Koç University
COMP341
Introduction to Artificial Intelligence
Assignment 4

Instructor: Barış Akgün
Due Date: May 26 2024, 23:59
Submission Through: Blackboard
Make sure you read and understand every part of this document

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the reasoning over time part of the class. You are asked to complete a coding part and answer a few questions about how it runs. The coding part of the homework will follow the Berkeley CS188 Fall 2021 Pacman Project P4: Ghostbusters at `https://inst.eecs.berkeley.edu/~cs188/fa21/project4/`.

This homework must be completed individually. Discussion about algorithms, algorithm properties, code structure, and Python is allowed but group work is not. Coming up with the same approach and talking about the ways of implementation leads to very similar code which is treated as plagiarism! Furthermore, do not discuss the answers directly as it will lead to similar sentences which is treated as plagiarism. If you are unsure, you should not discuss. Any academic dishonesty, will not be tolerated. **By submitting your assignment, you agree to abide by the Koç University codes of conduct.**

Important: Make sure you go over all the submission instructions at the end of this document before submission. Once you upload your submission, download it to make sure it is not corrupted and it has your latest documents. You are only going to be graded by your blackboard submission.

Warning: The solution code for the homeworks can be found online. We are going to compare your submission with these sources. We are also going to compare your code to latest and previous submissions of Koç students. If your code's similarity level is above a certain threshold, your code will be scrutinized. If there is strong suspicion of plagiarism, we will take action based on university policies.

Programming

You are going to do the 11 (Q0-Q10) programming questions about reasoning over time given in the website. You are only required to change *bustersAgent.py* and *inference.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. If you think you have the right answer but the autograder is not giving you any points, try to run it on individual questions (examples on how to do this is given in the website).

Hints

The first 8 programming questions are fairly straightforward especially if you paid attention in 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.

Getting an error related to cgi.message: If you get this exception, go to the corresponding file and add `import html`. Then change `cgi.message` to `html.message`. This would happen if you upgraded your Python version.

The DiscreteDistribution Class: Be careful during normalization when the total is 0! Otherwise, you might get a divide-by-zero exception later on. Also, the doctest for sampling assumes you normalize within the `sample` function. This is not necessary if you always normalize before. If you do not want to normalize within the `sample` function you can add a `dist.normalize()` to the doctest comments. However, make sure you normalize before any sampling steps later on in the code! Also, do not forget to normalize any distributions you calculate as a final step.

Exact Inference: There is really nothing to hint at other than what is given in the website. One minor warning is to pay attention to the time elapse loop. The required summation can happen "out of sequence". Also pay attention to jail positions, make sure to set probabilities for all the states (if a ghost is in jail, corresponding probability will be 1, all the other states will have 0 probability).

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 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 typically much bigger than the legal ghost positions.

Particle Weight Representation and Particle Resampling:

You may find it easier to use the `DiscreteDistribution()` class 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 `observeUpdate()` method of the `ParticleFilter` class. The advantage of using the `DiscreteDistribution()` data structure is that the existence of the `DiscreteDistribution.sample()` function. Be careful about normalization.

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

Dynamic Bayes Nets

For this part, 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. Each ghost will also have its own jail cell. If a ghost is in a jail cell, then its emission probability becomes 1 for the jail location.

In the previous cases, a ghost had legal positions which you used to pick uniformly distributed particles. In the DBN case, this gets slightly tricky. Think carefully about what you need to pick from. We will give a small example without explaining it:

Let {(1,3),(2,2),(2,3)} be a list of legal positions. Assuming that two 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)]}. The website is suggesting a solution using the `itertools` package. It also recommends to shuffle the resulting permutations. If you pass all the tests but the first one for Q8, lack of shuffling may be the culprit.

Note that the observations for the DBN part is multi-dimensional, based on the number of ghosts.
Getting the unhashable type error If you get an unhashable type error, go and look at the way you represent your particles. In Python, lists are not hashable (since they are *mutable*) but tuples are (since they are *immutable*).

Submission

You are going to submit a compressed archive through the blackboard site. This file should only contain *bustersAgents.py* and *inference.py*. Other files will be deleted and/or overwritten.

Submission Instructions

- You are going to submit a **single** compressed archive through the blackboard site. The file can have *zip*, *rar*, *tar*, *tar.gz* or *7z* format. If you submit files individually they may get lost.
- You are fine as long as the compressed archive has the required files within 4 folder levels.

- Code that does not run (e.g. due to syntax errors), that does not terminate (e.g. due to infinite loops) or that blows up memory will not get any points.
- **Important:** Download your submission to make sure it is not corrupted and it has your latest code. You are only going to be graded by your blackboard submission.

Best of luck and happy coding!