Introdução

Neste projeto, você criará agentes para a versão clássica do Pacman, incluindo fantasmas. Você implementará a busca **minimax**, **poda alfa-beta** e **expectimax** e experimentará o design da função de avaliação.

O código para este projeto contém os seguintes arquivos, disponíveis como um arquivo zip.

Arquivo para sua edição:	
`multiAgents.py`	Arquivo onde todos as buscas serão implementadas.
Arquivos que você pode olhar e entender:	
`pacman.py`	O arquivo principal que roda os jogos do Pacman. Este arquivo também descreve um tipo Pacman GameState.
`game.py`	A lógica por trás de como o mundo Pacman funciona. Este arquivo descreve vários tipos de suporte como AgentState, Agent, Direction e Grid.
`util.py`	Estruturas de dados (filas, pilhas, etc) úteis para implementar algoritmos. Você não precisa usá-los, mas pode achar úteis outras funções definidas aqui.
Arquivos de suporte que você pode ignorar:	
`graphicsDisplay.py`	Gráficos para Pacman
`graphicsUtils.py`	Suporte para Pacman
`textDisplay.py`	ASCII para Pacman
`ghostAgents.py`	Agentes para controlar os Fantasmas
`layout.py`	Código para ler arquivos de layout e armazenar seu conteúdo

Arquivos para editar e enviar: você preencherá partes do multiAgents.py durante a implementação.

Seu código será corrigido inicialmente com testes e posteriormente pelo professor e assistente. Por favor, não altere os nomes de quaisquer funções ou classes fornecidas dentro do código.

Implementação do Minimax

Você implementará um algoritmo de busca adversária na classe MinimaxAgent fornecida em multiAgents.py. Seu algoritmo minimax deve trabalhar com qualquer número de fantasmas. Em particular, sua árvore minimax terá vários níveis **min** (um para cada fantasma) para cada nível **max**.

Seu código também deve expandir a árvore do jogo para uma profundidade arbitrária. Pontue as folhas de sua árvore minimax com a função self.evaluationFunction, cujo padrão é scoreEvaluationFunction. MinimaxAgent estende MultiAgentSearchAgent, que dá acesso a self.depth e self.evaluationFunction. Certifique-se que seu código minimax faça referência a essas duas variáveis quando apropriado, pois essas variáveis são preenchidas em linha de comando.

Importante: Uma única camada de busca é considerada um movimento do Pacman e todas as respostas dos fantasmas, então a busca de profundidade 2 envolverá o Pacman e cada fantasma se

movendo duas vezes. Nós iremos verificar seu código para determinar se ele explora o número correto de estados do jogo.

Dicas:

- A implementação correta do minimax fará com que Pacman perca o jogo em alguns testes. Isso não é um problema: como é um comportamento correto, ele passará nos testes.
- Pacman é sempre o agente 0, e os demais agentes são fantasmas.
- Todos os estados em minimax devem ser GameStates, enviados para getAction ou gerados via GameState.generateSuccessor.
- Em mundos grandes, você verá que o Pacman é bom em não morrer, mas muito ruim em vencer. Ele muitas vezes fica parado sem fazer progressos. Ele pode até ficar ao lado de um ponto sem comê-lo porque não sabe para onde iria depois de comer esse ponto. Não se preocupe se você vir esse comportamento.
- Quando Pacman acredita que sua morte é inevitável, ele tentará encerrar o jogo o mais rápido possível por causa da constante penalidade por viver. Às vezes, isso é a coisa errada a se fazer com fantasmas aleatórios, mas os agentes minimax sempre assumem o pior.

Para rodar o seu agente, use o seguinte comando:

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

Implementação do Alfa-Beta

Crie um novo agente que use poda alfa-beta para explorar com mais eficiência a árvore minimax, no AlphaBetaAgent. Novamente, seu algoritmo deve estender a lógica de poda alfa-beta apropriadamente para vários agentes minimizadores.

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

Para rodar o seu agente, use o seguinte comando:

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

Implementação do Expectimax

Você implementará o ExpectimaxAgent, que é útil para modelar o comportamento probabilístico de agentes que podem fazer escolhas não-ótimas.

Em particular, se Pacman perceber que está preso, mas pode escapar para pegar mais alguns pedaços de comida, ele pelo menos tentará. Investigue os resultados desses 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 ver que seu ExpectimaxAgent ganha cerca de metade das vezes, enquanto seu AlphaBetaAgent sempre perde. Entretanto, a implementação correta do expectimax fará com que o Pacman perca alguns dos testes. Isso não é um problema. Para rodar o seu agente, use o seguinte comando:

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

Implementação do uma Função de Avaliação Melhor

Escreva uma função de avaliação melhor para o pacman na função betterEvaluationFunction. A função de avaliação deve avaliar estados. Para testar a sua nova função, altere a função como seguinte:

```
def scoreEvaluationFunction(currentGameState):
    #return currentGameState.getScore()
    return betterEvaluationFunction(currentGameState)
```