estamos avaliando *estados* em vez 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 (aproximadamente 665 do total de 1000), 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 que corresponde a `agentIndex = 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; na Questão 5, isso será corrigido.

- Quando o 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, conforme mostra o comando a seguir. Nesse comando, certifique-se de entender o porquê de o Pac-Man correr para próximo do fantasma neste caso. Explique isso no seu relatório.

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

QUESTÃO 3 (5 PONTOS)- PODA ALFA-BETA

Implemente 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 pseudocódigo no livro, então parte do desafio é estender a lógica da poda alfa-beta 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.

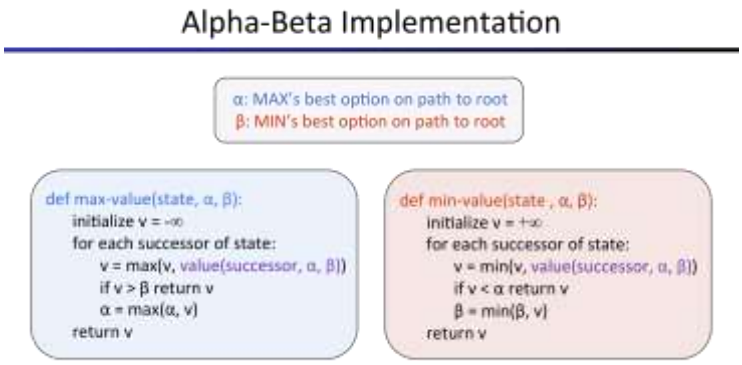
```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

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.

Avaliação: porque verificamos seu código para determinar se ele explora o número correto de estados, é importante que você execute a poda alfa-beta sem reordenar os filhos. Em outras palavras, os estados sucessores sempre devem ser processados na ordem retornada por GameState.getLegalActions. Novamente, não chame GameState.generateSuccessor mais do que o necessário.

Você não deve podar em igualdade para corresponder ao conjunto de estados explorado pelo autoavaliador. (De fato, alternativamente, mas incompatível com o autoavaliador, seria também permitir a poda na igualdade e invocar alfa-beta uma vez em cada filho do nó raiz, mas isso não corresponderá ao autoavaliador.)

O pseudo-código abaixo representa o algoritmo que você deve implementar para esta questão.



Para testar e depurar seu código, execute

```
python autograder.py -q q3
```

Isso mostrará o que seu algoritmo faz em uma série de pequenas árvores, bem como em um jogo de pacman. Para executá-lo sem gráficos, use:

```
python autograder.py -q q3 --no-graphics
```

A implementação correta da poda alfa-beta fará com que Pacman perca alguns dos testes. Isso não é um problema: como é um comportamento correto, passará nos testes.

QUESTÃO 4 (5 PONTOS) - EXPECTIMAX

Minimax e alfa-beta presumem que o agente sendo projetado está jogando contra um adversário ótimo, i.e., que toma as decisões ideais. Fantasmas aleatórios obviamente não são adversários ótimos. Sendo assim, utilizar a busca minimax pode não ser apropriado quando o Pacman está competindo com esse tipo de fantasma. Implemente o algoritmo `ExpectimaxAgent`, que é útil em situações nas quais os agentes adversários tomam decisões subótimas.

Para agilizar seu próprio desenvolvimento, fornecemos alguns casos de teste baseados em árvores genéricas. Você pode depurar sua implementação em pequenas árvores de jogo usando o comando a seguir:

```
python autograder.py -q q4
```

A depuração nesses casos de teste pequenos e gerenciáveis é recomendada e ajudará você a encontrar bugs rapidamente.

Uma vez que seu algoritmo esteja funcionando em árvores pequenas, você deve observar seu sucesso também no PacMan. Equipado com o algoritmo `ExpectimaxAgent`, 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`).

Para verificar como o PacMan se sai equipado com o algoritmo Expectimax, use:

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

Ao executar o comando acima, 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 encurralado, 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 equipado com o `ExpectimaxAgent` ganha metade das vezes, enquanto que o agente equipado com o `AlphaBetaAgent` sempre perde. Certifique-se de entender porque o comportamento do expectimax é diferente do minimax.

A implementação correta do expectimax fará com que Pacman perca alguns dos testes. Isso não é um problema: por ser um comportamento correto, ele passará nos testes.

QUESTÃO 5 (6 PONTOS) – FUNÇÃO DE AVALIAÇÃO

Escreva uma função de avaliação melhor para o Pac-Man dentro da função `betterEvaluationFunction`. A função de avaliação deve avaliar os estados, em vez 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 labirinto `smallClassic` com dois fantasmas aleatórios mais da 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
```

Avaliação: o autograder executará seu agente no layout smallClassic 10 vezes. Iremos atribuir pontos à sua função de avaliação da seguinte maneira:

- Se você ganhar pelo menos uma vez sem que o tempo limite do autoavaliador seja alcançado, você receberá 1 ponto. Qualquer agente que não satisfaça esses critérios receberá 0 pontos.
- +1 por ganhar pelo menos 5 vezes, +2 por ganhar todas as 10 vezes
- +1 para uma pontuação média de pelo menos 500, +2 para uma pontuação média de pelo menos 1000 (incluindo pontuações em jogos perdidos)
- +1 se seus jogos demoram em média menos de 30 segundos na máquina autograder, quando executados com --no-graphics.
- Os pontos adicionais para pontuação média e tempo de computação só serão concedidos se você ganhar pelo menos 5 vezes.
- Não copie nenhum arquivo do Projeto 1, pois ele não passará no autograder no Gradescope.

Você pode testar seu agente sob essas condições com

```
python autograder.py -q q5
```

Para executá-lo sem gráficos, use:

```
python autograder.py -q q5 --no-graphics
```

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) em vez 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.