U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

OLX

Second-hand Marketplace Simulation using MAS

Agents and Distributed Artificial Intelligence

2nd project - Final Delivery

David Silva - up201705373
Luís Cunha - up201706736
Manuel Coutinho - up201704211

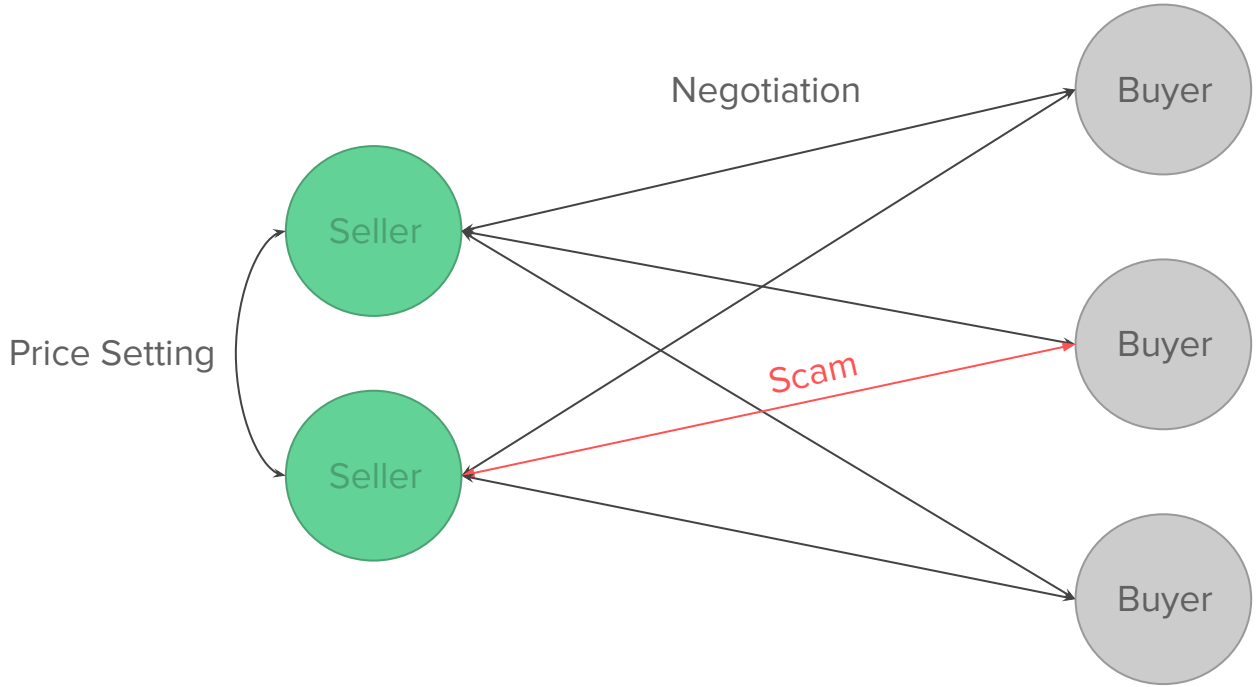
Part I - Presentation

Problem Description

- The modelled multi-agent system represents a **second-hand online marketplace** with two types of agents: **buyers** and **sellers;**
- **Sellers** seek to maximize the selling price of the products while trying to stay competitive with other sellers;
- **Buyers** wish to spend the least cash possible while competing with other buyers and avoiding getting scammed* by sellers;
- This time, Buyers and Sellers may buy/sell different quantities of the same item;

* When a buyer gets scammed, he loses the money without getting the product and the seller loses credibility (or reputation);

Global Schema



Agents Architecture and Strategies

Picking Price Strategies (Seller)

- **Naive:**
  - *No sellers:* % original cost
  - *With sellers:* % min. seller offer
- **Smart:**
  - *No sellers:* % original cost
  - *With sellers:* % weighted (cred.) avg

Counter Offer Strategies (Buyer)

- **Relative / Random Absolute TFT:**
  - *Counter-Price:* increase last buyer offer by relative / constant amount.
  - *Best Offer:* only product price
- **Smart:**
  - *Counter-Price:* Last buyer bid + fraction of difference buyer and seller bids.
  - *With sellers:* product price, seller credibility and negotiation time

