COS 445 - Strategy Design 3

Due online Monday, April 6tt at 11:59 pm

Instructions:

- 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* code solution, using the team feature on TigerFile.
- Your goal in this assignment, and all strategy designs, is to **maximize your absolute payoff**.
- You are allowed to engage with other teams over Ed or in person (but this is neither encouraged nor discouraged). 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."
- This assignment is open-ended, **please ask questions on Ed to clarify expectations as needed**. As with PSets, recall that SDs are not graded (but count towards your engagement credit).

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 $A \in \{0, \ldots, 99\}$.
- 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 $(-1, -1, accept)$.
- 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.

Your total payoff will sum your payoff in every round.

Code it up according to the specifications below.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 keeps:
```
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.

Submissions that do not compile, throw exceptions, or violate assertions may be disregarded (i.e., not receive the corresponding engagement credit).. Remember to test your code with settings besides the default settings of our testing environment.

The following questions are optional, but we recommend thinking them through to better understand the problem and design good strategies.

Part a

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

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?