**ACH2016 Inteligência Artificial**
**Profa. Karina Valdivia Delgado**
**(texto baseado no EP traduzido por Thiago Pereira Bueno)**

# Exercício Programa 1 (EP1)
# Busca

**Prazo limite de entrega no Tidia: 25/04/2019 às 23:55**

**Composição dos grupos:** de 1-3 alunos.

## 1 Introducão

Neste exercício-programa estudaremos a abordagem de resolução de problemas através de busca no espaço de estados do jogo Pac-Man. Utilizaremos parte do material/código livremente disponível do curso UC Berkeley CS188 .[1]

Os objetivos deste exercício-programa são:

(i) compreender a abordagem de resolução de problemas baseada em busca no espaço de estados;

(ii) implementar algoritmos de busca informada e não-informada e comparar seus desempenhos.

Figura 1: Cenário 1 - Encontrando um ponto fixo de comida

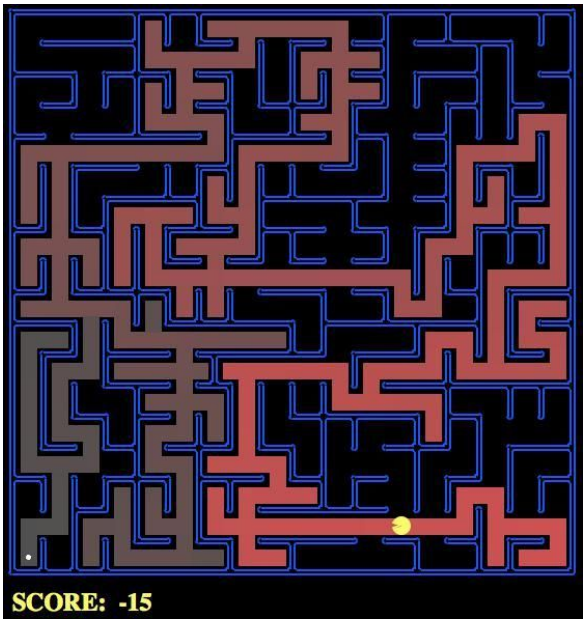
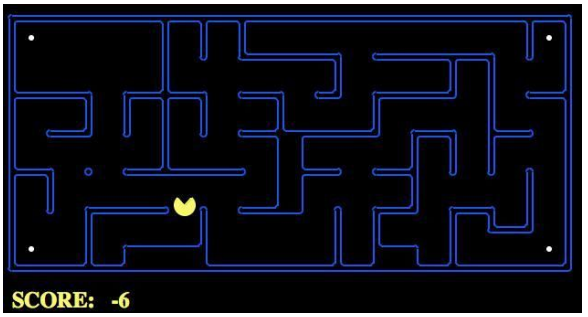



Figura 2: Cenário 2 - Encontrando os cantos do labirinto




[1] http://ai.berkeley.edu/project_overview.html

O jogo Pac-Man é um jogo eletrônico em que um jogador, representado por uma boca que se abre e fecha, deve se mover em um labirinto repleto de comida e fantasmas que o perseguem. O objetivo é comer o máximo possível de comida sem ser alcançado pelos fantasmas, em ritmo progressivo de dificuldade. Existem muitas variações do jogo Pac-Man, para esse exercício-programa consideramos alguns cenários (Figuras 1 e 2) nos quais utilizamos algoritmos de busca para guiar o Pac-Man no labirinto a fim de atingir determinadas posições e coletar comida eficientemente.

