PCS3848 - Inteligência Artificial - Primeiro Exercício Programa
Implementação do jogo Pacman utilizando algoritmo de busca A*

Monitora: Luciana Marques
luciana.marques@usp.br

2 de Setembro de 2019

1 Introdução

Este é o primeiro exercício de programação que fará parte da avaliação da disciplina de Inteligência Artificial (PCS3848).

O objetivo deste exercício é a fixação dos conceitos envolvendo **busca**, em especial o algoritmo de busca A*. Para solucioná-lo, certifique-se que entendeu os conceitos teóricos, e também reserve um tempo para ambientar-se com a estrutura do código fonte.

2 Apresentação do Exercicio

Este exercício foi baseado no **Pacman Project** da UC Berkley. Por ora, vamos nos concentrar na parte de **busca**. O código fonte será fornecido ao aluno juntamente com este enunciado.

Utilizaremos um tabuleiro igual do jogo clássico arcade Pacman. O objetivo é auxiliar o pacman a encontrar a comida navegando pelo tabuleiro, fazendo uso de uma implementação do algoritmo de busca de A* visto em aula.

Tanto o código já implementado quanto aquele que o estudante deverá implementar será em python. Para tal, é necessário desenvolver sua solução em um computador com python instalado, preferencialmente em um sistema UNIX.

2.1 Estrutura do código fonte

O código fornecido tem implementados diversos módulos que compõem o jogo. É interessante tentar entender como eles se relacionam e qual é sua função dentro do sistema, porém não é necessário compreender tudo completamente, uma vez que o exercício exigirá a edição de apenas um arquivo.

Para entender um pouco melhor sobre o que o programa faz antes de começar a fazer edições, você pode seguir as instruções na seção **Welcome to Pacman** na página da UC Berkley. Aqui destacaremos algumas delas.

A forma mais básica de se rodar o código fornecido é executando seu arquivo principal. Isto deve gerar um tabuleiro pequeno onde o pacman consegue alcançar a comida sozinho bem rapidamente.

```
$ python pacman.py
```

Você pode entender melhor sobre os parâmetros que o programa aceita mandando uma flag de 'help':

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

Isto irá gerar uma lista com todos os parâmetros aceitos. É interessante notar que o comportamento do pacman é determinado de acordo com o que é passado no argumento '-p'. Além disso, o código pode ser testado usando mais de um tipo de tabuleiro, este definido no argumento '-l'.

```
Options:
-h, --help            show this help message and exit
-n GAMES, --numGames=GAMES
                      the number of GAMES to play [Default: 1]
-l LAYOUT_FILE, --layout=LAYOUT_FILE
                      the LAYOUT_FILE from which to load the map layout
                      [Default: mediumClassic]
-p TYPE, --pacman=TYPE
                      the agent TYPE in the pacmanAgents module to use
                      [Default: KeyboardAgent]
-t, --textGraphics    Display output as text only
-q, --quietTextGraphics
                      Generate minimal output and no graphics
-g TYPE, --ghosts=TYPE
                      the ghost agent TYPE in the ghostAgents module to use
                      [Default: RandomGhost]
-k NUMGHOSTS, --numghosts=NUMGHOSTS
                      The maximum number of ghosts to use [Default: 4]
-z ZOOM, --zoom=ZOOM  Zoom the size of the graphics window [Default: 1.0]
-f, --fixRandomSeed   Fixes the random seed to always play the same game
-r, --recordActions   Writes game histories to a file (named by the time
                      they were played)
--replay=GAMETOREPLAY
                      A recorded game file (pickle) to replay
-a AGENTARGS, --agentArgs=AGENTARGS
                      Comma separated values sent to agent. e.g.
                      "opt1=val1,opt2,opt3=val3"
-x NUMTRAINING, --numTraining=NUMTRAINING
                      How many episodes are training (suppresses output)
                      [Default: 0]
--frameTime=FRAMETIME
                      Time to delay between frames; <0 means keyboard
                      [Default: 0.1]
-c, --catchExceptions
                      Turns on exception handling and timeouts during games
--timeout=TIMEOUT     Maximum length of time an agent can spend computing in
                      a single game [Default: 30]
```

