The combinatorial auction design for the final project consists of two phases. The first is the combinatorial clock-like auction we simulated in Lab 9. Recall that the outcome of this auction is the one that maximizes revenue, maximizing over all earlier rounds. The second is a "last and final round": i.e., bidders have one last chance to win their preferred goods. The outcome of the auction is whichever of the two phases yields higher revenue.

The spectrum auction designs for this last and final round involved proxy-bidding agents, and the CORE equilibrium concept. We are choosing to run a much-simplified last and final round, which works as follows: It is a sealed-bid auction in which bidders submit an XOR bid consisting of at most three atomic bids, \((x, b)\). The value \(b\) associated with bundle \(x\), must at least meet a reserve price, which is the sum of the prices of the goods in the bundle \(x\) discovered during the first phase. The outcome of this second phase is a welfare-maximizing allocation at stated prices (i.e., it is a first-price auction, with reserve prices to ensure that bidders do not shade their bids too much).

An obvious and important difference between this two-phase auction and the CCA-like of Lab 9 is the presence of a second phase. What is the point of the second phase; why was it added? The intuition for the second phase is that with it, the first phase can facilitate “price discovery”, so that bidders can get an idea about which goods are the most competitive. Then, already knowing something about prices, they can place more informed bids in the second phase on goods which maximize their utility.

Sounds promising, but life is not so simple. Why should bidders bother to bid in the first phase? In particular, why shouldn’t a bidder sit back and watch all the others bid to learn about their (private) preferences, and then use that information in the final round, without sharing any knowledge whatsoever about its own preferences? In fact, it probably should; indeed, all bidders should. So in this auction design, it is more important than before to incentivize sincere bidding in the first phase; otherwise, the phase is useless.

More specifically, the first phase can be informative only if bidders do not bid too low, and likewise, do not bid too high. Ideally, final prices in the first phase would at least somewhat resemble final prices in the second.

In the CCA, bidders are incentivized to bid in all rounds where their preferred bundle is not empty, because the auction can end at any time. We retain that rule in this two-phase auction, by allowing for the possibility that any outcome in the first phase be the outcome of the entire auction. This possibility is strong enough to discourage late bidding in the auction (i.e., bidders don’t bid too high), because it becomes more likely than an outcome of the first phase be the outcome of the entire auction when higher bids are placed.
But, if no one bids in the first phase at all, or if only very few bids are placed, it is very unlikely that the revenue of the revenue-maximizing outcome in the first phase will exceed that of the second phase, so this possibility is not strong enough to encourage early bidding. Instead, we are left with the problem of incentivizing bidders to bid in the first phase.

As a first attempt, we created a hack, to try to incentivize early bidding, namely: a bidder’s bid price on a bundle in the second phase could not exceed 1.5 (or 1.1, or 2, or etc.) times the maximum amount that bidder bid on that bundle in the first phase. But this hack was unsatisfying, because it imposed an artificial cap on revenue.

A better solution would tie together the first and second phases by somehow limiting the quantity and/or quality of the goods that an agent can bid on in the second phase to something that depends on the quantity and/or quality of the goods it bid on in the first. To this end, we have substituted an eligibility rule for the earlier hack, which constrains the extent to which bidders can bid in the second phase to depend on their behavior in the first phase. Recall that bidders will submit an XOR bid with at most three atomic bids in the second phase of the auction. The number of licenses that each of a bidder’s atomic bids can contain is at most the number of licenses contained in their reported demand set in the revenue-maximizing round of the first phase.

There are two things to note about this eligibility rule: 1. A bidder need not win anything at all in the revenue-maximizing round of the first phase to be eligible to bid in the second phase. They need only have submitted a valid bid. Their eligibility depends on the contents of that valid bid, not on whether they were allocated anything during that round. 2. Bidders will be informed of their eligibility in the second phase at the end of the first phase/start of the second.

Pseudocode

Here are all the rules, more formally, with parameter settings hard coded in this pseudocode:

```plaintext
// Initialization
Initialize all (actual) prices to 0: i.e., \( p^0(j) = 0 \), for all goods \( j \).

Initialize all deliberation prices to 0: i.e., \( q^0(i,j) = 0 \), for all goods \( j \) and all bidders \( i \).

Initialize the current round \( k \) to 0

// Valuations
Each bidder receives an XOR bid representing their valuation, with a number of atomic bids drawn from a normal distribution with mean 50 and standard
```
deviation 15. Each atomic bid consists of a bundle whose size is drawn from a
normal distribution with mean 30 and standard deviation 10.

