CMP9132 (Advanced Artificial Intelligence)

Guidance for Assignment 2

(Version 1.0)

1 Overview

For this assignment, you will have to implement an AI-based approach to decision making. Your code will be helping a character, called Tallon, decide what to do in a simple video game called *Mean Arena*[1]. No previous experience with this video game is required, which is good since the game was written specifically for this assignment.

This assignment is worth 50% of the marks for the module.

Note: Failure to follow submission instructions (see below) will result in a deduction of 10% of the marks you earn for this coursework.

2 Getting started

2.1 Start with Mean Arena

The code for *Mean Arena* is available from the CMP9132 site on Blackboard at:

`Assessment > Assessment Documents > Assessment Item 2`

1. Download:

   `meanArena.zip`

   from Blackboard.

2. Save that file to a computer — either yours or one of the lab machines.

3. Unzip the archive.

   This will create a folder `meanArena`

4. Switch to the folder `meanArena`.

5. From the command line, run the game:

   `python3 game.py`

   and watch it run. (No doubt there is a way to run this as a Jupyter notebook, but I haven't run it that way.)

[1] Any connection to a long-ago comic strip of the same name is purely coincidental.

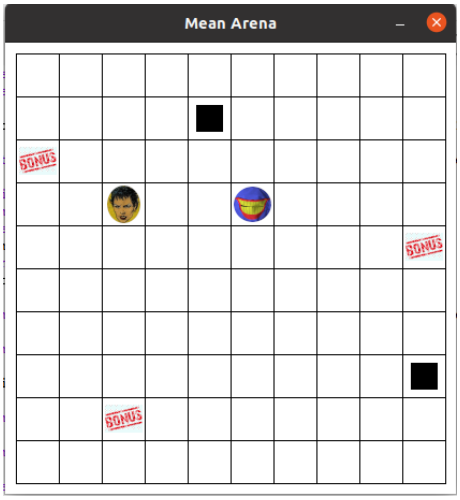

Figure 1: The interface for Mean Arena

Figure 1 shows a screenshot of the game in its early stages. The figure shows Tallon (the yellowish icon), a single Blue Meanie (evil grin in a blue background), some bonuses, and two pits. This is the default configuration — the game starts with a single Meanie, three bonuses and two pits. A new Meanie appears every few seconds. Tallon scores points by avoiding the Meanies, and by claiming bonuses.

The Meanies have two modes of behaviour. If they are too far away from Tallon, they move randomly. When they are closer to Tallon they move to intercept them — the simple algorithm is to reduce the x or y coordinate separating them from Tallon at every move. This is surprisingly effective, especially when there are several Meanies.

Tallon's default behaviour is to move towards the bonuses in the same way, ignoring the Meanies and the pits. This is the behaviour that you are asked to improve on with your solution.

2.2 Towards an AI Tallon

Now, your job is to write code that makes decisions for Tallon. If you look in `tallon.py`, you will find a method `makeMove()`. This is the method that is called by the game to move Tallon around the world, so everything that you do will be triggered by calls to this method. (That doesn't mean you have to write all your code in this method, but you will have to call any code that you want to run from this method.)

In particular, `makeMove()` has to return one of the four legal values:

```
Directions.NORTH
Directions.SOUTH
Directions.EAST
Directions.WEST
```

You don't have to, and indeed you should not plan on, writing code in any file other than `tallon.py`. (Though your evaluation will probably require you to modify `config.py` to alter parameters.) However, it may be helpful to understand roughly what the other files are doing. They are as follows:

`config.py` a file of configuration data which allows you to change the size of the grid, the number

of pits, the starting number of Meanies, the degree of visbility, how far away a Meanie can detect Tallon and so on.

`arena.py` code that draws the arena on the screen.

`game.py` code that runs the game until Tallon wins or loses. This includes the basic control loop for the game.

`world.py` code for the `World` object which keeps track of everything, and makes the decisions about how the Meanies will move. It provides the data that is used by `arena.py` to draw the state of the game on the screen.

`utils.py` some utilities that are used in a few places.

`graphics.py` simple Python graphics (not written by me).

And with that, it is over to you.

3 What you have to do

3.1 Write some code

Now you should fill out the `Tallon` class with code that improves on the current decision making about how to act. Given that the current code ignores both the pits (so that Tallon will often wander into a pit) and Meanies (so Tallon takes no evasive action), this should not be too difficult.

Note that you should write code under two conditions;

1. When the *Mean Arena* is fully observable.

   The practical impact of this is that Tallon "knows" where all the relevant objects are — Meanies, bonuses and pits. In implementation terms, this is achieved by setting the parameter `partialVisibility` to `False` in `config.py`.

2. When the *Mean Arena* is partially observable.

   The practical impact of this is that Tallon only "knows" where relevant objects are if they are within a certain distance, a distance controlled by the parameter `visibilityLimit` in `config.py`. In implementation terms, this is achieved by setting the parameter `partialVisibility` to `True`.

You can write two separate solutions, one for each condition, or you can write one solution that works for both solutions. The choice is yours.

