Trabalho Prático 2: Busca Multiagente

Prof. Luiz Chaimowicz

2016-09-17

Sumário

Esta especificação foi adaptada dos exercícios de Pac-Man de Berkeley, originalmente criados por John DeNero e Dan Klein.

Introdução

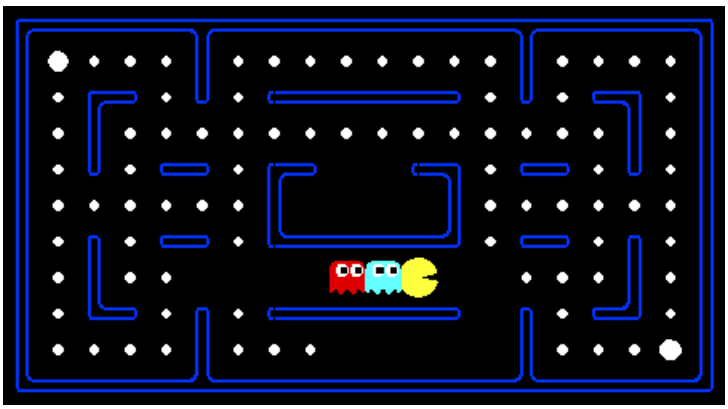

Figura 1: Pac-Man e os fantasmas. Minimax, Expectimax e avaliação

Neste projeto você criará agentes para o Pac-Man clássico, incluindo os fantasmas. Ao fazer isso você implementará tanto busca minimax quando busca expectimax, além de desenvolver uma função de avaliação.

Apesar do código-base não ser muito diferente do projeto anterior, você começará com uma nova versão do código, ao invés de editar os arquivos do projeto 1.

Assim como no projeto 1, este projeto possui um avaliador automático para que você verifique suas respostas e o seu desempenho no projeto. Como no projeto anterior, o autograder é executado com o comando:

```
python autograder.py
```

Note que alguns sistemas operacionais mais recentes podem ter apenas o Python 3 instalado. É possível verificar qual versão está instalada com o comando `python --version`. Não é incomum, quando o padrão for o Python 3, existir um binário `python2` disponível. Se esse for seu caso, substitua os comandos `python` deste documento por `python2`. A instalação do interpretador Python está fora do escopo deste documento.

Se você usa Linux ou outro Unix e o binário `python` apontar para o python3, é possível resolver o problema facilmente no shell criando um alias com:

```
alias python=python2
```

O código deste projeto consiste de vários arquivos Python, alguns dos quais você precisará ler e compreender para completar esse trabalho; outros arquivos você poderá ignorar com segurança. É possível baixar o código do moodle da disciplina.

Para avaliar uma pergunta específica, como a `q2`, use:

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

Por padrão o autograder exibe uma interface gráfica com a opção `-t`, e não a exibe com a opção `-q`. Você pode forçar a presença ou ausência de gráficos com as opções `--graphics` e `--no-graphics`, respectivamente.

Sobre o autograder

Para evitar quaisquer dúvidas, deixaremos explícito: por mais que sua nota não dependa apenas do autograder, ele é fornecido para que você possa se auto-avaiar. Se, por exemplo, o autograder disser que seu código não passou em nenhum teste, não espere que sua nota seja muito maior do que zero. Da mesma forma, se o autograder disser que você acertou todas as questões, a sua nota não será muito menor do que a maior nota possível.

Arquivos que você editará

- `multiAgents.py`: Onde todos seus agentes de busca multi-agente residirão.
- `pacman.py`: O ponto de entrada do Pac-Man. Descreve o tipo `GameState`, que você usará neste projeto.
- `game.py`: A lógica por trás do mundo. Define vários tipos de suporte, como `AgentState`, `Agent`, `Direction` e `Grid`.
- `util.py`: Estruturas de dados úteis para implementação de algoritmos de busca.