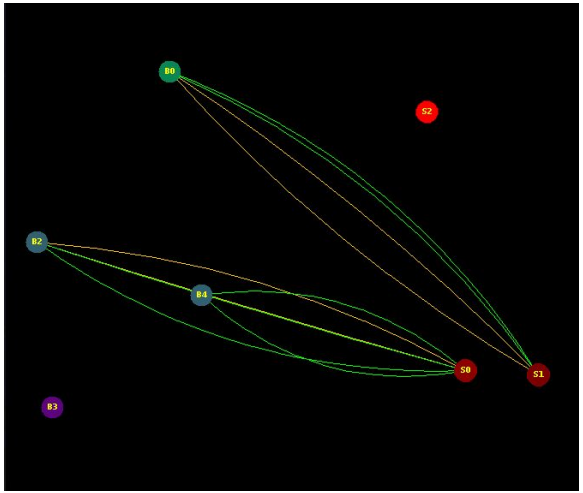
Offer Strategies (Seller)

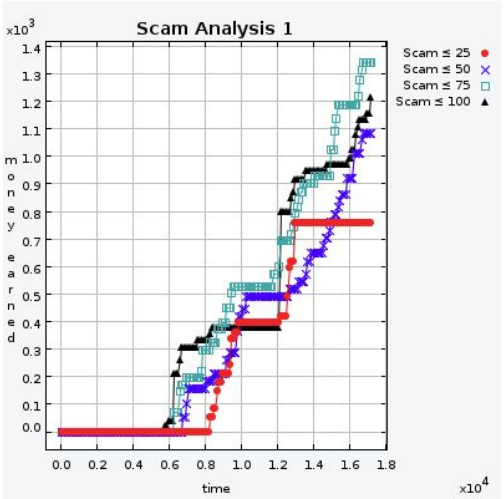
- **Relative / Random Absolute TFT:**
  - *Price:* decrease last seller offer by relative / constant amount.
- **Smart:**
  - *Price:* Last seller bid - fraction of difference buyer and seller bids.

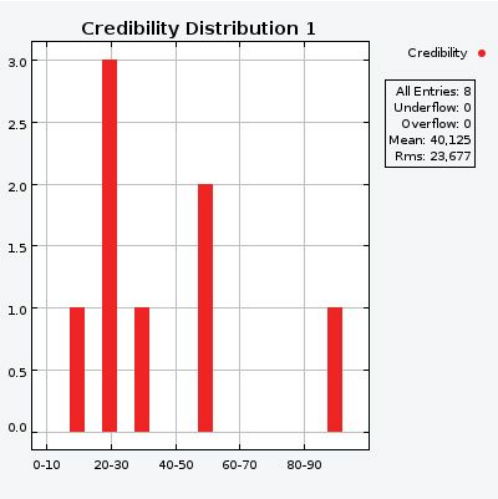
Controlled Variables

	System	*Seller*	*Buyer*
Independent	. Products . Seller count . Buyer count . Buyer waves . Wave period	. Product stock . Scam factor . Elasticity . Price picking strat. . Offer strat.	. To-Buy list . Patience . Picking strat. . Counter-offer strat.
Dependent	. Market prices	. Money earned . Credibility	. Money spent

Displays and Plots



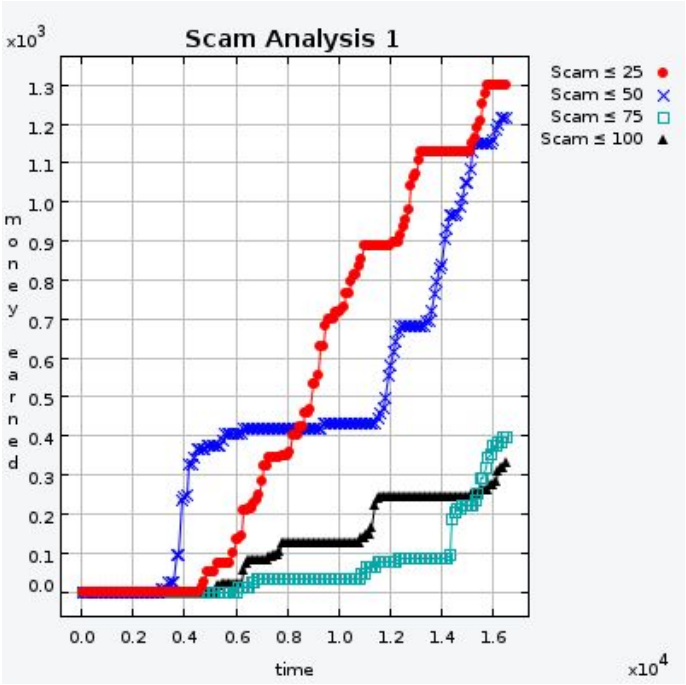




. **Sellers**: red nodes (credibility gradient)
. **Buyers**: green/blue/purple nodes (color based on strategy)
. **Trades**: green for success, orange for scam

