1 Introduction

This lab is a continuation of our exploration of the TAC Ad Exchange (AdX) game. Once again, you will be building an agent for the AdX Game. However, you will be playing a different, slightly more complicated, variant of the game than in the previous lab.

In this version of this game, your agent will compete in AdX auctions over two consecutive days, and will receive a different campaign each day such that the budget campaign your agent receives on the second day depends on its success fulfilling its campaign on the first day. This success will be measured by a metric called **quality score**. Your agent will still aim to maximize its profit.

The complexity we introduce here will help you gain a better understanding of the AdX game, and the kinds of agent strategies that can be successful. You will be given the opportunity to apply the knowledge you gain in this lab in the AdX final project, which will consist of a still more complicated AdX game.

2 Logistics

This lab will be conducted remotely. You will download the stencil code, fill it out, and submit via the course handin script, after which the TAs will run a competition, and inform the class of the results by posting a leaderboard on the course web site.[1]

Because we do not have in-person lab TAs to answer questions, please feel free to ask questions on Piazza.

This lab is due **Wednesday, April 8 at 3:00 PM EDT**. Competitions will then run hourly all week long and the leaderboard will be updated accordingly. You may resubmit your agent at any time, through **Friday, April 10 at 3:00 PM EDT**.

Just like in previous labs, we encourage pair-programming, especially if you are not very familiar with Java. Although pair-programming is undoubtedly more difficult remotely, we nonetheless encourage you to reach out to other students in the class to find a partner. For this purpose, we have left open Piazza's "Search for Teammates" functionality. **Please limit your group size to two.**

3 Game Description

This game is very similar to last week's One-Day AdX game. If you'd like to review the details of the AdX game, we recommend re-reading Lab 7.

The main difference introduced in the two-day variant is the following:

In the Two-Day 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 impressions for the first campaign), 1 denotes very good quality (in case the agent acquired the campaign's reach in its entirely), and 1.38442 denotes perfect quality (in case the agent acquired many many impressions, above and beyond the campaign's reach). 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!

[1] These leaderboards will name leading agents, not their developers, unless the students agree to have their names released.

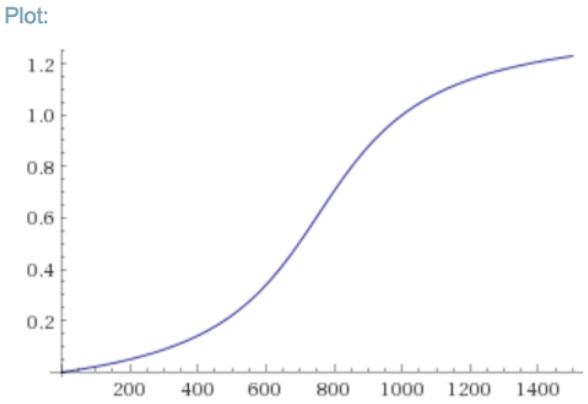
This two-day game has a built-in trade off: bid too high on the first day and your agent won't be profitable on the first day, but it will earn a high quality score, giving it a chance to be very profitable the second day; bid too low on the first day and your agent can be more profitable on the first day, but if it does not fulfill its campaign's reach, it will earn a lower quality score on the first day, and hence a lower budget for its campaign on the second day.

4 Quality Score

Let R be the reach associated with a campaign C. The quality score $Q_C(x)$ earned by an agent seeking to satisfy C who procures x impressions suitable for C is computed as follows:

$$Q_C(x) = \frac{2}{a}\left(\arctan\left(a\left(\frac{x}{R}\right) - b\right) - \arctan(-b)\right),$$

where $a = 4.08577$ and $b = 3.08577$. The following plot depicts the quality score function $Q_C(x)$ of a campaign C with reach of $R = 1000$. Note that $Q_C(0) = 0$, $Q_C(R) = 1.0$, and $\lim_{x\to\infty} Q_C(x) = 1.38442$.



This plots shows that the value of obtaining the first few impressions on the road to fulfilling a campaign is relatively low compared to the value of obtaining the middle and final impressions.

If your agent procures zero impressions on its first campaign, then its quality score drops to zero which means the budget of its second campaign will be identically zero (it will gain zero revenue for fulfilling that campaign!). On the other hand, if it attains a quality score greater than 0.0, then its second campaign will have a positive budget, which will hopefully lead to a positive profit.

The campaign distribution in the Two-Day, Two-Campaign game is the same as in the One-Day game. That is, the reaches and budgets are randomly drawn as in the One-Day variant (see Lab 7, Appendix A). On the second day, however, the second campaign's budget is multiplied by $Q_{C_1}(x)$, where x is the number of impressions acquired by the first campaign, C_1.

