Trabalho Prático

Inteligência Artificial – PUCRS

Prof. Luan Garcia

Introdução

Neste projeto, seu agente Pacman irá encontrar caminhos através de um mundo de labirintos, tanto para chegar em algum local específico quanto para coletar comida de forma eficiente. Sua tarefa será desenvolver algoritmos de busca e aplicá-los para os diversos cenários do Pacman.

Informações sobre avaliação e entrega

O trabalho deve ser realizado em grupos de 2 a 4 alunos. NÃO SERÃO ACEITOS TRABALHOS INDIVIDUAIS.

A avaliação ocorrerá em aula de laboratório, mediante apresentação e explicação do código para o professor.

A data da entrega (apresentação) está descrita no moodle.

Teremos duas aulas de laboratório específicas para implementação e apresentação do trabalho, mas também pode ser implementado fora do ambiente de aula.

A nota contará como parte da média de trabalhos da disciplina.

Atenção: Abaixo será sugerido o uso de um autograder, um conjunto de testes automáticos para avaliar a corretude de sua implementação. A nota deste trabalho considera a apresentação e explicação do código, os resultados do autograder não são necessários para avaliação e servem apenas como auxílio para implementação.

Código base

O código base do projeto que utilizaremos para nossas tarefas foi desenvolvido por uma equipe da Universidade de Berkeley, na Califórnia.

Este projeto consiste em diversos arquivos Python, alguns dos quais você precisará compreender para completar suas tarefas, e alguns que você pode ignorar. O código está disponível no moodle da disciplina.

Arquivos que você irá editar:	
search.py	Arquivo onde os algoritmos de busca devem ser implementados.
searchAgentes.py	Arquivo onde os agentes baseados em busca devem ser implementados.
Arquivos que você não deve editar, mas pode ser interessante olhar:	
pacman.py	Arquivo principal que executa o jogo. Descreve um Pacman GameState, que você irá utilizar.
game.py	A lógica de como o mundo de Pacman funciona. Descreve diversas classes que você utilizará.
util.py	Estruturas de dados para implementar os algoritmos de busca.

Os demais arquivos não são relevantes para as tarefas, apenas possuem código para lidar com gráficos e toda a execução do jogo.

O jogo do Pacman

Depois de baixar e descompactar a pasta do projeto, você pode executar o programa utilizando o seguinte comando:

```
python pacman.py
```

Este comando executa uma versão do jogo em que você controla o Pacman utilizando as setas do teclado.

O agente mais simples no arquivo searchAgentes.py é o agente GoWestAgent, que sempre se move para o oeste (ele é um agente reativo simples). Em uma configuração muito simples, ele é capaz de vencer:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

Porém, com um layout de labirinto em que é necessário fazer curvas, as coisas não funcionam tão bem:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

A execução de pacman.py permite diversas opções que podem ser expressas de forma extensa (exemplo: --layout), ou de forma reduzida (exemplo: -l). A lista com todas opções e valores padrão pode ser vista utilizando o comando help:

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

Ou também no arquivo commands.txt, dentro da pasta do projeto.

Tarefas

No arquivo **searchAgents.py**, você irá encontrar uma classe **SearchAgent** completamente implementada, que planeja um caminho através do mundo do Pacman e depois executa o caminho passo a passo. Os algoritmos de busca que irão formular um plano para este caminho não estão implementados. Implementá-los será sua tarefa.

Para testar se o **SearchAgent** está funcionando, você pode executar o comando abaixo:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

O comando acima diz para o **SearchAgent** utilizar como algoritmo de busca a função **tinyMazeSearch**, que está implementado em **search.py**. Para este labirinto, o Pacman deve conseguir navegar pelo mundo sem problemas. Note que tinyMazeSearch retorna uma lista ações!

Importante: Todas as funções de busca devem retornar uma lista de ações que levarão o agente do estado inicial até o estado objetivo. Estas ações precisam ser movimentos válidos dentro do jogo (direções válidas, não atravessar paredes etc.).

Nota: Se você quiser utilizar o autograder como forma de avaliar sua implementação, utilize as estruturas de dados presentes no arquivo **util.py**. Você pode implementar suas próprias estruturas (de acordo com cada algoritmo de busca), mas o funcionamento do autograder não é garantido neste caso.

Dica: O tabuleiro do Pacman mostra um overlay dos estados explorados e a ordem em que eles foram explorados pelo algoritmo de busca utilizado. Quanto mais forte o tom de vermelho, mais cedo o estado foi explorado. Você pode verificar se o seu algoritmo de busca está funcionando como o previsto analisando visualmente o padrão de exploração de estados.

(2 pontos) Busca em profundidade (Depth First Search)

Implemente um algoritmo de busca em profundidade na função **depthFirstSearch** dentro do arquivo **search.py**. A versão implementada deve ser da busca em grafo, ou seja, evite explorar nodos da borda que já tenham sido visitados.

Sua implementação deve funcionar para todos os layouts de labirinto abaixo:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs

python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs
```

Utilize a chamada abaixo para verificar se seu código passa em todos os testes automático do autograder:

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

(2 pontos) Busca em largura (Breadth First Search)

Implemente um algoritmo de busca em largura na função **breadthFirstSearch** dentro do arquivo **search.py**. A versão implementada deve ser da busca em grafo, ou seja, evite explorar nodos da borda que já tenham sido visitados.

Para testar sua implementação, utilize os comandos abaixo:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs
```

Utilize a chamada abaixo para verificar se seu código passa em todos os testes:

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

(2 pontos) Busca de custo uniforme (Uniform Cost Search)

A busca em largura encontrar o menor caminho até o objetivo se o custo de todas as ações for igual. Quando os custos variam, podemos querer encontrar caminhos diferentes. Por exemplo, considere um ambiente que existam fantasmas. Podemos querer forçar que o Pacman evite áreas que tenham

fantasmas tornando o custo da ação mais caro, ou que o Pacman privilegie áreas que tenham mais comidas, tornando o custo mais barato.

Implemente um algoritmo de busca de custo uniforme em grafos na função **uniformCostSearch** no arquivo **search.py**. Seu algoritmo deve ter sucesso em todos os layouts de labirinto dos comandos abaixo, onde custos de ações variam (as funções de custo já estão implementadas).

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze  -p StayWestSearchAgent
```

Utilize a chamada abaixo para verificar se seu código passa em todos os testes:

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

(4 pontos) Busca A*

Implemente um algoritmo de busca A* em grafos na função **aStarSearch** no arquivo **search.py**. Esta função requer uma função heurística como argumento. Uma função heurística possui dois argumentos: um estado no problema de busca e o próprio problema. A função **nullHeuristic** em **search.py** é um exemplo trivial implementado em que todos os custos de heurística são nulos. Já a função **manhattanHeuristic** em **searchAgents.py** é uma heurística utilizando a distância de Manhattan. Não é necessário desenvolver sua própria heurística, mas você é encorajado a desenvolvê-la por razões de aprendizado.

Para testar sua implementação de A* com a heurística da distância de Manhattan, utilize o comando abaixo:

```
python pacman.py -l bigMaze -z .5  -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

Utilize a chamada abaixo para verificar se seu código passa em todos os testes:

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