Documentation

# Bolt Auction Challenge

Last Updated on 2nd April 2021

# Contents

# Copyright Notice

The Bolt Auction Challenge is a part of ongoing research work. No part of this documentation or codebase should be shared outside of the course.

# 1

# About the Challenge

The Bolt Auction Challenge is a unique challenge that requires designing an interactive bot.

## 1.1 Simulator

The Bolt Auction Challenge is run on a custom-designed fully asynchronous event-driven simulator.

# 2

# Notation

| Symbol | Description |
|---|---|
| $\eta$ | Minimum bid increment |
| $n$ | Number of Players |
| $[n]$ | Set $\{1, 2, \ldots, n\}$ |
| $\delta$ | Random variable denoting delay |
| $b_i$ | $i^{\text{th}}$ Player's bid value |
| $T$ | Random variable denoting the duration of the auction |

Table 1: List of symbols used in the document

- $\delta$ is sampled from a Normal Distribution with mean 5 seconds and standard deviation 2 seconds.
- $T$ is sampled from an Exponential Distribution with mean 30 seconds
- $\eta = 1.125$ (Constant)

# 3

# Auction

The auction consists of an object valued at 1 unit with multiple players participating in it.

## 3.1 Allocation Rule

The object is allocated to a player via the allocation rule in equation 1, there is only a single winner. Ties are broken randomly.

$$y_i(b_i) = \begin{cases} 1 \text{ if } b_i = \max_{\forall j \in [n]} b_j \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

Only the last bid is considered for determining the winner.

## 3.2 Payment Rule

The expected payment received by a player is given by Equation 2. This is an *all-pay* auction in which each player pays his/her last bid.

$$t_i(b_i) = -b_i \tag{2}$$

## 3.3 Termination

The auction lasts for $T$ seconds and selects a winner at the end of $T$ seconds.

| Time | Action |
|---|---|
| 0 sec | Auction starts |
| 3 sec | Bob's `start_auction()` is called |
| 3.1 sec | Bob submits a bid of 0.2 |
| 5 sec | Alice's `start_auction()` is called |
| 5.1 sec | Alice submits a bid of 0.3 |
| 6 sec | Alice's `receive_bid()` is called with `bid_value` = 0.2 |
| 8 sec | Bob's `receive_bid()` is called with `bid_value` = 0.3 |
| 8.1 sec | Bob submits a bid of 0.5 |
| 13 sec | Alice's `receive_bid()` is called with `bid_value` = 0.5 |
| 13.1 sec | Alice submits a bid of 0.7 |
| 15 sec | Auction terminated |
| 15 sec | Alice pays 0.7 |
| 15 sec | Bob pays 0.5 |
| 15 sec | Alice wins the auction and receives 1 |

Table 2: Event timeline for an example auction

# 4

# Environment

The environment provides the following functionality:

## 4.1 Information Feedback

You receive information of start of a new auction and other player's bids delayed by $\delta$, which is a random variable. It's value may vary from player to player and bid to bid.

## 4.2 Re-submit

At any time, you have the option to resubmit a new bid value provided that your new bid value exceeds the old one by $\eta$ times. That is, if $b_i' \geq \eta b_i$ then $b_i \leftarrow b_i'$.

# 5

# Bot

Your task is to provide a contingency strategy in for your bot in an asynchronous manner. You need to provide 4 functions:

## 5.1 Initialize

The `__init__` function for your python class to initialize your variables.

## 5.2 Start Auction

The `start` function is given an *Auction ID* as an argument. You need to update your internal data structures. You may also submit any number of bids in this function call.

## 5.3 Receive Bid

The `receive_bid` function is given an *Auction ID* and a bid value submitted by another agent.

## 5.4 End Auction

The `end_auction` function is given an *Auction ID* signalling the end of an auction.

## 5.5 Submit Bid

For submitting a bid, you need to use the following syntax:

```
1 await super().submit_bid(auction_id, bid_value)
```

This function call will submit your bid to the main server.