Arquivos de suporte que você pode ignorar

- `graphicsDisplay.py`: Gráficos.
- `graphicsUtils.py`: Suporte para gráficos.
- `textDisplay.py`: Gráficos ASCII.
- `ghostAgents.py`: Agentes para controle dos fantasmas.
- `keyboardAgents.py`: Interfaces com teclado para controle do Pac-Man.
- `layout.py`: Código para leitura e salvamento de arquivos de layout.
- `autograder.py`: Avaliador automático do projeto.
- `testParser.py`: Parser dos arquivos de teste e solução do avaliador automático.
- `testClasses.py`: Classes de teste.
- `test_cases/`: Diretório contendo os casos de teste de cada questão.
- `multiagentTestClasses.py`: Classes de avaliação específicas para este projeto.

Mecânica e submissão

Você irá completar partes de `multiAgents.py` durante a elaboração deste trabalho. Você deve submeter esse arquivo com seu código e comentários. Por favor *não modifique nem submeta* nenhum outro arquivo desta distribuição.

Por favor veja a seção "Submissão" para saber como submeter seu código. Projetos submetidos fora do padrão serão penalizados. Em outras palavras: use o script `prepare-submission.py`.

Avaliação

Seu código será avaliado automaticamente para verificação de corretude. Por favor *não modifique* o nome de nenhuma função ou classe dentro do código. Note, no entanto, que sua pontuação será baseada também na avaliação do seu código e não somente no julgamento do avaliador automático. Se necessário, revisaremos sua submissão individualmente para garantir que você receba o devido crédito.

Este trabalho valerá 10 pontos, com cada questão valendo 2.5 pontos cada uma.

Honestidade acadêmica

Nós verificaremos as submissões par-a-par em busca de redundância. Se você copiar o código de alguém e submetê-lo com pequenas mudanças, nós saberemos. Os detectores de plágio são difíceis de enganar. Portanto, nem tente. Não nos desaponte, pois confiamos que vocês submeterão somente o fruto de seu trabalho.

Este trabalho é individual e, portanto, compartilhar sua submissão com a de um colega não será aceito.

Obtendo ajuda

Se você tiver alguma dúvida, contate a organização do curso: temos o moodle e possibilidade de agendamento para tirar dúvidas. O objetivo deste projeto é que ele seja compensador e que você aprenda, não frustrante. No entanto, só saberemos como ajudar se você nos perguntar.

Discussão

Ao postar suas dúvidas ou discutir com colegas, tente não dar respostas diretas. O aprendizado é melhor quando entendemos o problema a ponto de elaborar respostas.

Pac-Man multiagente

Após baixar, descompactar e mudar para o diretório do código, será possível jogar um jogo de pacman com o comando

```
python pacman.py
```

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

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

Repare que ele joga mal mesmo em layouts simples:

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

Inspecione o código do `ReflexAgent` e certifique-se de *entender* o que ele faz.

Note que `pacman.py` suporta opções que podem ser especificadas de forma longa (`--layout`, por exemplo) ou curta (`-l`). É possível ver a lista de opções com:

```
python pacman.py -h
```

Nota: Caso você encontre algum erro relacionado a `Tkinter`, visite http://tkinter.unpythonic.net/wiki/How_to_install_Tkinter.

Parte 1: Minimax

Agora você escreverá um agente de busca com adversários na classe `MinimaxAgent` do arquivo `multiAgent.py`. Seu agente minimax deve funcionar com qualquer número de fantasmas. Portanto, você escreverá um algoritmo ligeiramente **mais geral** que o visto em sala. Em particular, sua árvore minimax terá várias camadas *min* (uma para cada fantasma) para cada camada *max*.

