CS 407
Programming 2 + Grad Programming 2
Due: Sunday of Week 13 11:59pm (via Blackboard)

This assignment may be completed individually or in groups of 2. Your code for file you should edit, `algorithms.py`) should be submitted via Gradescope. You should also submit a pdf writeup as described below, Please submit **only** these two files.
You may not discuss this assignment with anyone but course staff and your partner. If you want a partner and don't have one, post to Piazza. Both partners must contribute to all aspects of the assignment.

Goal

In this assignment you will program a number of learning algorithms we discussed in class. Graduate studens will also give some example scenarios where particular algorithms behave in particular ways and implement a variant of one of the algorithms.

Setup

Tools: Make sure Python 3 works on your computer system (`http://python.org/download/`).

Obtaining materials: Download the learning.zip archive from Blackboard and make sure you have both the relevant files: `algorithms.py` and `autograder.py`.

Assignment Overview

The assignment consists of 11 problems. It is expected that undergraduates will complete problems 1–5 and graduate students will also complete problems 6–11. (Optionally, undergraduates may complete them for extra credit.0 Grading will be based on how many points you score on the autograder and completion of writeup:

2.0 points 38–40 points on the autograder and Q1–Q5 of writeup adequately addressed.

1.5 points 31–37 points on the autograder and Q1–Q3 of writeup, plus either Q4 or Q5 depending on what you did, adequately addressed.

1.0 points 22–30 points on the autograder and Q1–Q3 of writeup adequately addressed.

0.5 points 15–21 points on the autograder and Q1–Q3 of writeup adequately addressed.

0.25 points 5–14 points on the autograder and some progress on writeup

0.0 points 0–4 points on the autograder or inadequate writeup

Completing problems 1–5 gives 25 points on the autograder, which is full credit for undergraduates.

Writeup

The goal of the writeup is for you to reflect on what you learned from the assignment. It is not intended to be lengthy (a paragraph per question totaling at most about 1 page for undergrads, 2 if you do the graduate assignment), but we are looking for evidence of real reflection. In particular, to receive full credit you need to have meaningful comments in response to each question. Undergraduates should address questions 1-3 of the following; students completing the graduate part should also complete 4 and 5 as appropriate depending on how much of it they completed.

1. Describe something important you learned about learning algorithms in the course of the assignment. Make sure to clearly address both what you learned and why you think it is important.
2. Discuss a non-trivial way one of your initial implementations was incorrect and what you had to change to fix it. (This should be a substantive error you had in the underlying logic, not a simple typo or syntax error.)
3. Suggest a change or improvement you could make to one of the learning algorithms you implemented and discuss when and why it might be helpful.
4. What is something you found unexpected or interesting about the behavior of one of the learning algorithms you discovered while working on one of the scenarios or during further exploration? Explain both your specific observation and what was unexected or interesting about it.
5. As discussed below, optimistic regret matching leads to a different form of convergence. Discuss an example of when and why the distiction could be important.

Problem details

Each problem is marked in algorithms.py, which is the only file you should edit. The problems and their point values for the autograder are as follows.

1. Expected Utilities / Values (5 points)

   The assignment starts with a function `expectedValues` that will be useful in implementing your learning algorithms. Given a game and your opponent's strategy (which might be mixed), calculate the expected utility / value of each action you could take. The game will be represented by a matrix (i.e. a list of lists) that gives your payoffs. So for prisoner's dilemma this would look like G=[[-1,-9],[0,-6]]. E.g., if you play C (strategy 0) and your opponent plays D (strategy 1), your utility is G[0][1]=-9. The opponent's strategy will be a list with the probability of each action, e.g. [0.5,0.5]. Note that this assumes we have a single opponent, which will be the case for all problems on the assignment. This function, like others in the assignment returns [0.0] so that the autograder will run without errors. You should replace it with a proper return value! In prisoner's dilemma, the best reply is always [0.0,1.0] because D is a dominant strategy.

2. Best Response Dynamics (5 points)

   Our first real learning algorithm! The function `bestResponseDynamics` should implement one iteration of best response dynamics as described in the slides. Here the player to update has already been chosen and supplied with its payoffs for the game and the opponent's strategy (exactly as in the previous problem). So you just need to return the a strategy that is a best reply. The autograder will even accept mixed strategies if you want, as long as they are best replies.

3. Fictitious Play (5 points)
4. Smoothed Fictitious Play (5 points)

5. Regret Matching (5 points)

   Each of these learning algoriths maintains state, so they are implemented as classes. Each has an `__init__` method that sets of the state you will need. You shouldn't need to change this method, but can if you think it will make things easier for you. The `updateStrategy` method should perform one iteration of the loop that takes in the opponent's most recent strategy and calculates the strategy the algorithm will play next per the slides. Recall that we can use the opponent's strategy in the first round to initialize the algorithm, so your code should behave sensibly even if the "strategy" that gets fed in is not a probability distribution! For regret matching we have provided an additional helper method you may want to implement.

   This is the end of the assignment for undergraduates. The rest is for graduate students or extra credit for undergraduates.

6. (2 points)

7. (2 points)

8. (2 points)

9. (2 points)

10. (2 points)

    These five problems ask you to identify scenarios where your learning algorithms behave in the manner described. For each player you will be able to specify that player's payoff matrix and the value that will get passed to each player is the first round as the opponent's strategy (the player's "prior"). For problems 6 and 7 you only need to supply the prior because the game is already specified. The values you need to provide have been set to None and should be replaced by your solution.

11. Optimistic Regret Matching (5 points)

    In problem 10 you found a scenario where regret matching failed to have it's strategy converge to a mixed Nash equilibrium. This problem has you implement a variant of regret matching which will be better behaved. It uses "optimistic" updates, a technique whose theoretical properties are an area of active research. Specifically, where ordinary regret matching uses the update rule

    $$\mathit{regretSums} = \mathit{regretSums} + \mathit{currentRoundRegrets}$$

    optimistic regret matching uses the update rule

    $$\mathit{regretSums} = \mathit{regretSums} + 2\mathit{currentRoundRegrets} - \mathit{lastRoundRegret}$$

    That is, instead of updating its regret sums normally it updates them twice as fast but then undoes this extra part on the next update. With this change, you should see convergence of the strategy to a mixed Nash equilibrium (not just the average strategy).

Autograder Instructions

You can run the autograder with the command `python autograder.py`. Typically you will only want to run the problem you are current working on, which you can do with the `-q` flag. For example, `python autograder.py -q1` will run the autograder just on problem 1.