// The first phase: an ascending-auction
do {
  Increment $k$ by 1

  Define deliberation prices at round $k$ as follows:
  • for all goods $j$ bidder $i$ is currently winning, $q^k(i,j) = p^{k-1}(j)$
  • for all goods $j$ bidder $i$ is not currently winning, $q^k(i,j) = p^{k-1}(j) + \varepsilon$

  Wait 10 seconds.

  Get demand sets from all bidders at their respective deliberation prices.

  The server converts each bidder’s demand set into a bid, which is a tuple
  consisting of the bidder’s reported favorite bundle and that its implicit
  value for that bundle. These implicit values are computed by tallying the
  prices of all the goods in the favorite bundle, using the bidders’
  deliberation prices.

  The server then constructs a set of valid bids, namely those that satisfy
  the revealed preference rule.

  If the set of valid bids is non-empty {
    Run winner determination to find a revenue-maximizing allocation.

    For each winning bundle in the allocation, set the (actual) prices of
    the goods in that bundle to be the deliberation prices of the winning bidder:
    i.e., if $S$ is a winning bundle, $i^*$ is the winner of $S$, and $j$ is a good in $S$,
    then $p^*(j) = q^*(i^*,j)$.

    For all goods $j$ that are unallocated, $p^*(j) = p^{k-1}(j)$.

    Notify each bidder of their own allocation, and all prices.
  }
} until the set of valid bids is empty.

Find the revenue-maximizing round in the ascending-auction phase.

// The second phase: a last and final round
Inform bidders of their eligibility, namely the number of goods in their demand set, if it was valid, during the revenue-maximizing round of the ascending-auction phase.

Set reserve prices on each bundle to be the sum of the prices of the items in that bundle, where those item prices are the prices discovered in the revenue-maximizing round of the ascending-auction phase.

Wait 60 seconds.

Get from each bidder one XOR bid comprised of at most three atomic bids, with bid prices that are at least the reserve price for that bundle.

Keep only those XOR bids that meet the eligibility requirement.

Run VCG on the XOR bids submitted, and store the allocation.

For each winner, set its (bundle) payment equal to its bid price.

// The outcome
The final outcome is whichever outcome yields greater revenue, either that of the first phase or of the second.

Server

The VCG solver uses CPLEX, so when you run the server, you must pass it the following VM argument:
-Djava.library.path=/local/projects/cplex/cplex12.6/cplex/bin/x86-64_linux/

Agent

Your agent for the final project should extend FinalProjectAgent in labs.finalproject. You can peruse a sample agent at labs.finalproject.FinalProjectDemo. You can test against the server at labs.finalproject.FinalProjectServer.

Your job is to implement:
- onSimpleOpenOutcry, in the first phase (for the CCA-like auction).
- onSimpleSealed, in the second phase (for the VCG-like final round)

Be sure to check the payment rule to make sure that you are bidding in the auction that you intend to (e.g., market.getPaymentRule().equals(PaymentType.VCG)).
API

There is a slight change/improvement in the API for the final project, beyond what was available in Lab 8. Specifically, what was the `getHighBid` method has been replaced with a `getMarketState` method, which includes more information than merely the standing high bid:

- `MarketState getMarketState(FullType type)`
  - This method takes as input a `FullType` and returns the market state for that `Tradeable`, which, in general, consists of the agent ID of the agent that it is currently allocated to and its current price. In our combinatorial auction games, however, the information revelation policy is such that the agent ID is set to `null` when your agent is currently winning the `Tradeable`, but prices are always available. These prices are the reserve prices (if any) in sealed-bid auctions and the current high bid in ascending auctions.

We have also added a method to the API which allows your agent to ping the server for samples of other agents’ valuations: i.e., for XOR bids drawn from the other agents’ valuation models. Note that these samples tell you nothing about the particular valuations that have been realized by any of the competing agents, but as they are drawn from the same model as those realizations, they are representative of the agents’ true valuations.

- `Map<Set<FullType>, Double> sampleValuation(int agentNum)`
  - Returns a sample XOR bid for `agentNum`. The `agentNum` argument is optional; omit it if you want a generically noisy bid bundle.

Lastly, eligibility will be broadcast to agents at the start of the second phase of the auction. You can find your agent’s eligibility status here:

- `int getEligibility()`
  - Returns the maximum number of goods your agent can bid on in each of the atomic bids that comprise the XOR bid it submits in the last and final round (second phase of the auction).