. **Sequence plots (money/time)**: counter-offer strategy, offer strategy, elasticity and scam-factor;
. **Histogram**: credibility and market price;

Experiment #1 - “Does crime pay?”



- **Goal**: investigate if sellers with an high probability of scamming earn more money;
- **Setup**:
  - 20 sellers
    - 25 product-stock (original price 100$)
    - SMART picking/offer strategy
    - 20 elasticity
    - Scam factors: 25/50/75/100 (%)
  - 10 buyers (x 4 waves)
    - 5 product-stock
    - SMART counter-offer strategy
- **Results**: As we can see, seller return decreases with the scam factor. Due to buyers using SMART, they take seller credibility to account, and end up buying from no scammers (higher credibility)

Experiment #2 - “Is SMART actually smarter?” (Buyer)

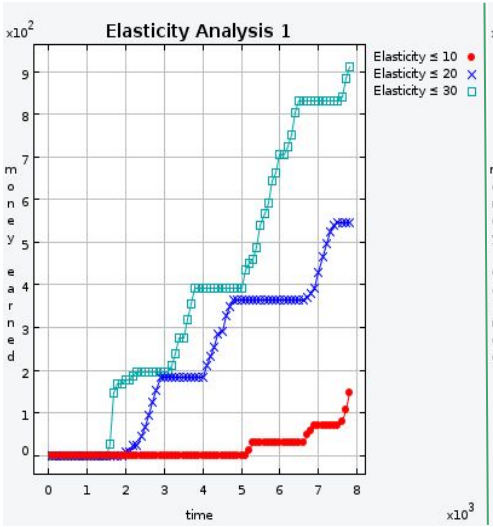
- **Goal**: investigate if buyers with a SMART strategy spend less money on average than the TFT strats;
- **Setup**:
  - 100 sellers:
    - 40 product-stock (original price 100$)
    - SMART picking/offer
    - 20 elasticity
    - Scam factors: 25/50/75/100 (%)
  - 99 buyers:
    - 20 product-stock
    - Counter-Offer: SMART/RELTFT/ABSTFT (33% each)

- **Results:**
  - SMARTs spend more money in the 1st phase
  - **SMARTs** buy all the products needed first (**better strategy if limited supply**)
  - Both **TFTs** strategies **overpay** for the product - not taking credibility into account leads to being scammed more often
  - **Absolute TFT** ends up being a **better negotiator** than **Relative TFT** (despite the slow start)

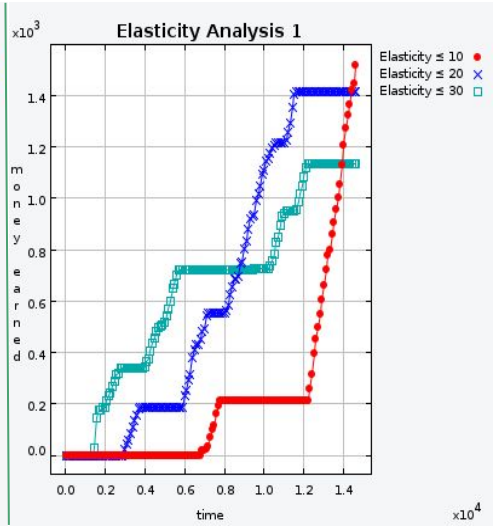
Experiment #3 - Seller price elasticity

- **Goal:** analyse the effect of the seller's elasticity in the amount of money they earn.
- **Setup:**
  - only one product (original price 100€)
  - everyone uses the **SMART** strategy
  - buyers looking for 5 items:
    - 40 in supply > demand
    - 90 in supply = demand
  - sellers:
    - 18 sellers each 25 stock each
    - 10/20/30(%) elasticities

supply > demand



supply = demand



- **Results:**
  - when **supply > demand**, sellers that **sell at a lower price earn more money** than their counterparts
  - in contrast, if there is **enough demand**, sellers with **lower elasticity tend to earn more** while selling less items, as we can see in the **supply = demand** plot
  - we conclude that **predicting demand** is essential to know how low you can go when negotiating a product

