Welcome to Pacman

After downloading the code (search_and_games.zip), unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line:

```
python pacman.py
```

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman's first step in mastering his domain.

The simplest agent in searchAgents.py is called the GoWestAgent, which always goes West (a trivial reflex agent). This agent can win:

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

But, things get ugly for this agent when turning is required:

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

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal.

Note that pacman.py supports a number of options that can each be expressed in a long way (e.g., --layout) or a short way (e.g., -l). You can see the list of all options and their default values via:

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

Also, all of the commands that appear in this portion of the project also appear in commands.txt, for easy copying and pasting.

Problem 1: Breadth-First Search

In the breadthFirstSearch function in search.py, implement the breadth-first graph search algorithm (states/locations that are already visited will not be visited again), for Pacman to find the path to the dot in the maze. You will probably want to make use of the Node class in search.py.

Experiments: Test your code using:

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

python pacman.py -l smallMaze -p SearchAgent -a fn=breadthFirstSearch

python pacman.py -l mediumMaze -p SearchAgent -a fn=breadthFirstSearch -z .5 --frameTime 0

python pacman.py -l bigMaze -p SearchAgent -a fn=breadthFirstSearch -z .5 --frameTime 0
```

Problem 2: A* Search

Implement the A* graph search in the empty function aStarSearch in search.py, for Pacman to find the path to the dot in the maze. You will probably want to make use of the Node class in search.py and the PriorityQueue class in util.py.

You would need a heuristic function. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example.

In your code, you can use the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

Run the following experiments to test your code.

Experiments:

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

python pacman.py -l smallMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

python pacman.py -l mediumMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
```

A few additional notes:

- If Pacman moves too slowly for you, try the option --frameTime 0.
- All of your search functions need to return a list of *actions* that will lead the agent from the start to the goal.