CS440 – Creative Contest1: Multi-Agent Pacman

Purpose: In this project, you will apply the search algorithms and problems implemented in the search module to handle more challenging scenarios that include controlling multiple pacman agents and planning under time constraints. There is room to bring your own unique ideas, and there is no single set solution. I look forward to seeing what you come up with!

Getting Started: Follow these steps to get started.

1. Download the contest1.zip files.
2. Copy your `search.py` from lab1 into the contest1 directory (replacing the blank `search.py`).
3. Go into `myAgents.py` and fill out `findPathToClosestDot` in `ClosestDotAgent`, and `isGoalState` in `AnyFoodSearchProblem`. You should be able to copy your solutions from lab1 over.
4. Run `pacman.py` and you should be able to see 4 pacman agents travelling around the world collecting dots.

Objective: Your code will support more than one Pacman agent and there will be cost associated with how long Pacman "thinks" (compute cost) so you need to focus on efficiency as well. All your work must be contained in `myAgents.py`.

There are a variety of layouts contained in the `layouts` directory. Your agents will be exposed to a variety of maps of different sizes and amounts of food. Your code will be scored with +10 for each food pellet eaten and +500 for collecting all food pellets. Additionally there will be a -0.4 penalty for each action taken and a penalty of -1*total compute used to calculate next action (in seconds)*1000. Each agent starts with 100 points. You win if you collect all food pellets and your score is the current amount of points. You lose if your score reaches zero. This can be caused by not finding pellets quickly enough, or spending too much time on compute. Your score for this game is zero. If your agent crashes, it automatically receives a score of zero.

Each agent can see the entire state of the game, such as food pellet locations, all pacman locations, etc. The `GameState` in `pacman.py` now supports multiple Pacman agents and many of the `GameState` methods now have an extra argment, `agentIndex`, which is to identify which Pacman agent it needs. For example, `state.getPacmanPosition(0)` will get the position of the first pacman agent.

To get started designing your own agent, you will want to subclass the `Agent` class. This provides an important variable, `self.index` which is the `agentIndex` of the current agent. This can be used when deciding on actions.

The autograder will call the `createAgents` function to create your team of pacmen. By default, it is set to create N identical pacmen but you may modify the code to return a diverse team of multiple types of agents if you desire. By default, the game runs with `ClosestDotAgent`. To run your own agent, change `agent` for the `createAgents` method in `myAgents.py`.

Testing: You can run your agent on a single test:

```
pacman.py -l test1.lay
```

There are no hidden tests. You will also want to check on how you perform on all the layouts by using the autograder.

Grading: You will receive the full 10pts for this project by submitting working agents that score at least 700 points when running the autograder on my laptop. Additionally, this is a contest so the top scoring agent will receive a bonus of 3 extra credit points. The second place agent will receive a bonus of 2 extra credit points. The third place agent will receive a bonus of 1 extra credit point.

Submit your `myAgent.py` in Canvas for grading.