### 1.1 Instalacão

Para a realização deste EP será necessário ter instalado em sua máquina a versão 2.7 do Python. Faca o download dos arquivo ep1.zip do Tidia. Descompacte o arquivo ep1.zip e rode na raiz do diretório o seguinte comando para testar a instalação:

```
$ cd search/

$ python pacman.py
```

## 2 Pac-Man como problema de busca

Neste exercício-programa você resolverá diversos problemas de busca. Independente da busca, a interface que implementa a formulação do problema é definida pela classe abstrata e seu conjunto de metodos abaixo (disponível no arquivo search.py na pasta search/):

```
# arquivo search.py

class SearchProblem:

    def getStartState(self):

        """ Returns the start state for the search problem. """

        # ...

    def isGoalState(self, state):

        """ Returns True if and only if the state is a valid goal state. """

        # ...

    def getSuccessors(self, state):

        """        For a given state, this should return a list of triples (successor, action,
        stepCost), where successor is a successor to the current state, action is the
        action required to get there, and stepCost is the incremental cost of expanding
        to that successor. """

        # ...

    def getCostOfActions(self, actions):

        """        This method returns the total cost of a particular sequence of
        actions. The sequence must be composed of legal moves. """

        # ...
        """
```

Note que embora a formulação de busca seja sempre a mesma para cada cenário de busca, a representação de estados varia de problema a problema. Por exemplo, veja a classe PositionSearchProblem no arquivo searchAgents.py e entenda a representação de estados utilizada para esse problema. Isso será importante pois você terá que implementar outra representação de estados para outro cenário de busca (Cenário 2) em um dos exercícios.

# 3 Implementação

Os arquivos que você precisará editar:

- search/search.py onde os algoritmos de busca serão implementados;
- search/searchAgents.py onde os agentes baseados em busca serão implementados.

Os arquivos que você precisará ler e entender:

- pacman.py arquivo principal para executar o jogo.
- util.py estruturas de dados para auxiliar a codificação dos algoritmos de busca.

Observação: Utilize as estruturas de dados Stack, Queue, PriorityQueue e PriorityQueueWithFunction disponíveis no arquivo util.py para implementação da fronteira de busca. Essas implementações são necessárias para manter a compatibilidade com o arquivo de testes (autograder.py). Listas, tuplas, tuplas nomeadas, conjuntos e dicionários do Python podem ser utilizados sem problemas caso necessário.

# 4 Parte prática

Você deverá implementar algumas funções nos arquivos search.py e searchAgents.py.

Observação: Não esqueça de remover o código util.raiseNotDefined() ao final de cada função.

## 4.1 Cenário 1 - Encontrando um ponto fixo de comida

**Codigo 1 - Busca em Profundidade (DFS)**

Implemente a busca em profundidade (DFS) no arquivo search.py na função depthFirstSearch. Evita a expansão de estados previamente visitados. Para testar seu algoritmo rode os comandos:

```
$ python pacman.py -l tinyMaze -p SearchAgent

$ python pacman.py -l mediumMaze -p SearchAgent

$ python pacman.py -l bigMaze -z .5 -p SearchAgent
```

Observação: note que o tabuleiro do jogo mostra os estados visitados pela busca por cor, isto é, quanto mais vermelho escuro mais cedo na busca o estado foi visitado. Além disso, para todas essas buscas seu algoritmo DFS deve encontrar uma solução rapidamente, isto e, se demorar mais que alguns milissegundos algo provavelmente está errado no seu código!

**Codigo 2 - Busca em Largura (BFS)**

Implemente a busca em largura (BFS) no arquivo search.py na função breadthFirstSearch. Nova-mente, evite estados repetidos. Para testar seu algoritmo rode os comandos:

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

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

Observação: se o Pac-Man se mover muito devagar tente usar a opção --frameTime 0 na linha de comando.

**Codigo 3 - Busca de custo uniforme**

Implemente a busca de custo uniforme (UCS) no arquivo search.py na função uniformCostSearch.

