Computer Science 384
St. George Campus

Thursday, July 18, 2019
University of Toronto

Homework Assignment #4: Bayes Nets
Due: Friday, Aug 2, 2019 by 10:00 PM

Silent Policy: *A silent policy will take effect 24 hours before this assignment is due, i.e. no question about this assignment will be answered, whether it is asked on the discussion board, via email or in person.*

Late Policy: 10% per day after the use of 3 grace days.

Total Marks: This assignment represents 11% of the course grade. It's broken into two components:

1. Part I: Topic Module based assessment (worth 6%)
2. Part II: Coding based assessment (worth 5%)

As there are two parts to the assignment, there are also two TAs that you can approach for help.

The first, Zhewei Sun, will work to support the module based portion of the assignment (i.e. Part I). He can be contacted at Zhewei Sun at zheweisun at cs.toronto.edu.

The second, Raeid Saqur, will work to support the coding portion of the assignment (i.e. Part II). He can be contacted at raeidsaqur at cs.toronto.edu.

Instructions required to complete the two parts are provided below.

1 Part I: Topic Modules (60 points)

Handing in Part I

What to hand in on paper: Nothing.

What to hand in electronically: This portion of the assignment will be submitted electronically, through Google Forms. The assignment will consist of 3 modules, each worth 20 points. The URLs to the 3 modules you will have to complete can be found here:

`https://docs.google.com/spreadsheets/d/18XAStHol5FaGOKLXRPaDKh8PLYicI8jd_EmliwMapdA/edit?usp=sharing`.

You will use the last 5 digits of your student ID to identify your required modules. Different versions are assigned to each student, so make sure you only complete the modules assigned to you.

If you have problems accessing any of the modules, please e-mail the Part I TA immediately (Zhewei Sun at zheweisun at cs.toronto.edu).

Module	Topic
1	Probability Review
2	Bayesian Networks
3	D-Separation and Variable Elimination

Each module consists of a mix of multiple choice, true/false, and calculation questions. Whenever applicable, round your answer to **3 decimal places with the zero attached** to the decimal (e.g. 0.120). Feedback on the questions will only be given after submission of the form. Once you submit, click on the ”View accuracy” link. You will be able to see which questions you answered correctly/incorrectly.

Each module is worth 20 points. You may submit as many times as you want. The score of your final submission will be used to determine your final grade. For each module, you are given 5 attempts without penalty. Afterwards, each additional attempt will result in a penalty of 1 point. (For example, on your 7th attempt of Module 1, you obtain a score of 18 points. Your final score will then be 18 - (2 $\times$ 1) = 16 points for Module 1.)

2 Part II: Coding Assessment: Pacman Revisited (50 points)

Acknowledgements: This portion of the homework is based on one of Berkeley's CS188 EdX course projects. It is a modified and curtailed version of “Project 4: Ghostbusters” available at http://ai.berkeley.edu/tracking.html. The coding portion relates to the lecture material on reasoning over time.

Handing in Part II

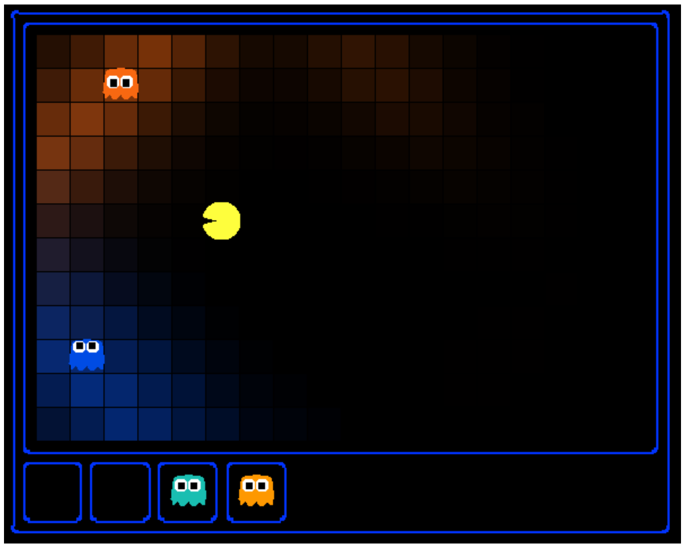

Figure 1: I can hear you, ghost. Running won't save you from my Temporal Inferences!

What to hand in on paper: Nothing.

What to hand in electronically: This portion of the assignment will be submitted electronically, through MarkUs. Download the Part II files from the A4 web page. Modify bustersAgents.py and inference.py as described below, so that they solve the problems specified in this document. You will submit the following files:

- bustersAgents.py
- inference.py
- acknowledgment form.pdf

How to Submit: If you submit before you have used all of your grace days, you will submit your assignment using MarkUs. Your login to MarkUs is your teach.cs username and password. It is your responsibility to include all necessary files in your submission. You can submit a new version of any file at any time, though the lateness penalty applies if you submit after the deadline. For the purposes of determining the lateness penalty, the submission time is considered to be the time of your latest submission. More detailed instructions for using Markus are available at *http://www.teach.cs.toronto.edu/ csc384h/summer/markus.html*.