Experiment #4 - “Is SMART actually smarter?” (Seller)

- **Goals**: investigate if sellers with a SMART strategy earn more money on average than the TFT strats;
- **Setup**:
  - 18 sellers
    - 25 products (original price 100$)
    - SMART/RELTFT/ABSTFT picking/offer (split equally)
    - 20 elasticity
    - Scam factors: 0%
  - 40 buyers
    - 5 products
    - SMART counter-offer strategy
- **Results**: From the results we can’t conclude that the SMART strategy is better than the other ones, as the money earned follows a very similar evolution. In some experiments, SMART’s performance was surpassed by the Relative TFT.

Conclusions and Future work

- Using Repast allowed us to **streamline the development of test scenarios** for the Olx framework, which was a critical point of the first assignment;
- The experiment results were **coherent with real-world assumptions** which corroborates the applicability of the system;
- In the future, we see the system being extended with:
    - **Prediction methods** on the outcome of a negotiation to better estimate its utility;
    - **Machine Learning** based strategies;
    - Model second hand market for **rare items;**
    - Model **scalpers;**

Part II - Additional Information

Run Program

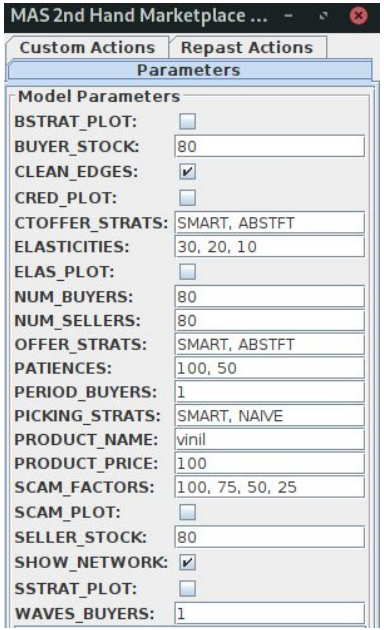
- Execute:
  - java -jar olx.jar <cli_args>
  - ./gradlew run --args="<cli_args>" in the root directory
- Command line arguments (<cli_args>):

-h, --help	shows help message and exit
-k, --kill	platform is shutdown after last buyer exits
-c, --config CONFIG*	file (YAML or JSON) with buyers and sellers configuration
-g, --generator CONFIG*	file (YAML or JSON) with buyers and sellers generation params
-b, --batch BATCH_NUM †	exec in batch mode with BATCH_NUM runs
-cl, --clean	keep only the last 4 buyer-trades in the negotiation network
-l, --logger	activate logging per agent
-s, --scam,	perform a scam analysis
-bs, --bstrat,	perform a buyer strategy analysis
-ss, --sstrat,	perform a seller strategy analysis
-e, --elasticity,	perform an elasticity analysis
-p, --price	perform a product price analysis
-cr, --credibility,	perform a credibility distribution analysis
-nn, --nonet	don't show the buyer/seller network

* Only in batch mode

† Should be used with --kill

Non-batch mode



- Set parameters using interface
- The following params receive a list of values:
  - Elasticities
  - Scam factors
  - Offer strats
  - Counter-offer strats
  - Picking strats
- List values create a partition in the sellers/buyers:
  - SCAM_FACTORS = [75, 75, 100, 25]:
    - First half of sellers will have scam factor of 75
    - Third quarter will have scam factor of 100
    - Last quarter will have scam factor of 25

Implemented Classes

Main Classes:

- **OLX** - main class responsible for initialization and agents creation
- **Buyer** - agent that represents a buyer
- **Seller** - agent that represents a seller
- **Buyer Launcher** - agent that periodically launches new buyers

Models:

- **OfferInfo** - contains the product and offered price
- **Product** - contains the name and its original price
- **Scam** - contains the OfferInfo scammed
- **SellerOfferInfo** - OfferInfo with seller credibility
- **Stock** - wrapper for representing quantities of the same product

Utils:

- **Config** - helper class that parses the config files
  - JsonConfig
  - GeneratorConfig
- **CoolFormatter** - custom log formatter
- **Stats** - responsible for calculating the statistics of the platform