Para testar seu algoritmo rode o comando:

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

**Codigo 4 - Busca heurística**

Implemente busca em grafo A* no arquivo search.py na função aStarSearch. Para testar seu algoritmo rode o comando:

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

## 4.2 Cenário 2 - Encontrando os cantos do labirinto

**Codigo 5 - Formulação de problema de busca dos 4 cantos**

Nesse exercício vamos implementar um novo problema de busca. Você deverá completar a implementação dos métodos da classe CornersProblem no arquivo searchAgents.py. Você deverá escolher uma representação de estados que codifique somente a informação necessária para detectar se todos os 4 cantos do labirinto foram visitados. Para testar a formulação do problema rode o comando:

```
$  python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

$  python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

Observação: Não utilize um objeto GameState como estado de busca! Se utilizar, seu código provavelmente ficará errado e muito lento.

**Codigo 6 - Heurística para o problema dos 4 cantos**

Além da nova formulação de problema, você deverá implementar uma heurística não trivial e consistente na função cornersHeuristic. Para testar sua heurística rode o comando:

```
$ python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
```

Nesse exercício consideramos heurísticas triviais aquelas que retornam 0 para todos os estados ou calculam o valor real do custo (por meio de buscas sistemáticas). O primeiro tipo de heurística trivial não ajuda muito do ponto de vista da eficiência do algoritmo e o último tipo demora muito mais tempo que o ideal.

**Relatorio**

Após o desenvolvimento da parte prática, você deverá testar seus algoritmos e redigir um relatório claro e sucinto (máximo de 4 páginas). Assim, você deverá:

(a) compilar em tabelas as estatísticas das buscas implementadas em termos de nós expandidos e tamanho de plano;

(b) discutir os méritos e desvantagens de cada método, no contexto dos dados obtidos;

(c) responder as questões teóricas;

### 5.1 Questões

**Questão 1 - Busca em Profundidade**
A ordem de exploração do espaço de estados seguiu conforme esperado? O Pac-Man de fato se move para todos os estados explorados em sua deliberação para encontrar uma solução para a meta? A sua implementação de busca em profundidade encontra uma solução de custo mínimo? Por quê?

**Questão 2 - Busca em Largura**
A sua implementação de busca em largura encontra uma solução de custo mínimo? Por quê?

**Questão 3 - Busca de Custo Uniforme**
Execute os comandos abaixo e explique sucintamente o comportamento dos agentes StayEastSearchAgent e StayWestSearchAgent em termos da função custo utilizada por cada agente.

```
$ python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

$ python pacman.py -l mediumDottedMaze -p StayWestSearchAgent
```

**Questão 4 - Busca Heurística**
Você deve ter percebido que o algoritmo A* encontra uma solução mais rapidamente que outras buscas. Por quê? Qual a razão para se implementar uma heurística consistente para sua implementação da busca A*?

# 6 Entrega

Entregar um arquivo EP1-NUSP-NOME-COMPLETO.zip contendo APENAS os arquivos:

(1) search.py e searchAgents.py com as implementações da parte prática.
(2) Relatório de no máximo 4 páginas em formato PDF com: as estatísticas das buscas implementadas, discussão dos méritos e desvantagens de cada método e as resposta às questões.

Não esqueça de identificar cada arquivo com seu nome e número USP. No código coloque um cabeçalho em forma de comentário.

# 7 Criterio de avaliação

A avaliação da parte prática depende dos resultados dos testes automatizados do autograder.py. Dessa forma você terá como avaliar por si a nota da parte prática que receberá para esse EP. Para rodar os testes automatizados, execute o seguinte comando:

```
$ python autograder.py
```

Com relação ao relatório, avaliaremos principalmente sua forma de interpretar comparativamente os desempenhos de cada busca. Não é necessário detalhar a estratégia de cada busca ou qual estrutura de dados você utilizou para cada busca, mas deve ficar claro que você compreendeu os resultados obtidos conforme esperado dadas às características teóricas de cada busca.

Parte pratica (18 pontos)
Cenario 1: autograder (12 pontos)
Cenario 2: autograder (6 pontos)

Relatório (12 pontos)
Questões: (8 pontos)
Comparação e discussão: (4 pontos)