In this lab, we will play two two-day variants of the AdX Game. In the first, there will be one campaign that spans the two days, and in the second, there will be two one-day campaigns.

The first variant is very similar to the One-Day variant we played in Lab 6. The only difference is that the single campaign given to an agent at the start of the game lasts for two days instead of one. So your agent can/should bid on users/impressions for this one campaign over the course of two days. At the end of the first day, your agent will receive feedback from the server detailing how many impressions the agent won that day, and the total cost of those impressions. Your agent’s strategy can/should take into account this information!

In the second variant, agents will be given an initial campaign that lasts for just one day (just like in the One-Day variant), and then, contingent upon the agent’s performance in fulfilling its initial campaign, which will be measured by its **quality score**, it will be given a second campaign, whose budget will be discounted by this quality score. The quality score is a number between 0 and 1.38442, where 0 denotes very low quality (in case the agent acquired 0 impression for the first campaign), 1 denotes very good quality (in case the agent acquired the number of impressions needed), and 1.38442 denotes perfect quality (in case the agent acquired many many impressions, above and beyond the required number of impressions). Your agent is thus faced with the usual task of fulfilling its first campaign profitably, but at the same time in such a way as to earn a valuable second campaign!

**API for AdX Two-Days Games**

**TwoDaysBidBundle Object**

Once again, to avoid the communication overhead required to conduct each ad auction in real time (each day there are 10,000 simulated users!), the agents use a **TwoDaysBidBundle** object to communicate all their bids to the server at once.

The constructor for this **TwoDaysBidBundle** object takes 4 parameters:

1. **Day**: the day for which the bid is placed, either 1 or 2.
2. **Campaign ID**: the ID for the campaign you are bidding on.
3. **Day Limit**: a limit on how much you want to spend in total on that day.
4. **Bid Entries**: a collection of **SimpleBidEntry** objects, which specify how much to bid in each market segment.

A **SimpleBidEntry** object has 3 parameters:
1. **Market Segment**: there are a total of 26 possible market segments.
2. **Bid**: a double value.
3. **Spending Limit**: a double value that represents the maximum value the agent is willing to spend in this market segment.

For examples of how to create a `SimpleBidEntry`, please refer to last lab’s documentation. Assume you have already created one, called `bidEntries`. You could then create a `TwoDaysBidBundle` for your campaign on the first day that includes `bidEntries`, and limits total spending to the budget, as follows:

```java
BidBundle bidBundle = new TwoDaysBidBundle(1,
    this.myCampaign.getId(),
    this.myCampaign.getBudget(),
    bidEntries);
```

**TwoDaysOneCampaign Class**

For the first variant, your job is to extend the abstract class `TwoDaysOneCampaignAgent` by implementing the `getBidBundle(int day)` method. This method takes as input a day, and return a `TwoDaysBidBundle` containing the agent’s bids for the given day. The class has three attributes:

1. **Campaign** object, `myCampaign`, which contains the campaign assigned to the agent at the beginning of the game. This campaign lasts for two days.
2. **Integer**, `reachDay1`. This integer is populated on the second day, and denotes the number of impressions procured by the agent on day 1 for `myCampaign`.
3. **Double**, `costDay1`. This double is populated on the second day, and denotes the total spent on impressions on day 1 for `myCampaign`.

**TwoDaysTwoCampaigns Class**

For the second variant, your job is to extend the abstract class `TwoDaysTwoCampaignsAgent` by implementing the `getBidBundle(int day)` method. This method takes as input a day, and should return a `TwoDaysBidBundle` containing all the agent’s bids for the given day. The class has two attributes:

1. **Campaign** object, `firstCampaign`, which contains the campaign assigned to the agent for the first day of the game. This campaign lasts for a day.
2. **Campaign** object, `secondCampaign`, which contains the campaign assigned to the agent for the second day of the game. This campaign also lasts for a day. This object is only available on the second day and it is `null` otherwise.
Quality Score

Let $R$ be the total reach of a campaign $C$. The quality score $Q_C(x)$ of the agent owning $C$ as a function of the number of procured impressions, $x$, is computed as follow:

$$Q_C(x) = \left(2 / \frac{4.08577}{R}\right) \left(\arctan\left(\frac{4.08577}{R}x - 3.08577\right) - \arctan(-3.08577)\right)$$

Note that $Q_C(0) = 0$, $Q_C(R) = 1.0$, and $\lim_{x \to \infty} Q_C = 1.38442$.

If your agentsprocures no impressions, your quality drops to zero which means the budget of your second campaign is zero (you have no reward for fulfilling that campaign!). On the other hand, if you attain a quality score greater than 0.0, then your second campaign will have a positive budget, which could potentially lead to positive profit.

Let $C_1$ and $C_2$ be the first and second campaigns assigned to an agent in the Two-Days, Two-Campaigns game variant. The reach and budget of the first campaign are randomly drawn in the same way as in the One-Day variant. The reach and budget of the second campaign are also randomly drawn, but the budget is discounted by $Q_{C_1}(x)$, where $x$ is the number of impressions acquired for the first campaign.

Quality Score Example

The following plot depicts the quality score function $Q_C(x)$ of a campaign $C$ with reach of 1000.

![Plot]

This plot shows that the value of obtaining the first few impressions to fulfill a campaign is relatively low compared to the value of obtaining the middle and final impressions.