Homework 1: A Pacman Agent

due Monday, September 30 at 1:35 PM

In this assignment, you will be building agents that solve mazes, using the framework of the classic arcade game Pacman. The goal of this assignment is to implement different search algorithms and see how they work in practice, as well as to gain familiarity with how to do object-oriented programming in Python and work with larger Python projects

Getting Started

We will be building on a Python version of Pacman written by John DeNero and Dan Klein at UCal Berkeley. This can be found in the pacman.zip file uploaded with this assignment. Unzip the file and navigate into the pacman directory in your commandline. Then run:

```
python pacman.py
```

This should pull up a Pacman game that you can play using your arrow keys.

By adding extra options in the command line, you can also run the game on different layouts. Different layouts will allow you (or an agent!) to play the game on different grids (and possibly without ghosts). To run the game on another layout, you will need to run the program using the flag `--layout` followed by the name of the layout. For example, you can try running the game on tinyMaze by running:

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

You can find a full list of possible layouts in the layout directory. For this assignment, we will be focusing on the maze layouts. In particular, I highly recommend testing your code on the tinyMaze layout.

Additionally, you can use the flags `-p` and `-a` to run the program with an agent controlling Pacman. Currently, one search agent, which carries out a pre-determined series of actions, is available. You can run it using the following command.

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

The Code

While the pacman program contains many files, you do not need to edit or read all of them. In particular, you do **not need to edit any files other than search.py.** (Other files may have sections labeled “your code here”, but these sections implement other agents that are **not** a part of this assignment.)

In addition to search.py, you will want to read through the SearchAgent and PositionSearchProblem classes in searchAgents.py (but you do not need to edit this file).

Additionally, you will want to use utils.py, which implements the data structures you need to implement search. You may also want to read through portions of pacman.py (which provides different flags you can use when running the program) and game.py (which gives some basic information about how the program encodes game states.)

Search Algorithms

Pseudocode for searching is as follows:

```
function Search(problem) returns a list of actions
  initialize the frontier using the initial state of the problem
  initialize explored to empty
  while the frontier is not empty:
    choose the next node from the frontier
    if the node contains a goal state:
      return list of actions from start state to goal state (solution found)
    add the state key to the explored dictionary
    for each successor of the node state:
      if the successor state is not in explored:
        add node of the successor onto the frontier
  return an empty list (no solution was found)
```

Your goal is to turn this pseudocode into actual code in search.py. Note that the bolded line is the part that will change between different search algorithms.

There are a few things you will need to do to in order to implement search. First, you will need to create a Node class. **A Node object contains the state, parent node, action leading to the state, step cost, and path cost.** In order to keep track of all of these, you will need to use the methods of PositionSearchProblem (which is in searchagents.py). You may also want to write a method to get all successors of a node. When writing and testing your search functions, you must pay attention to the difference between nodes and states. The state of a searchAgent is its position in the maze, whereas a node (which contains a state) is used to structure the search.

In addition to the Node class, you will need a way to structure you search. As mentioned earlier, you will likely want to use the classes in util.py to store your Nodes in a stack, queue, or priority queue. Additionally, you will need to create a data structure that tracks all states you have already visited (called explored in the pseudocode above). I recommend creating a dictionary, where each visited state is a key whose value is the cost of the best found path to that state.

You will need to implement the following three types of search:

Depth First Search

Start by implementing depth-first search, based on the pseudocode above. Once you have filled in the depthFirstSearch function, you can run your code using the following command:

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

Breadth First Search

Now, implement breadth-first search. Note that your code should be very similar to your depth-first search function, but the data structure you use to manage the frontier will be different.

A* Search

Finally, implement A* search. As this is the only informed search you are implementing, you will need to use a heuristic function. A null heuristic, which always returns 0, is provided in search.py. Two additional heuristics, manhattanHeuristic and euclideanHeuristic, are provided in searchAgents.py. Use one of these heuristics (not the nullHeuristic) or create your own. Be sure to comment in your code which heuristic you used and what that heuristic calculates.

Wrapping Up

Run each of your searches on three different layouts: tinyMaze, mediumMaze, and bigMaze. For each search on each layout, report the number of nodes that were expanded, the cost of the path that was returned, and whether or not the path found was optimal. (You may need to write extra code to keep track of these numbers.) When reporting results for A* search, additionally report which heuristic you used.

Finally, briefly (in one to two sentences) discuss which of these search algorithms worked the best for this maze solving problem and why you believe it worked best.

Turn in your answers in a Word, PDF, or text document. Submit this document, along with your search.py file. (Do not submit any other files.)