1. Make certain that your code runs on teach.cs using python3 (version 3.7) using only standard imports. This version is installed as python3 on teach.cs. Your code will be tested using this version and you will receive zero marks if it does not run using this version.
2. Do not add any non-standard imports from within the python file you submit (the imports that are already in the template files must remain). Once again, non-standard imports will cause your code to fail the testing and you will receive zero marks.
3. Do not change the supplied starter code. Your code will be tested using the original starter code, and if it relies on changes you made to the starter code, you will receive zero marks.

2.1 Introduction

Pacman spends his life running from ghosts, but things were not always so. Legend has it that many years ago, Pacman's great grandfather Grandpac learned to hunt ghosts for sport. However, he was blinded by his power and could only track ghosts by their banging and clanging.

In this portion of the assignment, you will design Pacman agents that use sensors to locate and eat invisible ghosts. You'll advance from locating single, stationary ghosts to hunting packs of multiple moving ghosts with ruthless efficiency.

The code for this portion of the assignment contains the following files, available as a zip archive (PartII.zip).

Files you'll edit:

- **bustersAgents.py**: Agents for playing the Ghostbusters variant of Pacman.
- **inference.py**: Code for tracking ghosts over time using their sounds.

Files you will not edit:

- busters.py The main entry to Ghostbusters (replacing Pacman.py)

- bustersGhostAgents.py New ghost agents for Ghostbusters
- distanceCalculator.py Computes maze distances
- game.py Inner workings and helper classes for Pacman
- ghostAgents.py Agents to control ghosts
- graphicsDisplay.py Graphics for Pacman
- graphicsUtils.py Support for Pacman graphics
- keyboardAgents.py Keyboard interfaces to control Pacman
- layout.py Code for reading layout files and storing their contents
- util.py Utility functions

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code or you will wreak havoc on the autograder. Also do not change any of the other files, as your code will be tested against the original versions of these files. We may also run some additional tests on your code, in addition to the tests run by the autograder supplied in the zip file. Nevertheless, the marks given by the autograder should be a good indication of the final mark you will obtain.

2.2 Ghostbusters and BNs

In this assignment the goal is to hunt down scared but invisible(!) ghosts. Pacman, ever resourceful, is equipped with sonar (ears) that provides noisy readings of the Manhattan distance to each ghost. The game ends when Pacman has eaten all the ghosts. To start, try playing a game yourself using the keyboard.

```
python3 busters.py
```

The blocks of color indicate where the each ghost could possibly be given the noisy distance readings provided to Pacman. The noisy distances at the bottom of the display are always non-negative, and always within 7 of the true distance. The probability of a distance reading decreases exponentially with its difference from the true distance.

Your primary task in this project is to implement inference to track the ghosts. For the keyboard based game above, a crude form of inference was implemented for you by default: all squares in which a ghost could possibly be are shaded by the color of the ghost. Naturally, we want a better estimate of the ghost's position.

Fortunately, Bayes' Nets provide us with powerful tools for making the most of the information we have. In this portion of the assignment you will implement an algorithm to perform exact inference using Bayes' Nets for the purpose of ghost tracking.

While watching and debugging your code with the autograder, it will be helpful to have some understanding of what the autograder is doing. Test files can be in the subdirectories of the test_cases folder. As you implement and debug your code, you may find it useful to run a single test at a time. In order to do this you will need to use the -t flag with the autograder. For example if you only want to run the first test of question 1, use:

```
python3 autograder.py -t test_cases/q1/1-ExactObserve
```

In general, all test cases can be found inside

```
test_cases/q*.
```

2.3 Question 1: Exact Inference Observation (15 points)

In this question, you will update the observe method in ExactInference class of inference.py to correctly update the agent's belief distribution over ghost positions given an observation from Pacman's sensors. A correct implementation should also handle one special case: when a ghost is eaten, you should place that ghost in its prison cell, as described in the comments of the observe function.

To run the autograder for this question and visualize the output:

```
python3 autograder.py -q q1
```

As you watch the test cases, be sure that you understand how the squares converge to their final coloring. In test cases where is Pacman boxed in (which is to say, he is unable to change his observation point), why does Pacman sometimes have trouble finding the exact location of the ghost?

Note: your busters agents have a **separate inference module** for each ghost they are tracking. That's why if you print an observation inside the observe function, you'll only see a single number even though there may be multiple ghosts on the board.

Hints:

- You are implementing the online belief update for observing new evidence. Before any readings, Pacman believes the ghost could be anywhere: this translates to being a uniform prior (see initialize-Uniformly). After receiving a reading, the observe function is called, which must update the belief at every position.
- Before typing any code, write down the equation of the inference problem you are trying to solve.
- Try printing noisyDistance, emissionModel, and PacmanPosition (in the observe function) to get started.
- In the Pacman display, high posterior beliefs are represented by bright colors, while low beliefs are represented by dim colors. You should start with a large cloud of belief that shrinks over time as more evidence accumulates.
- Beliefs are stored as util.Counter objects (like dictionaries) in a field called self.beliefs, which you should update.
- You should not need to store any evidence. The only thing you need to store in ExactInference is self.beliefs.