5 API for AdX Two-Days Games

5.1 `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 a `Set<SimpleBidEntry>`, called `bidEntries`. You could then create a `TwoDaysBidBundle` for your campaign on the first day that includes `bidEntries`, and limits total spending to your campaign's budget, as follows:

```
TwoDaysBidBundle bidBundle = new TwoDaysBidBundle(
    1,
    this.getFirstCampaign().getId(),
    this.getFirstCampaign().getBudget(),
    bidEntries);
```

On the second day, you could do something similar, but setting `day = 2`, and the campaign to `this.getSecondCampaign()`.

5.2 `MyTwoDaysTwoCampaignsAgent` Class

You should implement your agent strategy in `MyTwoDaysTwoCampaignsAgent.java`.

This class should extend the abstract class `TwoDaysTwoCampaignsAgent` by implementing `getBidBundle(int day)`. 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 methods for accessing its campaigns:

1. `getFirstCampaign()`, which gets the campaign assigned to the agent for the first day of the game. This campaign lasts for a day.
2. `getSecondCampaign()`, which gets 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; it is `null` otherwise.

5.2.1 Agent Naming

As in previous labs, your agent needs a name. You can give it a name near the top of `MyTwoDaysTwoCampaignsAgent.java`.

Please complete this form as part of your submission. The agent name that you select will appear on the leaderboard, so keep that in mind if you'd like your scores to be anonymous to the class.

To prevent name overlap, the TAs will notify you if you choose a name that's already taken by another agent.

5.2.2 Helper Functions

The `adxgame` dependency contains support code to iterate over market segments. Concretely, `MarketSegment` is implemented as an `Enum`, so to iterate over all of them, you can do something like:

```
for (MarketSegment m : MarketSegment.values()) { ... }
```

The static function `marketSegmentSubset(MarketSegment m1, MarketSegment m2)` returns a boolean indicating whether `m2` is a subset of `m1` (N.B. market segments are subsets of themselves).

6 Code Installation and Submission

The installation, testing, and submission process is identical to that of the last lab. Regardless, we urge you to read and follow all the steps carefully, to be sure that our remote competitions run smoothly.

6.1 Installation

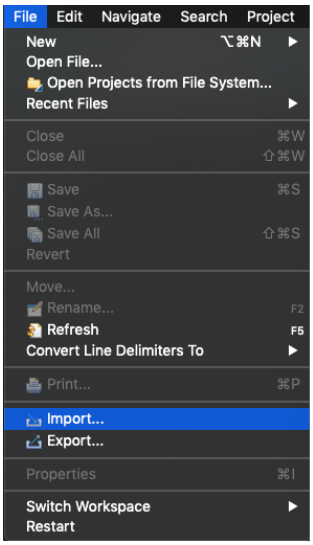
Download the stencil code from the course website. You will notice that instead of being a single `.java` file, the stencil code is now an entire Java project, complete with a package structure and a file called `pom.xml`, along with a few python scripts.

We will be using the Apache Maven build system for this project. Put simply, Maven allows us to export our Java projects into runnable JAR files in a standardized fashion (as defined in `pom.xml`, which acts as a configuration file for Maven).

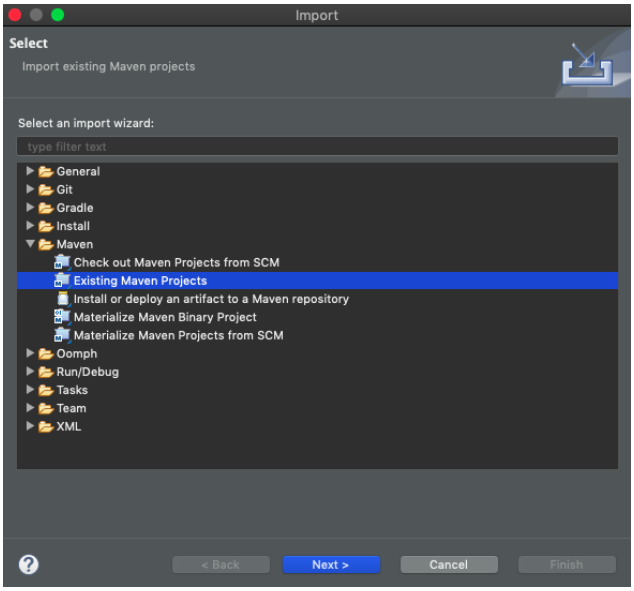
If you do not already have Maven installed, please download and install it. If you are working on a department machine or via FastX, you will already have Maven installed, which you can verify by running `mvn -version`.

If you have not installed Eclipse on your machine, you should download that as well.