Seu código também deve ser capaz de expandir a árvore do jogo a profundidades arbitrárias. Pontue as folhas da sua árvore minimax com a função fornecida em `self.evaluationFunction` que, por padrão, resolve para `scoreEvaluationFunction`. `MinimaxAgent` extende `MultiAgentSearchAgent`, que dá acesso a `self.depth` e `self.evaluationFunction`. Certifique-se que seu código minimax referencia essas duas variáveis quando apropriado, uma vez que elas serão populadas em resposta às opções de linha de comando.

Importante: Um nível na árvore de busca é entendido como um movimento do Pac-Man e as respostas de **todos** os fantasmas. Logo, uma busca de profundidade 2 implicará em o Pac-Man e os fantasmas de movendo duas vezes.

Avaliação

Nós verificaremos seu código para determinar se ele explora o número correto de estados do jogo. Essa é a única forma confiável de detectar bugs sutis em implementações de minimax. Como resultado, o autograder será chato com o número de vezes que você chamar `GameState.generateSuccessor`. Se você chamá-lo mais vezes ou menos vezes que o necessário, o autograder reclamará.

Para testar e depurar o seu código, execute:

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

Isso exibirá o desempenho do seu algoritmo num conjunto de árvores pequenas, bem como em um jogo de pacman. Para executar sem gráficos, use:

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

Dicas e observações

- Uma implementação correta fará com que o Pac-Man perca algumas partidas nos testes. Isso não é um problema: como o comportamento será correto, ele passará nos testes.
- A função de avaliação desta parte já foi escrita (self.evaluationFunction). Você não deveria ter que mudar essa função, mas note que estamos avaliando *estados* ao invés de ações.
- Os valores minimax do estado inicial do layout minimaxClassic são 9, 8, 7, -492 para profundidades 1, 2, 3 e 4 respectivamente. Note que seu agente de minimax ganhará mais do que perderá (665/1000 jogos para nós) apesar da predição da profundidade 4.

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

- Pac-man é sempre o agente 0 e os agentes se movem em ordem crescente de índice.
- Todos os estados no minimax devem ser `GameStates`, ou passados como argumento para `getAction` ou gerados por `GameState.generateSuccessor`. Neste projeto não haverá abstração para estados simplificados.
- Em mapas maiores, como `openClassic` e `mediumClassic` (o padrão), você verá que o Pac-Man é bom em não morrer, mas ruim em ganhar. Ele comumente vagará sem progresso, inclusive ignorando comida próxima por não saber o que fazer após comê-la. Não se preocupe com esse comportamento. Na parte 5 você o resolverá.
- Quando o Pac-Man acreditar que sua morte é inevitável, ele tentará terminar o jogo tão logo quanto possível, devido à penalização constante de viver. Às vezes essa é a coisa errada a se fazer com oponentes aleatórios, mas agentes minimax sempre esperam o pior:

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

Certifique-se de entender o motivo do Pac-Man se suicidar nesse caso.

Parte 2: Poda alpha-beta

Faça agora um agente (em `AlphaBetaAgent`) que usa poda alpha-beta para explorar a árvore minimax de forma mais eficiente. Novamente o seu algoritmo será ligeiramente mais geral que o pseudocódigo visto em aula. Parte do desafio é estender a lógica da poda alpha-beta apropriadamente para múltiplos agentes.

Você deveria ver um ganho de desempenho (com alpha-beta de profundidade 3 rodando tão rápido quando minimax de profundidade 2). Idealmente, profundidade 3 em `smallClassic` deveria rodar em apenas alguns segundos por movimento, ou mais rápido.

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

Os valores encontrados por `AlphaBetaAgent` deveriam ser idênticos aos valores minimax encontrados por `MinimaxAgent`, apesar das ações selecionadas poderem varia devido ao comportamento diferente no caso de empates. Novamente, os valores minimax do estado inicial do layout `minimaxClassic` são 9, 8, 7, -492 para profundidades 1, 2, 3 e 4 respectivamente.

