Koç University
COMP341
Introduction to Artificial Intelligence
Assignment 4

Instructor: Barış Akgün
Due Date: May 24 2020, 23:59
Submission Through: Blackboard

Make sure you read and understand every part of this document

Important: Download your submission to make sure it is not corrupted and it has your latest report/code. You are only going to be graded by your blackboard submission.

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 2018 pacman project P4: Ghostbusters at `https://inst.eecs.berkeley.edu/~cs188/fa18/project4.html`. The questions for the report part are given in this document.

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.**

You may find yourself having trouble implementing the coding part. In this case, we are going to let you use someone else's code to answer the given questions, as long as you credit the person or the website you take the code from. If you chose this option, we are only going to grade your report.

Grading

You are given two options about submitting your homework: (1) only the code and (2) only the report. The second option is given to you in case you are not able to implement the programming part. These options will be graded differently:

- **Only the Code:** Your maximum grade will be 100%, based on your code's output.
- **Only the Report:** Your maximum grade will be 75%.

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 previous submissions of Koç students. If your code's similarity level is above a certain threshold, your code will be scrutinized. If we see any plagiarism, you will lose points in the best case and disciplinary action will be taken against you in the worst. For the report part you will need to find an existing code or take it from one of your friends. In this case you will use whatever code you find to answer the questions given in this document. You **must** credit the code you used and **must not** submit a code part.

Code Only

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-bu-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 probabilites for all the states (if host 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
When you are done with the first 8 (Q0-Q7) questions, I suggest you take a break and answer the relevant questions in your report before working on these last 3 questions. I do not want to spoil all the fun for when you get back. So I will only briefly mention a few things.
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*).

Report Only

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. Five sentences is more than enough for most of them. Limit yourself to 300 words. It is okay if you over-generalize, as long as your direction is clear and correct.

Create a PDF file named *report.pdf* containing your answers for submission. **Write your name and your number on the report as well!**

Written Q1:
Run the autograder on Q3 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 q3/1-ExactPredict, q3/2-ExactPredict, 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/q2/2-ExactUpdate
python autograder.py -t test_cases/q2/3-ExactUpdate
```

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 q6 for the 5th and the 6th test case 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 (Q2, Q3 vs Q5, Q6). Also comment on if 5000 particles make sense for the problems you have seen.

Written Q5:
In the DBN questions (Q8-Q10), 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.

Submission

You are going to submit a compressed archive through the blackboard site. The file should extract to a folder with your student ID without the leading zeros. This folder should only contain *report.pdf* OR *bustersAgents.py* and *inference.py*, depending on how you want to be graded. Other files will be deleted and/or overwritten. Do not submit any code if you only want us to grade your report.

Important: Download your submission to make sure it is not corrupted and it has your latest report/code. You are only going to be graded by your blackboard submission.

Submission Instructions

- You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *rar*, *tar*, *rar*, *tar.gz* or *7z* 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. Multiple folders (apart from operating system ones such as MACOSX or DS Store) greatly slows us down and as such will result in penalties.
- Code that does not run or that does not terminate will not receive any credits.
- Do not trust the way that your operating system extracts your file. They will mostly put the contents inside a folder with the same name as the compressed file. We are going to call a program (based on your file extension) from the command line. The safest way is to put everyting inside a folder with your ID, then compress the folder and give it your ID as its name.
- Once you are sure about your assignment and the compressed file, submit it through Blackboard.
- After you submit your code, download it and check if it is the one you intended to submit.
- **DO NOT SUBMIT CODE THAT DOES NOT TERMINATE OR THAT BLOWS UP THE MEMORY.**

Let us know if you need any help with setting up your compressed file. This is very important. We will put all of your compressed files into a folder and run multiple scripts to extract, clean up and grade your code. If you do not follow the above instructions, then scripts might fail. If these scripts fail then you will receive a 0.