COMP341 Introduction to Artificial Intelligence
HW5: Reasoning over Time

This programming homework will test your knowledge and your implementation abilitites of what you have learned in the *reasoning over time* part of the class. This homework has two parts; a programming part and a written report part. The written report will require you to interpret the output of the programming part.

The programming part of this homework follows the Berkeley CS188 Spring 2014 Pacman Project 4: Ghostbusters project at `http://ai.berkeley.edu/tracking.html`.

This homework needs to be completed individually. Discussion about algorithms and data structures are allowed but group work is not. We trust that you will submit your own work for full credit.

However, you might find yourself having trouble implementing the algorithms. I am going allow you to copy or look at someone else's code as long as you credit the person or the website! In this case I will grade you differently, as described in the next part.

Any academic dishonesty, which includes you taking someone else's code but not properly crediting the source, will not be tolerated.

Grading

The code part and the report will have 2:1 weight ratio in your submission (programming 2/3, report 1/3). In order for the report to be graded, corresponding programming parts need to be submitted.

In case you have copied/used someone else's code as stated above, you will only be graded for your report. In this case you will be eligible for up to half the credit, i.e., the report will have 1/2 weight. You should not submit your code in this case **AND** properly credit the source in your report. Reports without proper credit will not be graded.

Finally, we are going to compare your code to previous year's submissions, each others and other online sources known to us. If your code's similarity level is above a certain threshold, your code will be scrutinized and you might end up getting a 0.

Part 1: Programming

Instructions

You are going to do the 7 programming questions about reasoning over time given here: `http://ai.berkeley.edu/tracking.html`. Feel free to use the forum in the blackboard for general discussion, however remember there is no group work.

You are only required to change *inference.py* and *bustersAgent.py*. If you have any issues with other parts of the code let your instructor or TA know ASAP, even if you manage to solve your problem. Use the data structures and sampling functions in the *util.py* to your advantage. If you think you have the right answer but the autograder is not giving you any points, try to run it on individual questions. In case that you are sure your algorithm is working properly, but the autograder is not giving you any points do not worry. Let the TA or the instructor know after your submission and we are going to arrange a time for you come in and argue why you think your algorithm is running correctly.

Hints

The first 5 programming questions are fairly straightforward especially if you paid attention in the class. Furthermore, both the website and the code comments include a lot of hints so make sure you read them!

Website and the Comments

I cannot stress this enough so I am just going to repeat. This homework has a lot of guidance, both in its **webpage** and in the code **comments**. I suggest you read them carefully. Also read $q1, q2, q3$, $q4, q5$ and $q6, q7$ together.

Exact Inference

There is really nothing else to hint at. One minor warning is that to pay attention of the time elapse loop. The required summation can happen "out of sequence".

Uniform Initialization of Particles

As stated in the comments, do not initialize randomly (even with a uniform distribution!) but do it uniformly. Imagine I give you 10 slots and 100 balls. How would you fill those slots uniformly? Find the number of particles per ghost position you need to have and take it from there. There are cases analogous to you having 104 balls for 10 slots. I leave it up to you to deal with the remainder (i.e 4) as you like, random is fine. A piece of information; the number of particles is much bigger than the legal ghost positions.

Particle Weight Representation and Particle Resampling

You may find it easier to use `util.Counter()` data structure to keep track of particle weights instead of having another list. This will be indexed by the ghost positions. However, we have more particles than these positions. I leave it up to you to figure out how to deal with this in case you chose this. Note that this data structure can represent any number given an index in general, they do not have to be probabilities.

Note that you need to resample after updating the particle weights in the `observe()` method of the inference class. The advantage of using the `util.Counter()` data structure is that the existence of the `util.sample()` function.

You can always use another list or keep a combined list for the weights, in which case you will need to implement your own sampling function.

Dynamic Bayes Nets

When you are done with the first 5 questions, I suggest you take a break and answer the relevant questions in your report before working on these last 2 questions. I do not want to spoil all the fun for when you get back. So I will only briefly mention a few things.

A particle will include positions of two ghosts. The emission models are for individual ghosts. To get a particle weight, you need to multiply the emission probabilities together. If a ghost is in a jail cell, then its emission probability becomes 1.

In the previous cases, a ghost had legal positions which you used to pick uniformly distributed particles. In this case it gets slightly tricky. Think carefully about what you need to pick from. I will give a small example without explaining it. Let $\{(1,3),(2,2),(2,3)\}$ be a list of legal positions. Assuming that 2 ghosts cannot be in the same position, you need to pick from: $\{[(1,3)(2,2)]$, $[(1,3)(2,3)]$, $[(2,2)(1,3)]$, $[(2,2)(2,3)]$, $[(2,3)(1,3)]$, $[(2,3)(2,2)]\}$.

Part 2: Report

This part includes answering the following questions based on your program's output on the given pacman tests. You are expected to answer the questions concisely. This means maximum ten sentences per question. Answers that are longer than 300 words will only be graded for the first 300 words or so. It is okay if you over-generalize, as long as your direction is clear and correct. **Write your name and student ID on your report.** This was not in the previous homework instructions.

Written Q1:

Run the autograder on $q2$ and watch the probabilities. Why do they settle even though the ghost is moving? Can you tell the two ghosts apart and if so how? (Hint: Run these test cases q2/1-ExactElapse, q2/2-ExactElapse, if you need to observe them individually)

Written Q2:

Try the following lines of code and watch the probabilities settle:

```
python autograder.py -t test_cases/q1/2-ExactObserve
python autograder.py -t test_cases/q1/3-ExactObserve
```

Why is it the case that in one of them we can find the ghost but not in the other one?

Written Q3:

Run the autograder on $q4$ and watch the probabilities. Can you tell when the particles get re-initialized? Comment on the reason(s) on why pacman gets in that situation? Would increasing the number of particles be a solution?

Written Q4:

Compare how the probabilities evolve between the exact inference and the approximate inference cases ($q1, q2$ vs $q4, q5$). Also comment on if 5000 particles make sense for the problems you have seen.

Written Q5:

In the DBN questions ($q6, q7$), you had to work on a particle that had multiple ghost positions. Their transition were dependent on each other but their emissions were not. How did you create a new particle with elapsed time and how did you calculate its weight? You can use mathematical equations to help you explain this.

Written Bonus:

What is the difference between the sonar in this homework and the one we have seen in class?

Submission

You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar* or *tar.gz* format. This should extract to a folder with your student ID which includes *inference.py*, *bustersAgents.py*, and *report.pdf*.

Submission instructions:

- You are going to submit a compressed archive through the blackboard site.
- The file can have *zip*, *tar* or *tar.gz* format.
- This compressed file should extract to a folder with your *student identification number* with the two leading zeros removed which should have 5 digits.
- The previous point is very important, I do not want to see multiple folders (apart from operating system ones such as MACOSX or DS Store). I do not want to play inception with your code.
- Inside the folder, you should only have *inference.py*, *bustersAgents.py*, and *report.pdf*. Anything else might will be deleted. If they interfere with the grading process, I will simply ignore your programming homework.
- One advice is after creating the compressed file, move it to your desktop and extract it. Then check if all the above criteria is met.
- **DO NOT SUBMIT CODE THAT DOES NOT TERMINATE OR THAT BLOWS UP THE MEMORY**. I will take these as malicious acts and will proceed accordingly.

Let us know if you need any help with setting up your compressed file. This is very important. I will put all of your zipped files into a folder and run multiple scripts to grade, separate reports and do similarity testing. If you do not follow the above instructions, then scripts might fail. This will lead you to get a 0 from your programming part.