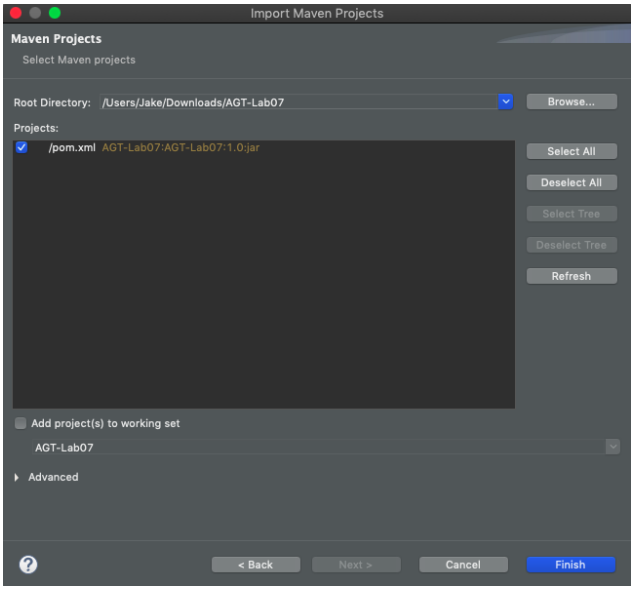
Once you have installed Maven and unzipped the stencil code, open Eclipse and select **File** → **Import**.



Then, select **Maven** → **Existing Maven Project**.

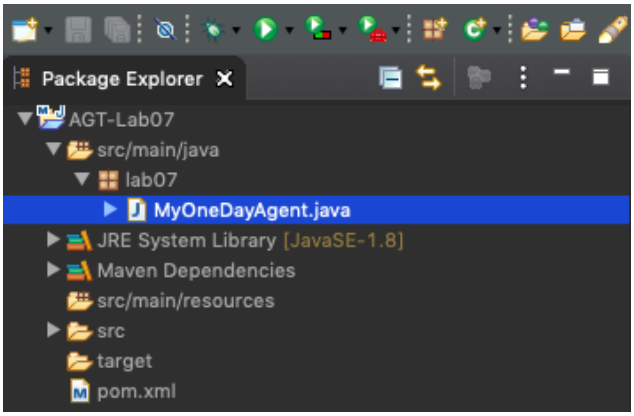


Finally, navigate to, and select, the stencil project and make sure `pom.xml` is selected. Select **Finish**.



This will create an Eclipse project for this lab.

Your task resides in the file `MyTwoDaysTwoCampaignsAgent.java` in the package `lab08` under `src/main/java`. Feel free to create and use any other packages, classes, etc. within the project, as long as you do not change the location or name of `MyTwoDaysTwoCampaignsAgent.java`.



6.2 Submission

You will be submitting your code via the course handin script.

First, if you are not already working on a department machine or via FastX, you will need to transfer your files to the department filesystem. We have provided a utility to do this upload for you. You are free to transfer the files any way you'd like—this utility is just an attempt to simply your life.[2]

To use this utility, from your project's root directory, run `python3 upload.py`.[3] It will transfer your files to `/course/cs1951k/student/<cslogin>/AdXLab2`.

Next, SSH to Brown, and navigate to your project's root directory (if you used the upload script, `/course/cs1951k/student/<cslogin>/AdXLab2`).

To make sure that your agent will work as intended, we have provided a script that launches the server, along with your agent and 9 opponent bots. You should run this script to make sure your agent connects to the server properly, doesn't crash, and properly submits its bids.

The command is `/course/cs1951k/pub/2020/AdXLab2/test`. It should be run from your project's root directory, and it runs 5 iterations of the game, and should take about one minute, or less (but it may take a bit longer the first time if Maven needs to download any packages).

If it succeeds, you'll be able to see the results of your game against the bots. As such, feel free to also use this script to debug your agent's strategy.

Finally, run `/course/cs1951k/pub/2020/AdXLab2/submit` to submit your agent. We only store your most recent handin, so if you wish to update your agent and submit a newer version at some later date, just repeat these steps and your submission will be replaced.

7 Competition Details

Once the due date for the lab has passed, the TAs will begin running competitions between your submitted agent and those of your classmates. A competition will be run every hour on the hour, and will consist of 100 simulations of the two-day, two-campaign AdX game. In each simulation, your agent will play against 9 other agents, chosen at random (so your strategy can assume 9 competitors).

The results of these competitions will be posted on a leaderboard on the course website. We will be updating the leaderboard regularly for the rest of the week, even after the lab is due. At any time during the week, you are free to improve your agent and resubmit. To guide your improvements, you will have access to your agent's logs (which is everything your agent prints during each game). These logs will be available at `/course/cs1951k/student/<cslogin>/logs/AdXLab2.zip`.

[2] Well, not really. Just this very very small part of it.

[3] To run this script, you will need Python 3. You may also need to install it, as well as the `pysftp` library, via `pip`.