Trần Nguyễn Khánh Duy
ITDSIU18049

IT097IU: Intro to Artificial Intelligence
Lab#3/Assignment#3
Informed Search in Pac-Man

There are three exercises in this lab:

1. Best-first search
2. A* Search
3. Solving the Corners Problem

Introduction

In this assignment, your Pac-Man agent will find paths through its maze world to reach a particular location. You will build general search algorithms and apply them to many different Pac-Man scenarios.

Files you'll edit:

`search.py`	Where all of your search algorithms will reside.
`searchAgents.py`	Where all your search-based agents will reside. [ONLY for Ex: 3]

Files you should look at but NOT edit:

`util.py`	Useful data structures for implementing search algorithms.
`pacman.py`	The main file that runs Pac-Man games. This file describes a Pac-Man `GameState` type, which you use in this lab.
`game.py`	The logic behind how the Pac-Man world works. This file describes several supporting types like `AgentState`, `Agent`, `Direction`, and `Grid`.

Finding a fixed food dot using Informed Search

Exercise 1: Implement the Best-First Search (BFS) algorithm in the `bestFirstSearch` function in `search.py`. Test your code the same way you did for other search algorithms.

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=befs
python pacman.py -l mediumMaze -p SearchAgent -a fn=befs
python pacman.py -l bigMaze -p SearchAgent -a fn=befs -z .5
```

Does BFS find a least cost solution? How many nodes are expanded?

Exercise 2: Implement the A* Search algorithm in the `aStarSearch` function in `search.py`. Use the same algorithm as shown in your text (or class). `aStarSearch` function takes an optional heuristic function as an argument. The heuristic function itself takes two arguments (a state in the search problem, and the problem itself). `search.py` provides a `nullHeuristic` function that you can look at. Also, in the `searchAgents.py` a Manhattan heuristic as well as Euclidian heuristic function is defined. Test your code the same way you did for other search algorithms.

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=astar
python pacman.py -l mediumMaze -p SearchAgent -a fn=astar
python pacman.py -l bigMaze -p SearchAgent -a fn=astar -z .5
```

To specify a heuristic function from `searchAgents.py`, use the following:

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

The function `manhattanHeuristic()` is already written in `searchAgents.py` Alternately you could write your own in `search.py`

Exercise 3a: Finding All the Corners

The real power of A* will only be apparent with a more challenging search problem. Now, it's time to formulate a new problem and design a heuristic for it.

In *corner mazes*, there are four dots, one in each corner. Our new search problem is to find the shortest path through the maze that touches all four corners (whether the maze actually has food there or not). Note that for some mazes like `tinyCorners`, the shortest path does not always go to the closest food first! **Hint**: the shortest path through `tinyCorners` takes 28 steps.

Note: Make sure to have breadth-first search working correctly before working on this exercise, because it builds on it.

Implement the `CornersProblem` search problem in `searchAgents.py`. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve:

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

You will need to define an abstract state representation that *does not* encode irrelevant information (like the position of ghosts, where extra food is, etc.). In particular, do not use a Pacman `GameState` as a search state. Your code will be very, very slow if you do (and also wrong).

Hint: The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners.

Exercise 3b: Corners Problem: Heuristic

Note: Make sure you have a correct running version of A* before working on this.

Implement a non-trivial, consistent heuristic for the `CornersProblem` in `cornersHeuristic`.

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

Note: `AStarCornersAgent` is a shortcut for

```
python pacman.py -l mediumCorners -p SearchAgent –a
fn=astar,prob=CornersProblem,heuristic=cornersHeuristic
```

Admissibility vs. Consistency: Remember, heuristics are just functions that take search states and return numbers that estimate the cost to a nearest goal. More effective heuristics will return values closer to the actual goal costs. To be *admissible*, the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal (and non-negative). To be *consistent*, it must additionally hold that if an action has cost *c*, then taking that action can only cause a drop in heuristic of at most *c*.

Remember that admissibility isn't enough to guarantee correctness in graph search -- you need the stronger condition of consistency. However, admissible heuristics are usually also consistent, especially if they are derived from problem relaxations. It therefore is usually easiest to start out by brainstorming admissible heuristics. Once you have an admissible heuristic that works well, you can check whether it is indeed consistent, too. The only way to guarantee consistency is with a proof. However, inconsistency can often be detected by verifying that for each node you expand, its successor nodes are equal or higher in in f-value. Moreover, if BFS and A* ever return paths of different lengths, your heuristic is inconsistent. This stuff is tricky!

What to submit

1. Fill out the table below:

	Best First Search			A* Search		
Maze	**#nodes expanded**	**Solution length**	**Is it optimal?**	**#nodes expanded**	**Solution length**	**Is it optimal?**
tiny	15	8	No	14	8	Yes
medium	269	68	No	221	68	Yes
big	620	210	No	549	210	Yes

2. What happens on `openMaze` for the various search strategies?
3. For each exercise where a heuristic is used, clearly show/mention the heuristic function.
4. Based on the above, a short discussion/reflection of how the searches compare to each other and to the uninformed searches from Assignment#2.
5. Source code + README (how to compile and run your code). This should include your code for the search node, Best-First Search, and A* (and Corners Search Agent, if you did Exercise 3).
6. Please create a folder called "yourname_IT097IU_Lab3" that includes all the required files and generate a zip file called "yourname_IT097IU_Lab3.zip".
7. Please submit your work (.zip) to Blackboard.

2. We will find that DFS will take too long on openMaze since it only does path-checking. If we use full cycle to check with DFS, it will find a solution (a very unusual one!)

4. Although the above search algorithms perform in different approaches, the output solutions are the same. Maybe it's because these algorithms are all suitable to work in this game.