3.2 Evaluate your solution(s)

An important part of the assessment is to carry out an evaluation of the solution (or solutions) that you have written to control Tallon. The idea is that you should try to figure out how well your code works across a range of conditions.

You should definitely include results for both fully observable and partially observable versions of the *Mean Arena*, but beyind that, you can choose what parameters to vary over your evaluation.

You might, for example, look at performance over a range of sizes of the grid, different numbers of bonuses and/or pits, and how quickly the Meanies spawn. However, any of the values in `config.py` could be used.

3.3 Write a report

You have to write and submit a report (in addition your code). The report does not have to be long — it should be a maximum 3 pages, font size of 11 or 12 points, not including the cover sheet, references, and any appendixes — but it should cover two things:

1. It should describe the approach to decision-making that you implemented in your code, and it should say why you consider it to be an AI approach.
2. It should describe the results of your evaluation. This should include a discussion ofthe ocnclusions that you draw from the evaluation.

   If you wish, tables and figures from the evaluation can be placed in appendices rather than being part of the 3 pages of the report.

3.4 Submit your work

See Section 4 for a description of what to submit.

3.5 Things to know

Here are some things to think about when writing your version of `tallon.py`:

- Your code must contain a recognisable AI algorithm. Code that does not contain such an algorithm will lose marks.

  You should use the report to justify why your code counts as AI.

- What AI algorithm you implement is up to you.

  The marks you get for your code will be partly determined by the sophistication of the code that you implement. If you use a simple rule-based approach, for example, that will not get as much credit as something which makes use of some of the probabilistic methods that we covered in the lectures.

- When this is marked, I'm going to take your version of `tallon.py` and run it against a vanilla copy of the `meanArena` code. So there is no point writing any code in any of the other files.

  Having said that, in order to carry out your evaluation, you will want to change some of the paarmeters in `config.py`, but that is not part of the code you will submit.

- To get a good mark you will need to do some evaluation.

  40 of the 100 marks for this assessment will be for the evaluation.

3.6 Limitations

There are some limitations on what you can submit.

1. Your code should be submitted as a Python source file `tallon.py` (not, for example, as a PDF of a Python file, or a Jupyter Notebook).

   Submissions of code that are not Python source files will lose marks.

2. Your code should be in Python 3.X.

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

   Code written in Python 2.X is unlikely to run with the clean copy of `meanArean` that I will test it against. If it doesn't run, you will lose marks.

3. The base class for the code you write must be called `Tallon`, and it must provide a `makeMove()` method. Otherwise you code will not be executed when I run it against a vanilla copy of `meanArena`.

4. Code using libraries that are not in the standard Python 3.X distribution and which we have not used in the workshops for the module *may* not run when I test your code. If you choose to use such libraries and your code does not run when I test it, you will lose marks.

   It is possible to create a good solution to the problem just using the standard Python distribution, and it is possible to create a good solution using the standard Pythion distribution and the libraries that we used in the workshops.

5. You are not allowed to copy, without credit, code that you might get from other students or find lying around on the Internet. (This includes the use of code that was distributed as part of the module — if you use code from files other than `tallon.py` without attribution, I will consider that to be plagiarism.) I 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 I 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 two parts:

1. A PDF file of your report. This should be submitted to "ASSESSMENT ITEM 2 UPLOAD" in Blackboard.

   The PDF file should be named:

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

   so my PDF file would be named `mean-parsons-simon.pdf`. This naming convention means it is easier for me to handle all the submissions efficiently (which is important in getting your marks for the assignment back to you on time). Submissions that are not in PDF format will lose marks.

2. A ZIP file which should be submitted to "ASSESSMENT ITEM 2 SUPPORTING DOCUMENTATION UPLOAD" in Blackboard.

   The ZIP file should include only the Python source file `tallon.py`

   The ZIP file should be named:

   ```
   mean-<lastname>-<firstname>.zip
   ```

   so my ZIP file would be named `mean-parsons-simon.zip`. Submissions that are not in ZIP format will lose marks. Again, the naming convention will help speed the marking.

   Do not just include the whole `meanArena` folder. You should only include the one file that includes the code you have written.

Submissions that do not follow these instructions will lose marks.

5 How your work will be marked

There will be three components of the mark for your work, exactly as in the Assignment Brief.

1. Functionality in a fully observable world

   I will test your code by running your `tallon.py` file against a clean copy of `meanArena` with `partialVisibility` set to `False`.

   Your mark will be based on both the performance of the code, and on the sophistication of the AI method, based both on looking at the code, and on what you say about it in your report.

2. Functionality in a partially observable world.

   I will test your code by running your `tallon.py` file against a clean copy of `meanArena` with `partialVisibility` set to `True`.

   Your mark will be based on both the performance of the code, and on the sophistication of the AI method, based both on looking at the code, and on what you say about it in your report.

3. Quality of your evaluation

   Your mark will be based on your description of your evaluation in your report.

Version list

- Version 1.0, January 25th 2022