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

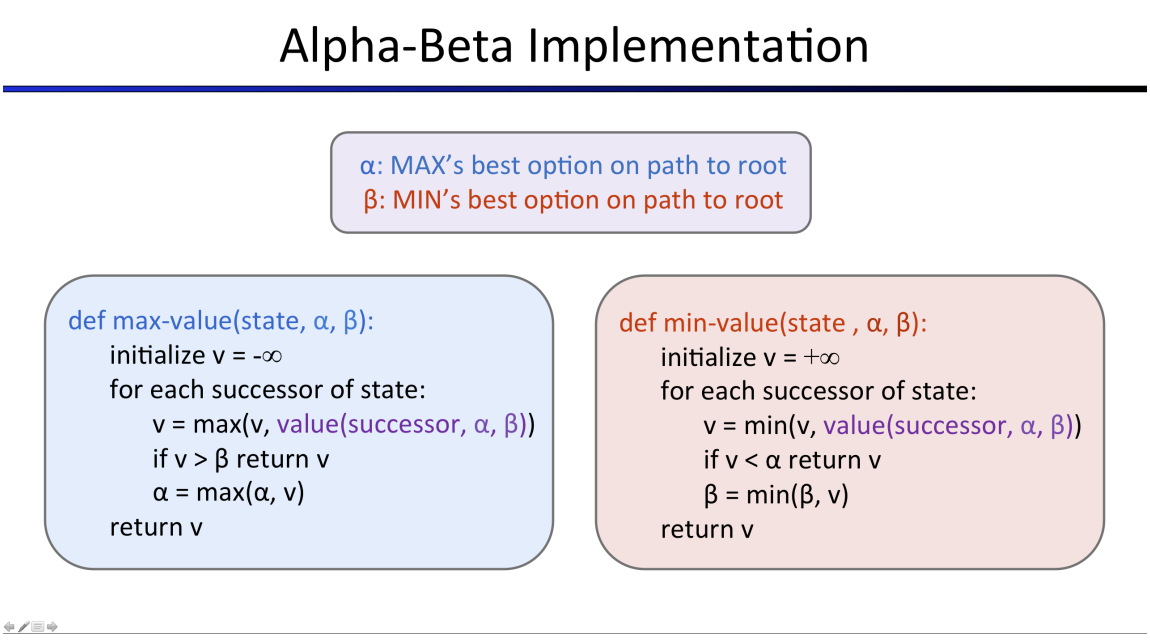

Figura 2: Implementação de poda alpha-beta

Para testar e depurar seu código, execute:

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

Isso exibirá o desempenho do seu algoritmo num conjunto de árvores pequenas, bem como em um jogo de pacman. Para executar sem gráficos, use:

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

Avaliação

Nós verificaremos seu código para determinar se ele explora o número correto de estados do jogo. É importante que você faça a poda **sem reordenar** os nós filho. Em outras palavras, estados sucessores devem ser avaliados **na mesma ordem** retornada por `GameState.getLegalActions`. Novamente, não chame `GameState.generateSuccessor` mais que o necessário.

Você não deve podar em caso de igualdade, de modo que a sua solução seja semelhante à nossa.

Uma implementação correta fará com que o Pac-Man perca em alguns testes. Isso não é um problema: como o comportamento será correto, ele passará nos testes.

Parte 3: Expectimax

Minimax e poda alfa-beta são excelentes, mas eles supõem que você está jogando contra um adversário racional que toma decisões ótimas. Como qualquer pessoa que já ganhou no jogo da velha pode dizer, esse não é sempre o caso. Nesta parte você implementará o `ExpectimaxAgent`, que é útil para modelar comportamento probabilístico de agentes que podem tomar decisões sub-ótimas.

A beleza desses algoritmos está em sua aplicabilidade geral. Para adiantar o seu desenvolvimento, nós disponibilizamos alguns casos de teste baseados em árvores genéricas. Você pode depurar sua implementação em árvores de jogos pequenas usando o comando:

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

Testar nesses casos de teste pequenos e gerenciáveis é recomendado. Certifique-se de, quando computar suas médias, usar floats. Por padrão, divisão inteira em Python trunca o resultado, de modo que $1/2 = 0$. Com floats, $1.0/2.0 = 0.5$.