# 6

# Getting Started

## 6.1 Installing Pre-requisites

The simulator requires `aiohttp` package and `Jinja2` templating engine. In order to install the prerequisites, you can use the following command. The simulator has only been tested on Python 3.7.4

```
pip install -r requirements.txt
```

## 6.2 Running an Auction

The simulator consists of two parts, the central server and player clients. You must first start the central server by running `bolt.py`. You can view all output at http://localhost:8080

```
python bolt.py
```

Next, you must start the player clients that will initially register themselves with the server with a unique port number as the argument.

```
python player_client.py [Port Number]
```

Lastly, you need to run a script that orchestrates an auction.

```
python run_tournament.py [Number of Matches]
```

# 7

# Usage

## 7.1 Making a bot

A template for making your own bot is given in `my_bots/dummy.py`.

```
1  from .client import Client
2
3
4  class Team_0_Bot(Client):
5      def __init__(self):
6          super().__init__()
7          self.name = "Superman"  # Your Bot's Name
8          # Your Initialization Code Here
9
10     async def start(self, auction_id):
11         await super().start(auction_id)
12         # Your code for starting an auction
13
14     async def receive_bid(self, auction_id, bid_value):
15         await super().receive_bid(auction_id, bid_value)
16         # Your code for receiving bids
17
18     async def end_auction(self, auction_id):
19         await super().end_auction(auction_id)
20         # Your code for ending auction
```

Figure 1: Template Code

1. Rename the class name to have your team number.
2. Pick a nickname and rename your bot on line 7.
3. Remove the comment on the functions and replace them with your logic. *Do not remove the calls to* ***`super()`***
4. Rename the file to `team_<teamNo>.py`

### 7.2 Configuring the client

In any copy of `player_client_[n].py` change the 3rd line to import your bot.

```
from my_bots.team_<teamNo> import Team_<teamNo>_Bot
```

In the 7th line instantiate your bot.

```
bot = Team_<teamNo>_Bot() #Your bot here
```

### 7.3 Running custom auctions

In order to run your own custom auction, you need to use `run_auction.py`. First, you need to specify which bots to run on line 11.

```
players = ['Batman', 'Superman']  # Select the names of your bots here
```

Then, you can run the file as follows:

```
python run_auction.py <Duration> <Key>
```

Where `<Key>` is any random number.

### 7.4 Running multiple auctions

In order to run multiple auctions in the manner we would be running in the tournament, you need to use `run_tournament.py`. First, you need to specify which bots to run on line 10.

```
players = ['Batman', 'Superman']  # Select the names of your bots here
```

Then, you can run the file as follows:

```
python run_auction.py <No_of_matches>
```

# A

# Tournament Details

All bots will run in a league tournament in which every bot gets a chance to play 100 auctions against every other bot. At the end of the day, we $p_i$ will be the total profit of a bot ($p_i = \sum_{\forall j} t_i^{(j)}(b_i^{(j)}) + y_i^{(j)}(b_i^{(j)})$ where $(j)$ denotes the $j^{\text{th}}$ auction).

### A.1 Scoring

Let the profit made by your bot in the tournament be $p_i$, then your score $s_i$ will be given by Equation 3.

$$s_i = \frac{p_i}{|\max_{\forall j \in [n]} p_j|} \tag{3}$$

# B

# Grading

| Submission | Weightage | Grading |
|---|---|---|
| Warm-Up | 2.5% | 2 marks for working code + $0.5 \times s_i$ |
| Final Submission | 10% | 2 marks for neat code + $8 \times s_i$ |
| Strategy Document | 2.5% | 2.5 marks for write-up |

Table 3: Grading Details of Project Components

### B.1 Details

#### Warm-Up Submission

In this submission you are required to submit a code that works without errors. Entire 2 marks will be given if the code does not throw a single error.

Additionally, you will receive $0.5 \times s_i$ marks based on your performance. We will release the results soon so that you can judge your performance and improve your strategies.

#### Final Submission

In this submission you are not only required to submit error-free code but your code should be neat. We expect your code to be as python-ic as possible (check out https://docs.python-guide.org/writing/style/ for more details) and your code should be appropriately commented. The marks will be given based on a manual evaluation by the TA.

Your score would be majorly determined by your performance in the tournament ($8 \times s_i$ marks).

#### Strategy Document

You should make a document describing your strategy in detail. This would be manually evaluated by the TA. We would be looking for the following points:

1. An explanation of the strategy.
2. Why did you pick this strategy?
3. If you do well in the tournament explain why you believe you performed better than other strategies.
4. If you don't do well in the tournament, you still have an opportunity to learn from your mistakes. Analyze where you failed and how could you have done better.
5. We are expecting a qualitative analysis but quantitative analysis would also be appreciated.

# C

---

# FAQ

## C.1 Can my score be negative?

Yes, your score can be negative if your bot makes loss. You can lose marks from other components due to this. (Notice that $-\infty \leq s_i \leq 1$)

## C.2 I am facing an error.

Have you tried turning it on and off?

1. Kill all server and client processes

2. Delete the `db.sqlite3` file.

3. Follow the steps mentioned in Section 6.

### C.3 I tried C.2 but I am still facing an error.

Contact the TA early on. Last moment doubts may not be cleared and errors may not be fixed, the TA is a human.