COS 445 - Strategy Design 3

Due online Monday, April 1st at 11:59 pm

Instructions:

- **You may not take late days on the Strategy Designs**. If it helps, think of the Strategy Designs as being due on Friday, except we have given everyone three free late days.
- You should aim to work in a team of two, but you are allowed to work alone or in a team of three. Your team should submit a *single* writeup, using the team feature on codePost. You should also submit a *single* code solution, using the team feature on TigerFile.
- Your goal in this assignment, and all strategy designs, is to **maximize your absolute payoff**. Your goal is *not* to outperform other submissions. Any justification you provide should explain why your strategy maximizes your absolute payoff (and only this kind of justification will contribute to your writeup score). Your code score is computed solely based on how you perform in comparison to "if your solution were replaced by a course staff submission" (that is, you are not being compared to a strategy that you play against, so it does you no good to harm the strategies you play against).[1]
- You are allowed to engage with other teams over Ed or in person (but this is neither encouraged nor discouraged). If this is part of your strategy, you should discuss what you did and why you did it in your writeup. You are allowed to coordinate with other teams, or trick other teams. You are *not* allowed to promise other teams favors (e.g. monetary rewards) or threaten punishment outside the scope of this assignment. For example, you are allowed to promise "if your code does X, our code will do Y." You are not allowed to promise "if your code does X, I will buy you a cookie." If this is part of your strategy, your justification should explain why it will help you *on this assignment*.
- Please reference the course collaboration policy here.
- Please reference the following document for further detail on how these assignments are evaluated: GradesForStrategy.pdf.
- This assignment is open-ended, **please ask questions on Ed to clarify expectations as needed**.

Reminder!

Please read the instructions at GradesForStrategy.pdf to better understand how the strategy design assignments are graded (which in turn should clarify how to answer the prompts).

[1]This claim is *slightly* inaccurate in order to save runtime while computing code scores. You are free to read the full details **here**, and to ask for clarification on Ed, but I'm comfortable advising that the best way to optimize your code score is to just optimize your own payoff and not to overthink subtleties in precisely how the code scores are computed.

Ultimatums with Reputations (25 points)

Your best friend from high school just finished their MBA and is now entering the corporate world. Their MBA taught them to treat every interaction like an economic transaction, and to use these interactions to project a reputation which might help their future transactions. Your friend heard you were taking COS 445 and asked for your help maximizing their payoff in a series of formally-posed two-player games.

Despite your skepticism that your friend's mentality is the right approach to business, you agree to help (they are after all your best friend, and this is surely not the worst idea they've had which required your help). In this challenge, you'll play the following game:

Setup:

- You will play a sequence of *rounds*. Every round consists of a simple two-player game (defined below).
- You will be matched with a random partner in each round who you did not play before.
- You will see some information about your partner's play in previous rounds before deciding what to do.

Round:

- Each round you will play the Ultimatum game with your partner. The following three bullets define the Ultimatum game.
- One of you will be labeled as Alice, the other will be labeled Bob.
- First, Alice proposes to Bob any number in $A \in \{0, \dots, 99\}$.
- Bob may then accept or reject. If Bob accepts, Alice gets payoff A and Bob gets payoff $200 - 2A$. If Bob rejects, both Alice and Bob get payoff 0.

History:

- You will learn some information about your partner's past play before deciding what to do (defined in the following four bullets).
- You will see, for their most recent round as Alice: what Alice proposed and whether it was accepted or not.
- You will also see, for their most recent round as Bob: what Alice proposed and whether it was accepted or not.
- If any such previous rounds don't exist, the default will be $(-1, -1, \mathit{accept})$.
- You will also see the total number of rounds as Bob where they accepted, and the total number of rounds as Bob where they rejected.

Total Payoffs:

- Your total payoff will sum your payoff in every round. Remember that your goal is to maximize your payoff so that you perform well in comparison to "if your strategy were replaced by a course staff strategy", not to perform better than your classmates.

Code it up according to the specifications below, and write a brief justification. Your strategy will implement the Ultimatum interface provided in `Ultimatum.java`, which requires the following four methods, as documented in `Ultimatum.java`:

- The first function should take as input a history in the format described above: (Feel free to rename these variables in your implementation!)
  ```
  public void setup( int opponentSawAliceSaid,
  boolean opponentAsBobAccepted, int opponentAsAliceSaid,
  boolean opponentAsAliceWasAccepted,
  int opponentAsBobAcceptedCumulative,
  int opponentAsBobRejectedCumulative);
  ```
- The second function, as Alice, should output an integer $A \in \{0, \ldots, 99\}$ corresponding to your proposal to (try to) keep for yourself:
  ```
  public int propose();
  ```
- You should also define a function to output accept or reject given an offer A. Note that A will always refer to the payoff that Alice keep:
  ```
  public boolean accept(int aliceProposal);
  ```

We've also provided two sample strategies, `Ultimatum_levelzero.java` and `Ultimatum_smweinberg.java`, and the usual construction of testing code. Both of these strategies are very poor strategies, but they are intended to illustrate legal submissions.

Your file must follow the naming convention `Ultimatum_netID1.java`, where `netID1` is the netid of the primary submitter. We have updated the testing script so it will not accept misnamed strategies.

Your writeup should provide an overview of the main ideas in your code (remember that we also have your code — so you don't need to provide pseudocode or a step-by-step description of your algorithm), and justify why you think it will perform well, in addition to concrete answers to parts **a** and **b**.

Penalties may be given for code which does not compile, throws exceptions, or violates assertions. Remember to test your code with settings besides the default settings of our testing environment. Extra credit may be awarded for reporting substantive bugs in our testing code.

Also submit a single PDF file, containing answers to the following three prompts. Recall that your grade for part c is the maximum of your grade on the writeup and your grade for your strategy's performance (any penalties are subtracted after taking the maximum).

Part a (10 points)

What is a subgame-perfect Nash for the Ultimatum game? Describe another Nash equilibrium that is not subgame-perfect.

Hint: Recall that a *strategy* for Alice must select an action at *every node* where Alice acts (and a strategy for Bob must select an action at every node where Bob acts). You may want to think explicitly what the nodes are in this game, and what potential strategies are.

Part b (10 points)

If the strategy design were modified so that *none of your behavior as Bob was ever passed on.* What would you do as Bob? Knowing this, what would you do as Alice?

Part c (15 points)

Provide a brief justification for your strategy. Focus on convincing the grader that it is a good strategy, by explaining the main ideas and why you chose this strategy. You should aim to keep this under one page. This will not be strictly enforced, but the grader may choose not to read beyond one page. You should not think of this merely as a documentation explaining only *what* your code does. Instead, try to imagine that its purpose is to convince your friend *why* they should adopt your strategy.

Note: For all Strategy Designs, it is a good idea to explicitly reason about how you believe your classmates will behave. For this Strategy Design in particular (because the game is so simple), it may be a better-than-usual idea to explicitly reason about how you believe your classmates will behave. Note that you are allowed/encouraged to discuss your strategy publicly/privately with other teams. You are also allowed to share analysis (but you still may **not** share code). Of course, each team is still responsible for writing their own justification, and ensuring that it is convincing (you are allowed to reference any public/private communication in your justification, but you are ultimately responsible for writing a convincing justification).