2.2 Arquivos a serem editados

Você deve editar apenas o arquivo `search.py`, mais especificamente a função `aStarSearch()`. Você pode criar funções extras se assim quiser ou julgar necessário, mas certifique-se de não modificar outras partes do código.

```
def aStarSearch(problem, heuristic=nullHeuristic):
  """Search the node that has the lowest combined cost and heuristic first."""
  "*** YOUR CODE HERE ***"
  util.raiseNotDefined()
```

É importante que se entenda o que são os argumentos da função. A variável `problem`, no caso, é um objeto da classe `SearchProblem`. Ela está descrita abstratamente no início do arquivo `search.py`, mas você pode consultar o arquivo `searchAgents.py` caso queira ter um entendimento mais profundo.

Você pode utilizar as estruturas de dados que quiser para a implementação do algoritmo utilizando

bibliotecas do python. Existem, porém, implementações de estruturas de dados no arquivo `util.py`. É mais recomendável que se utilize estas estruturas.

Neste mesmo arquivo, há funções em que podem ser implementados outros algoritmos de busca. Você pode implementá-los e os testar se assim quiser para fins de curiosidade e de estudo, mas isto não será levado em conta na atribuição da nota.

3 Entrega da solução

A entrega da solução devera ser feita via moodle (edisciplinas). Na entrega, deverão ser anexados o código fonte do trabalho juntamente com um relatório. O código fonte deve também ser entregue por meio de um link para repositório no **github**.

O código fonte deverá seguir as instrucoes passadas neste enunciado. Não modifique outros arquivos além daquele indicado: `search.py`. Siga as boas práticas de programação de nomeação de variáveis, comentários e tente fazer o código o mais legível o possível.

Você pode fazer funções separadas, mas não mexa no funcionamento do restante do código fonte para não haver problemas de execução.

O relatório deverá conter:

1. **Nome e número USP** dos integrantes do grupo;
2. **E-mail** para contato;
3. **Descrição do código**: além de fazer um código legível, comente aqui sobre as estruturas usadas e justifique decisões tomadas;
4. **Link para o código**: colocar bem claro o link para seu código no github;
5. **Descrição das rotinas de heurística testadas**, e uma comparação entre elas por desempenho. Escolha qual a melhor heurística encontrada e **justifique**.

No caso do código fonte não funcionar, será necessário contactar os participantes. Por isto é imprescindível que o contato dos alunos esteja presente no relatório. Caso esta informação não esteja presente, haverá penalização de nota.

3.1 Data de entrega

A entrega do exercicio será até o dia **20 de Setembro (quarta-feira), até 23:55**. A solução com atraso poderá ainda ser submetida após este horário até as 10:00 do dia seguinte, sujeito a penalização de nota com base no tempo de atraso.

4 Critérios de Avaliação

A avaliação das soluções entregues seguirá a seguinte ponderação:

$$\text{Nota final} = 0.8 * (\text{exercício} + \text{relatório}) + 0.2 * (\text{desempenho}) \quad (1)$$

Ou seja, uma parte da nota será atribuída conforme o desempenho da solução obtida, o qual é medido pelo tempo que seu algoritmo demorou para comer a única comida que estará no tabuleiro. Certifique-se de procurar a melhor solução possível para garantir uma boa nota nesta parte. Todos os grupos serão submetidos, na avaliação, ao mesmo tabuleiro inicial.

Os códigos e relatórios serão avaliados conforme:

1. Código limpo, texto e código claros, sucintos, completos em seu entendimento/explicação;
2. Clareza e boa fundamentação teórica nas explicações;
3. Corretude.