6CCS3AIN & 7CCSMAIN, 2017

Coursework 1

(Version 1.1)

1 Introduction

This coursework exercise asks you to write code to control Pacman in the standard grid that we used for the practical exercises.

Read all these instructions before starting.

This exercise will be assessed.

2 Getting started

You should download the file `pacman-cw1.zip` from KEATS. This contains a familiar set of files that implement Pacman, and version 3.0 of `api.py` which defines the observability of the environment that you will have to deal with.

Version 3 of `api.py` (which you will already have met if you worked through Practical 3), further restricts Pacman's view of the world beyond what it was. Now, if they are moving, Pacman can only see food and capsules ahead along the corridor (that is in the same direction as Pacman is heading) up to the same 5 step distance that was possible before with Version 2. However, in this case, if there is a wall less than 5 steps away, visibility stops at the wall. If Pacman is passing a side corridor, they can see one step down that corridor (out of the corner of their eye).

With the new `api.py`, ghosts can be seen just like food and capsules — ahead if Pacman is moving, and forwards and backwards (and left and right if there are corridors) when Pacman is stationary.

Since Pacman moves at the same speed as the ghosts, not being able to see ghosts behind should not be a problem — if Pacman is moving, it can't be caught by a ghost behind, and if it reverses direction, the 5 steps are a cushion that should allow Pacman to stop before a ghost that is behind catches them. However, as a bonus, Pacman can always detect a ghost that is 2 steps or less away. (Perhaps Pacman can hear the ghosts move.) This means that Pacman should be able to avoid colliding with a ghost in the intersection of two corridors, something that can't be avoided if Pacman can only see along corridors.

3 What you need to do

3.1 Write code

This coursework requires you to write code to control Pacman and win games despite the limitations that `api.py` places on observability. There is a (rather familiar) skeleton piece of code to take as your starting point in the file `partialAgents.py`. This code defines the class `PartialAgent`.

There are two main aims for your code:

(a) Be able to win games when there are no ghosts.

(b) Be able to win one game in five, on average, when there are ghosts.

To win games, Pacman has to be able to eat all the food. For this coursework, "winning" just means getting the environment to report a win. Score is irrelevant.

3.2 Things to bear in mind

Some things that you may find helpful:

(a) We will evaluate whether your code can win a game when there are no ghosts by running:

```
python pacman.py -p PartialAgent -l mediumClassicNoGhosts
```

`-l` is shorthand for `-layout`. `-p` is shorthand for `-pacman`.

(b) We will evaluate whether your code can win a game when there are ghosts by running:

```
python pacman.py -n 5 -p PartialAgent -l mediumClassic
```

The `-n 5` runs five games in a row.

(c) When using the `-n` option to run multiple games, the same agent (the same instance of `partialAgent.py`) is run in all the games.

That means you might need to change the values of some of the state variables that control Pacman's behaviour in between games. You can do that using the `final()` function.

(d) You are not required to use any of the methods described in the practicals.

(e) If you wish to use any of the code I provided (such as that for `CornerSeekingAgent` and so on), you may do this, but you need to include comments that explain what you used and where it came from (just as you would for any code that you make use of but don't write yourself).

(f) You can only use libraries that are part of a the standard Python 2.7 distribution. This ensures that (a) everyone has access to the same libraries (since only the standard distribution is available on the lab machines) and (b) we don't have trouble running your code due to some library incompatibilities.

3.3 Write a report

Write up a description of your program along with your evaluation in a separate report that you will submit along with your code.

As you work through your implementation of Pacman's strategy, you will find that you are making lots of decisions about how precisely to translate your ideas into working code. The report should explain these at length. The perfect report will give enough detail that we don't feel we have to read your code in order to understand what you code does (we will read it anyway).

Remember, when doing this, that there is credit for creative and beautiful solutions. Make sure you highlight these aspects of your work, especially things that make your work unique.

Having said that, reports that are needlessly long will not get any more credit. We value concise reports (we have to read a lot of them).

Your report should also analyse the performance of your code. Because there is a certain amount of randomness in the behaviour of the ghosts, a good analysis will run multiple games to assess Pacman's performance. For example, you might like to try running:

```
python pacman.py -n 50 -p PartialAgent -l mediumClassic
```

to get a statistically significant number of runs. (Of course, to decide whether this was a statistically significant number of runs, you would have to do some statistical analysis — it might well need more runs.) All the conclusions that you present in your analysis should be justified by the data that you have collected.

3.4 Limitations

There are some limitations on what you can submit.

(a) Your code must be in Python 2.7.

Code written in a language other than Python will not be marked.

Code written in Python 3.X is unlikely to run with the clean copy of `pacman-cw1` that we will test it against. If is doesn't run, you will lose marks.

Code using libraries that are not in the standard Python 2.7 distribution *may* not run. If you choose to use such libraries and your code does not run, you will lose marks.

