Computer Science 384
St. George Campus

Monday, March 5, 2018
University of Toronto

Homework Assignment #3: Multi-Agent Pacman
— Part II —
Due: March 26, 2018 by 11:59 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.

Overview: Assignment #3 is comprised of two parts. In Part II, you will implement an expectimax agent, as well as an improved evaluation function. There will also be a set of short answer questions that accompany Part II, to be submitted in `writtenAnswers_p2.pdf`. This document details the requirements for Part II.

Total Marks: Part II has a total of 30 marks. (Recall Part I has a total of 40 marks.) Assignment #3 (Part I and Part II) has a total of 70 marks and represents 15% of the course grade.

What to hand in on paper: Nothing.

What to submit electronically: Edit your solution for Part I in `multiAgents.py` to address the questions in this Part II document. You may update your Part I solutions, but only the tasks outlined in this Part II document will be marked. Create a file `writtenAnswers_p2.pdf` containing your responses to the short answer questions.

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/winter/markus.html.

Clarification Page: Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment 3 Clarification page:
http://www.teach.cs.toronto.edu/~csc384h/winter/Assignments/A3/a3_faq.html.
You are responsible for monitoring the Assignment 3 Clarification page.

Questions: Questions about the assignment should be posted to Piazza:
https://piazza.com/utoronto.ca/winter2018/csc384/home.
If you have a question of a personal nature, please email the A3 TA, Kathy Ge, at kge at cs dot toronto dot edu or the instructor at csc384prof at cs dot toronto dot edu placing 384 and A3 in the subject line of your message.

Files you'll create and/or edit:

- **multiAgents.py** - Suitably augment this with your implementation of the search agents described in tasks 4 and 5 in this handout.
- **writtenAnswers_p2.pdf** - Where the requested written responses should be placed

Question 4: Expectimax (10 points + 8 points)

Minimax and alpha-beta are great, but they both assume that you are playing against an adversary who makes optimal decisions. As anyone who has ever won tic-tac-toe can tell you, this is not always the case. In this question you will implement the `ExpectimaxAgent`, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices. The only difference in implementation of Expectimax search and Minimax search, is that at a min node, Expectimax search will return the average value over its children as opposed to the minimum value.

As with the search and constraint satisfaction problems covered in this class, the beauty of these algorithms is their general applicability. To expedite your own development, we've supplied some test cases based on generic trees. You can debug your implementation on small the game trees using the command:

```
python autograder.py -q q4
```

Debugging on these small and manageable test cases is recommended and will help you to find bugs quickly. **Make sure when you compute your averages that you use floats**. Integer division in Python truncates, so that `1/2 = 0`, unlike the case with floats where `1.0/2.0 = 0.5`.

To see how the ExpectimaxAgent behaves in Pacman, run:

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

The correct implementation of Expectimax will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.

Explorative Questions – to submit (8 points):

1. Pacman should display more relaxed behavior when ghosts are nearby. In particular, if Pacman perceives that he could be trapped but might escape to grab a few more pieces of food, he'll at least try. Investigate the results of these two scenarios:

   ```
   python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
   python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
   ```

   Explain in **one or two sentences** why the `ExpectimaxAgent` might outperform the `AlphaBetaAgent` in this scenario. (2 points total)

2. Consider a game tree where the root node is a max node, and the minimax value of the tree is v_M. Consider a similar tree where the root node is a max node, but each min node is replaced with a chance node, where the expectimax value of the game tree is v_E. For each of the following, decide whether the statement is **True or False** and briefly explain in **one or two sentences** your answer.

   (a) **True or False:** v_M is always **less than or equal to** v_E. Explain your answer. (2 points)

   (b) **True or False:** If we apply the optimal **minimax** policy to the game tree with chance nodes, we are guaranteed to result in a payoff of at least v_M. Explain your answer. (2 points)

   (c) **True or False:** If we apply the optimal **minimax** policy to the game tree with chance nodes, we are guaranteed a payoff of at least v_E. Explain your answer. (2 points)

Question 5: Evaluation Function (12 points)

Write a better evaluation function for Pacman in the provided function, `betterEvaluationFunction`. Your evaluation function should evaluate states (in contrast to the function employed by your reflex agent, which evaluated actions). You may use any tools at your disposal for evaluation, including your search code from your last Pacman project. With depth 2 search, your evaluation function should clear the `smallClassic` layout with one random ghost more than half the time and still run at a reasonable rate. (To get full credit, Pacman should be averaging around 1000 points when he's winning.)
`python autograder.py -q q5`
Grading: the autograder will run your agent on the `smallClassic` layout 10 times. We will assign points to your evaluation function in the following way:

- If you win at least once without timing out the autograder, you receive 1 points. Any agent not satisfying these criteria will receive 0 points.
- +1 for winning at least 5 times, +2 for winning all 10 times
- +1 for an average score of at least 500, +2 for an average score of at least 1000 (including scores on lost games)
- +1 if your games take on average less than 30 seconds on the autograder machine. The autograder is run on the teach.cs machines which have a fair amount of resources, but your personal computer could be far less performant (netbooks) or far more performant (gaming rigs). You can use your teach.cs login to run your program on the teach.cs machines.
- The additional points for average score and computation time will only be awarded if you win at least 5 times.

Hints and Observations

- As for your reflex agent evaluation function, you may want to use the reciprocal of important values (such as distance to food) rather than the values themselves.
- One way you might want to write your evaluation function is to use a linear combination of features. That is, compute values for features about the state that you think are important, and then combine those features by multiplying them by different values and adding the results together. You might decide what to multiply each feature by based on how important you think it is.

HAVE FUN and GOOD LUCK!