6CCS3AIN (Artificial Intelligence)

Practical 07: Pacman is running scared

(Version 1.2)

1 Overview

This practical continues your journey towards the coursework. Last week you worked on a simple version of Pacman, treating it as an MDP, and building an implementation of value iteration that could be used to play this simple version of Pacman. This week we will complicate things by (re)introducing ghosts. Doing this obviously makes Pacman more complicated again. You will have to adapt your previous approach to incorporate ghosts, and you will have to deal with the fact that ghosts move (unlike food). A new version of the API allows Pacman to identify ghosts that are scared/edible and so take advantage of knowing that there is no need to run from scared/edible ghosts.

2 Another new layout

Since this week is all about bringing the ghosts back, we need a layout to work with that includes ghosts. The layout to use is is the one in Figure 1. You can think of this as an extension of `smallMDPGrid.lay`, which adds another food and a ghost[1]. To use it, do the following:

1. Download `smallGrid.lay` from KEATS.
2. Place `smallGrid.lay` in the `layouts` folder in your `pacman` folder
3. Run your `SimpleMDPAgent` from last week using this layout:

   ```
   python pacman.py --pacman SimpleMDPAgent --layout smallGrid
   ```

You may find that your `SimpleMDPAgent` can win games in this layout as it stands. However to be able to handle this layout reliably, and to deal with the `mediumClassic`layout that we started with (and which will feature in the coursework) you will probably need to modify your code. How to do this is discussed in the next section.

3 Two things you need to do to handle ghosts

If you got far enough last week to have a working MDP solver, there is not a lot that you have to do to to handle the new dynamic environment that Pacman has to cope with. If you didn't yet get a working MDP solver, stop working on this practical and go finish your `SimpleMDPAgent` as quick as you can.

The things you need to do to handle ghosts are:

[1]In fact, this "new" layout comes direct from the Berkeley AI project, and I used it to create `smallMDPGrid.lay`.

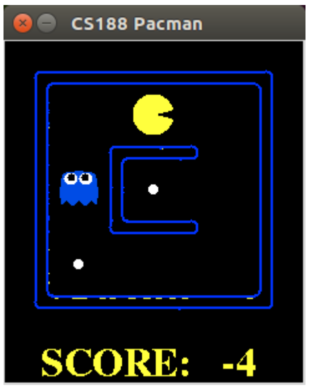

Figure 1: The layout to use this week

1. Be able to recognise that the locations where ghosts are, are locations to avoid.
2. Be able to cope with the fact that ghosts move.

Dealing with the first is a matter of adjusting how rewards are assigned to locations. For the `SimpleMDPAgent` you could just assign positive rewards to food, feed that into value iteration, and be done. To ensure that Pacman avoids ghosts, you need to feed negative rewards into value iteration. If you set the right rewards, value iteration should ensure that the utilities are such that Pacman runs away from ghosts when they need to.

Dealing with the second issue gets to the heart of MDPs and value iteration. They are a methods for dealing with non-deterministic actions, but they only give optimal solutions in static worlds. The utilities that are computed by value iteration, and the policy that is determined by those utilities, are only optimal if the rewards don't change. With ghosts moving around, the rewards change dramatically every step. The simple solution is to update rewards and run value iteration every step.

4 A wider world

Ultimately (by which I mean "for the coursework") you should be able to deal with `mediumClassic`, the layout that gets run by default:

```
python pacman.py --pacman SimpleMDPAgent
```

This is larger than `smallGrid`, so is computationally harder, and has two ghosts, so is also harder in a game playing sense. There are no big, conceptual, issues to deal with in moving from `SmallGrid` to `mediumClassic`[2], but you may find that you need to tweak some parameters in the MDP solver to get Pacman to play well (or at least as well as it is possible to play in a non-deterministic world).

Although there are no conceptual issues in moving from `SmallGrid` to `mediumClassic`, the amount of computation increases quite a lot, and you may need to think about some of the measures discussed in Section 7 to make you code run at a reasonable speed[3].

[2]By which I mean my code transferred from `smallGrid`, to `mediumClassic` without any major surgery — rather to my surprise — but, depending on your implementation, the same may not be true for you.

[3]Note that, I'm pleased to say, the map representation I gave you in `MapAgent` seems to scale well enough for this exercise — I just ran my `MDPAgent`, which is a subclass of the `MapAgent` I gave you, on `mediumClassic` over

5 The ghosts are scared?

It is possible to get Pacman to win in `MediumClassic` by just being good at running away from ghosts[4]. However, since that layout also includes capsules, Pacman is sometimes able to eat ghosts, and you may want to exploit this. There is a new version of the API that allows you to do this:

1. Create a backup copy of the current version of `api.py` that you are using. For example, call it `api-v4.py`
2. Download Version 5 of `api.py` from KEATS.
3. Place this in your `pacman` folder
4. Run your `SimpleMDPAgent` from last week using this version of the API:

   ```
   python pacman.py --pacman SimpleMDPAgent --layout smallGrid
   ```

You should see no difference in the output or functionality of your code — none of the existing API functions have changed. However, there are two new functions:

- `api.ghostStates`

  This returns a list of ghost locations, each paired with a flag that indicates whether the ghost is currently scared/edible or not.
- `api.ghostStatesWithTimer`

  As for `ghostStates`, but if the ghost is scared/edible, the flag indicates how many more time steps the ghost will remain scared/edible.

Sample output from `api.ghostStates` is:

```
[((3.0, 7.0), 0), ((13.0, 4.0), 0)]
[((3.0, 6.0), 1), ((13.0, 3.0), 1)]
```

The first line shows we have two ghosts, one at `(3, 7)` and one at `(13, 4)`, and neither are scared/edible. The second line, from the next point in time, shows the two ghosts at `(3, 6)` and `(13, 3)`, and shows that they are both now scared/edible.

6 Your mission

Your job for this week is to create an `MDPAgent` that can deal with ghosts. Getting the `MDPAgent` to a point where it can win games in `smallGrid` is the minimum you should aim to get done this week.

You will likely need more than the one hour of scheduled lab time to do this.

Working on getting `MDPAgent` to handle `mediumClassic` is also advisable (though may well take longer). If you can deal with `mediumClassic` by the end of this week, you will be very well placed for the coursework.

15 games, and that took 7 minutes, 25 seconds. If you want a benchmark to beat, my `MDPAgent` won 8 of those games and got an average score of 644. I'm sure you can do better (though there is no need to try, it won't be a requirement of the coursework).

[4]That is all my `MDPAgent` does.

7 More

Other things to think about

1. Since it is necessary to run value iteration at each step, in order to take account of the movement of ghosts, efficiency becomes a question. One obvious way of making value iteration run more efficiently are to allow/force it to terminate earlier. Thus, if you are waiting until all the values are no longer changing before terminating value iteration, try allowing some slack — set a threshold, and when the change in values is below the threshold stop the iteration. Increasing this threshold stops the iteration sooner, and gives a more approximate solution. Experiment with the threshold to see what effect it has on speed and performance.

2. Another obvious way of ensuring that value iteration is efficient is to stop it running after some fixed number of iterations. Can you find a number that works well for both the layouts?

3. Experiment with different decision criteria for selecting actions. Do you see any difference in performance?

8 Version list

- Version 1.2, October 8th 2019 – fixed typos.
- Version 1.1, September 27th 2019 – fixed typos.
- Version 1.0, September 23rd 2019.