(b) All your code must be included in `partialAgents.py`. You are only allowed to submit one file of code.

(c) Your code must only interact with the Pacman environment by making calls through functions in Version 3 of `api.py`. Code that finds other ways to access information about the environment will lose credit.

The idea here is to have everyone solve the same task, and have that task explore issues with partial observability.

(d) You are not allowed to modify any of the files in `pacman-cw1.zip` except `partialAgents.py`.

Similar to the previous point, the idea is that everyone solves the same problem — you can't change the problem by modifying the base code that runs the Pacman environment.

(e) You are not allowed to copy code that you might get from other students or find lying around on the Internet. We will be checking.

This is the usual plagiarism statement. When you submit work to be marked, you should only seek to get credit for work you have done yourself. When the work you are submitting is code, you can use code that other people wrote, but you have to say clearly that the other person wrote it — you do that by putting in a comment that says who wrote it. That way we can adjust your mark to take account of the work that you didn't do.

4 What you have to hand in

Your submission should consist of a single ZIP file. (KEATS will be configured to only accept a single file.) This ZIP flle should include a single PDF document (the report), and a single Python file (your code).

Your report must be named:

`cw1-<lastname>-<firstname>.pdf`

so my report file would be named `cw1-parsons-simon.pdf`. Reports that are not in PDF format, or are not named correctly will lose marks.

Code not written in Python will not be marked.

Remember that we are going to evaluate your code by running your code by using variations on

```
python pacman.py -p PartialAgent
```

(see Section 5 for the exact commands we will use) and we will do this in a vanilla copy of the `pacman-cw1` folder, so the base class for your agent must be called `PartialAgent`

To streamline the marking of the coursework, you must put all your code in one file, and this file must be called `partialAgents.py`

Do not just include the whole `pacman-cw1` folder. You should only include the one file that contains the code you have written. If you include more than one Python file, you will lose marks.

Remember that there are a lot of you taking this module, so there will be lots of work that we have to mark. Submissions that do not follow these instructions will create more work for us and slow down the marking. As a result, deviating from the instructions will mean that you lose marks.

5 How your work will be marked

There will be six components of the mark for your work:

(a) Functionality

We will test your code by running your `.py` file against a clean copy of `pacman-cw1`.

As discussed above, for full credit for functionality, your code is required to:

(a) Be able to win a game of Pacman when there are no ghosts.
This will be assessed running your code with the command:
`python pacman.py -q -p PartialAgent -l mediumClassicNoGhosts`

(b) Be able to win one game of Pacman in five when there are ghosts.
This will be assessed by running your code with the command:
`python pacman.py -q -n 5 -p PartialAgent -l mediumClassic`
and checking to see if it wins at least one game.
Since we will check it this way, you may want to reset any internal state in your agent using `final()` (see Section 3.2).

Code that fails to meet these requirements will lose marks.

Note that since we have a lot of coursework to mark, we will limit how long your code has to demonstrate that it can win. We will terminate run of the "no ghost" game after 30 seconds, and will terminate the five runs of the regular game after 2 minutes. If your code has failed to win within these times, we will mark it as if it failed to meet the requirements. (Note that, as illustrated above, we will use the `-q` option which runs the game with no interface making it run much faster. If you are concerned about the time your code takes to run, try it with the `-q` option.)

While the main requirement is for your code to run on the `mediumClassic` grid, we will also test your code on another grid. The point of this is to check to see if your code is general enough to run on another grid — that is, it does not have any hardwired assumptions about the size of the grid or the location of objects in it. Given the nature of the test, it makes no sense to say what this third grid will be.

(b) Style

There are no particular requirements on the way that your code is structured, but it should follow standard good practice in software development and will be marked accordingly.

Remember that your code is only allowed to interact with the Pacman environment through version 3 of `api.py`. Code that does not follow this rule will lose marks.

(c) Documentation

All good code is well documented, and your work will be partly assessed by the comments you provide in your code. If we cannot understand from the comments what your code does, then you will lose marks.

(d) Report

The contents of the report are described above. Your work will be assessed against the criteria laid out there.

(e) Results

In addition to looking at your experimental results to assess the functionality of your code, we will be looking to check that you did some evaluation, that you analysed the data as required, and that you have drawn sensible conclusions from the experiments. Proper statistical analysis will be credited.

(f) Creativity

In order to give credit to students who come up with particularly beautiful, sophisticated and/or creative solutions, there will be some marks available to recognise solutions that go beyond the average.

As with all instances of creativity, I can't specify what this would, other than to say that we (the markers) will know it when we see it. Impress us.

And if you want to be sure that we don't miss your creative solutions, make sure your report tells us about them.

A copy of the marksheet, which shows the distribution of marks across the different elements of the coursework, will be available from KEATS.

6 Version list

- Version 1.0, October 15th 2018
- Version 1.1, October 16th 2018
  - Fixed typos