6CCS3AIN/7CCSMAIN (Artificial Intelligence)

Practical 03: Pacman starts to pull things together

(Version 1)

1 Overview

In the previous practical, Pacman started to deal with a *partially observable* world. This week observability is even more limited. The good news is that this is the last update to observability that you will have to contend with before the first coursework, so there will be no new things to contend with. The other good news is that this update to the API should not need you to modify any of your code — it should be possible to just add the new API and run the same code as before. (Of course, it may no longer work so well.)

2 Less observability

The newest version of `api.py`, version 3, only returns information on food and capsules when they are in Pacman's line of sight down a corridor. When Pacman is moving, they can only see straight ahead until five steps ahead or the nearest wall. If the are passing a corridor, then they can see one step into that corridor. If Pacman is not moving, then they can see down any corridor they are on in any direction. Again this view continues to the five step limit from last week, or to a wall if that is closer.

The same limitation applies to ghosts. However, Pacman can *always* detect a ghost if it is two steps or less away. (The idea is that Pacman can hear ghosts when they get close, and two steps isteh minimum necessary to be able to run away safely.)

You should:

1. Download a copy of version 3 of `api.py` from KEATS. This can be found under the Practical for Week 4.

2. Add version 3 of `api.py` to your `pacman` folder.

   I would make a another copy of `pacman` first so that I kept the materials from Practical 01 and Practical 02 separate from Practical 02.

3. Run `SensingAgent`

   ```
   python pacman.py --pacman SensingAgent
   ```

4. Notice that when it starts up,you get output like:

   ```
   Legal moves:  ['West', 'Stop', 'East']
   Pacman position:  (9, 1)
   ```

```
Ghost positions:
Distance to ghosts:
Capsule locations:
[]
Food locations:
[(6, 1), (7, 1), (8, 1), (10, 1), (11, 1), (12, 1), (13, 1)]
```

This is the exact same `SensingAgent` as the last couple of weeks, it is just getting less data than before — it only sees the food in the corridor that it is in. It sees both ways along the corridor because it is initially stationary.

If you run and agent that moves and have it print out food locations you

3 Check your previous code works

The new version of `api.py` should work fine with your previous code — all the same API functions are there, so there should be no runtime errors that are the fault of the new API.

However, since the information that Pacman now has is reduced, you may find that your code no longer controls Pacman as well as it used to. The first thing you should do is to decide whether your previous Pacman controllers still work as well as they did, or whether you need to fix them.

4 Forage safely

Part of the requirement for the coursework is to develop a Pacman that can cope with ghosts while clearing the food and so winning a game. If you haven't yet tried to combine these two aspects in one agent, now is the time to try that. That is, try combining ideas from `CornerSeekingAgent` from Practical 2 and `HungryAgent` from Practical 1.

One way to do this is to think of your agent having two states:

- a "happy forager" state, when it is looking for food; and
- a "worried survivor" state, when it is trying to stay away from the ghost.

When the agent is in the "happy forager" state, it does its food seeking thing. When it is in the "worried survivor" state, it does its running from ghosts thing.

One way to deal with this idea of state is to use state variables to record information. That is you set some variable to record that the agent is happily foraging, and then execute foraging code until you hit an alert that a ghost is close. Then you set the variable to record that it is time to be in survival mode, and execute the appropriate code until the danger passes. One way to use this kind of state-based structure is illustrated in the `CornerSeekingAgent` code that is in the solution for Practical 2. In that case the state information records which corner the agent is currently heading for.

Getting the combined code to a point where it can, at the right point, switch from sweeping up food to running for its life, and back, is the minimum you should plan to get done this week. You will likely need more than the one hour of scheduled lab time to do this.

5 More

Other things to think about:

1. Whether or not you are using a map, you will need an overall plan to sweep the food up if you want to win games.

   One way to do this is through the use of *waypoints*. These are coordinates within the map that Pacman targets — if they are in the right places, then going from one to another will mean going through every location with food in it. Try identifying a set of waypoints that make this happen.

   Again you can use the way that the solution to `CornerSeekingAgent` finds corners to do this. However, you will need to develop a general approach to getting to a point. The hard-coded version in `CornerSeekingAgent` exploits the fact that corners have maximal and minimal coordinates, and that there are no walls between the top and bottom corners at a given ends of the grid.

2. If you have only run Pacman using the default layout or `mediumClassicNoGhosts` (which has the same layout), make sure your code runs on different layouts.

6 Version list

- Version 1.0, October 15th 2018.