Assim que seu algoritmo estiver funcionando em árvores pequenas, você poderá seu sucesso no Pac-Man. Fantasmas aleatórios não são agentes minimax ótimos e, portanto, modelá-los com busca minimax pode não ser apropriado. `ExpectimaxAgent` não mais computará `min` sobre todas as ações dos fantasmas, mas o valor esperado de acordo com o modelo de seu agente sobre o comportamento dos fantasmas. Para simplificar o código, suponha que você competirá somente contra agentes que escolhem uniformemente dentre o resultado de `getLegalActions`.

Para ver como o agente `ExpectimaxAgent` se comporta, execute:

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

Você agora deveria observar uma abordagem melhor por parte do seu Pac-Man. Em particular, se o Pac-Man perceber que pode escapar de uma emboscada para coletar mais comida, ele ao menos tentará. Investigue os resultados dos dois cenários abaixo:

```
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 deveria ver que seu agente `ExpectimaxAgent` vence aproximadamente metade das partidas, enquanto o seu agente `AlphaBetaAgent` sempre perde. Certifique-se de compreender as diferenças de comportamento.

A implementação correta de ExpectiMax levará o Pac-Man a perder em alguns dos testes. Novamente isso não é um problema: se for comportamento correto, passará nos testes.

Parte 4: Função de avaliação

Escreva `betterEvaluationFunction`, uma função de avaliação melhor que a usada atualmente. A função de avaliação deve avaliar estados, ao invés de ações. Você pode usar quaisquer ferramentas a sua disposição. Inclusive o código de busca do projeto anterior (nesse caso, **copie** o código para o módulo `multiAgents.py`). Com busca de profundidade 2, sua função de avaliação deveria ganhar no layout `smallClassic` com um fantasma aleatório mais de 50% das vezes. Para 100% de crédito o Pac-Man dever ter uma média de 1000 pontos quando estiver ganhando.

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

Avaliação

O autograder executará o seu agente no layout `smallClassic` 10 vezes. A pontuação se dará da seguinte forma:

- Se você ganhar pelo menos uma vez sem timeout no autograder, você receberá 1 ponto (desta questão). Quaisquer outros agentes receberão 0 pontos.
- +1 por vencer pelo menos 5 vezes, +2 para vencer todas as 10 vezes.
- +1 por uma pontuação média de 500, +2 para uma média de pelo menos 1000 (incluindo a pontuação em jogos perdidos)
- +1 se seus jogos levarem, em média, menos de 30 segundos na máquina em que o autograder for executado.
- Os pontos para pontuação média e tempo de computação serão dados apenas se você vencer pelo menos 5 vezes.

Dicas e observações

- Você pode querer usar o recíproco de valores importantes (como a distância até a comida), ao invés dos valores reais.
- Um método interessante para escrever sua função de avaliação é usar uma combinação linear das *features*. Ou seja, compute valores para características do estado que você considera importantes, então combine esses valores multiplicando-os por diferentes valores e somando os resultados. Você pode decidir multiplicar as *features* baseado em quão importantes você as considerar.

Submissão

Você deve submeter sua implementação pelo moodle da disciplina. Para submissão execute o script `prepare-submission.py`. Ele coletará os arquivos que devem ser submetidos e gerará um arquivo zip para submissão no moodle. Por favor não tente usar qualquer programa de compressão. Em suma: não tente ser esperto. O script foi escrito para garantir consistência na submissão.

```
python prepare-submission.py
```

O script perguntará por seu número de matrícula e gerará um arquivo cujo nome consistirá de seu número de matrícula com a extensão `.zip`. Dentro desse arquivo será adicionado o arquivo `multiAgents.py`.