2.4 Question 2: Exact Inference with Time Elapse (15 points)

In the previous question you implemented belief updates for Pacman based on his observations. Fortunately, Pacman's observations are not his only source of knowledge about where a ghost may be. Pacman also has knowledge about the ways that a ghost may move; namely that the ghost can not move through a wall or more than one space in one timestep.

To understand why this is useful to Pacman, consider the following scenario in which there is Pacman and one Ghost. Pacman receives many observations which indicate the ghost is very near, but then one which indicates the ghost is very far. The reading indicating the ghost is very far is likely to be the result of a buggy sensor. Pacman's prior knowledge of how the ghost may move will decrease the impact of this reading since Pacman knows the ghost could not move so far in only one move.

In this question, you will implement the **elapseTime** method in **ExactInference**. Your agent has access to the action distribution for any GhostAgent. In order to test your elapseTime implementation separately from your observe implementation in the previous question, this question will not make use of your observe implementation.

Since Pacman is not utilizing any observations about the ghost, this means that Pacman will start with a uniform distribution over all spaces, and then update his beliefs according to how he knows the Ghost is able to move. Since Pacman is not observing the ghost, this means the ghost's actions will not impact Pacman's beliefs. Over time, Pacman's beliefs will come to reflect places on the board where he believes ghosts are most likely to be given the geometry of the board and what Pacman already knows about their valid movements.

For the tests in this question we will sometimes use a ghost with random movements and other times we will use the GoSouthGhost. This ghost tends to move south so over time, and without any observations, Pacman's belief distribution should begin to focus around the bottom of the board. To see which ghost is used for each test case you can look in the .test files.

To run the autograder for this question and visualize the output:

```
python3 autograder.py -q q2
```

As an example of the GoSouthGhostAgent, you can run

```
python3 autograder.py -t test_cases/q2/2-ExactElapse
```

and observe that the distribution becomes concentrated at the bottom of the board.

As you watch the autograder output, remember that lighter squares indicate that pacman believes a ghost is more likely to occupy that location, and darker squares indicate a ghost is less likely to occupy that location. For which of the test cases do you notice differences emerging in the shading of the squares? Can you explain why some squares get lighter and some squares get darker?

Hints:

- Instructions for obtaining a distribution over where a ghost will go next, given its current position and the gameState, appears in the comments of ExactInference.elapseTime in inference.py.
- We assume that ghosts still move independently of one another, so although your code deals with one ghost at a time, adding multiple ghosts should still work correctly.

2.5 Question 3: Exact Inference Full Test (20 points)

Now that Pacman knows how to use both his prior knowledge and his observations when figuring out where a ghost is, he is ready to hunt down ghosts on his own. This question will use your observe and elapseTime implementations together, along with a simple greedy hunting strategy which you will implement for this question. In the simple greedy strategy, Pacman assumes that each ghost is in its most likely position according to its beliefs, then moves toward the closest ghost. Up to this point, Pacman has moved by randomly selecting a valid action.

Implement the **chooseAction** method in **GreedyBustersAgent** in **bustersAgents.py.** Your agent should first find the most likely position of each remaining (uncaptured) ghost, then choose an action that minimizes the distance to the closest ghost. If correctly implemented, your agent should win the game in q3/3-gameScoreTest with a score greater than 700 at least 8 out of 10 times. Note: the autograder will also check the correctness of your inference directly, but the outcome of games is a reasonable sanity check.

To run the autograder for this question and visualize the output:

```
python3 autograder.py -q q3
```

Note: If you want to run this test (or any of the other tests) without graphics you can add the following flag:

```
python3 autograder.py -q q3 --no-graphics
```

Hints:

- When correctly implemented, your agent will thrash around a bit in order to capture a ghost.
- The comments of chooseAction provide you with useful method calls for computing maze distance and successor positions.
- Make sure to only consider the living ghosts, as described in the comments.

3 Final Details

Clarification Page: Important corrections (hopefully few or none) and clarifications to the assignment (both Part I and Part II) will be posted on the Assignment 4 Clarification page: `http://www.teach.cs.toronto.edu/~csc384h/summer/Assignments/A4/a4_faq.html`. *You are responsible for monitoring the A4 Clarification page.* **Help Sessions:** There will be separate help sessions for Part I and Part II of this assignment. Dates and times for these sessions will be posted to the course website and to Piazza.

Questions: Questions about either part of the assignment should be posted on Piazza. Please be careful not to post spoilers! See: `https://piazza.com/utoronto.ca/summer2019/csc384`.

Getting Help: If you find yourself stuck on something, contact the one of the A4 TAs or an instructor for help. Office hours, help sessions, and the discussion forum are there for your support; please use them. If you can't make our office hours or a help session, let us know and we will work to schedule more. We want these assignments to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

GOOD LUCK!