Detailed Agents - Buyer Launcher

Attributes:

- **timeout -** milliseconds between buyer waves
- **nWaves -** number of buyer waves to create
- **FIRST_TIMEOUT -** time before the first wave is launched (not configurable)

Behaviors:

- **CreateBuyersBehaviour** - extend WakerBehaviour to periodically (see timeout) launch buyers in the Olx platform;

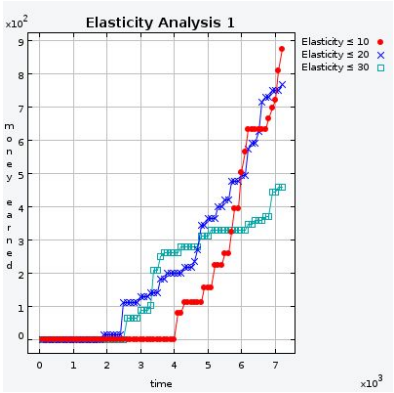
Buyer waves can be configured using the “**wavesBuyers**” and “**periodBuyers**” fields of the generator file (also using Repast interface).

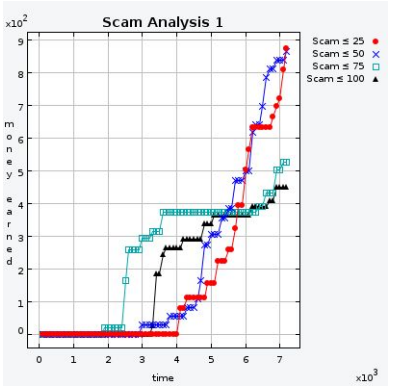
Implemented Classes - Repast Related

- **MyAverageSequence**
  - AverageSequence overload that returns 0 when the List is empty instead of throwing exception and breaking the program
- **MyDisplaySurface**
  - Extends DisplaySurface and overloads its removeDisplay method, avoiding NPE when the network is refreshed (due to null item in the “Options” default menu)
- **MyHistogram**
  - Extends Histogram and implements writeToFile and renameFile methods, allowing to write the histogram to a CSV file
- **Edge**
  - Extends DefaultEdge and implements DrawableEdge creating a convex undirected link between to nodes, avoiding edge overlap
- ***Plot / *Histogram**
  - Wrapper classes for the different plots and displays mentioned below

Plots and data

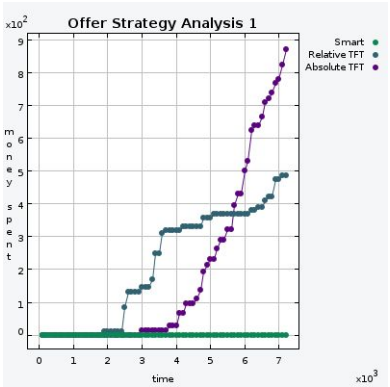
- Money earned grouped by scam factor (--scam)
  - Plot in analysis/snapshots/scam (line chart)
  - CSV data in analysis/csv/scam





- Money earned grouped by elasticity (--elasticity)
  - Plot in analysis/snapshots/elasticity (line chart)
  - CSV data in analysis/csv/elasticity

- Money earned grouped by seller offer strategy (--sstrat)
  - Plot in analysis/snapshots/seller_strat (line chart)
  - CSV data in analysis/csv/seller_strat

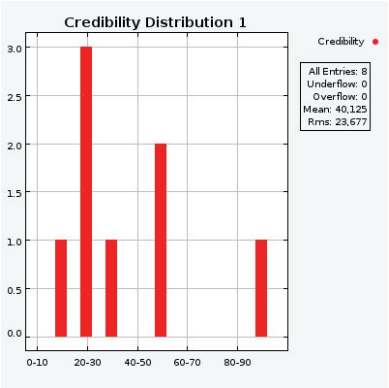


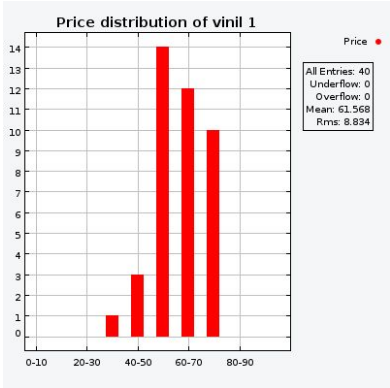
Plots and data (cont.)

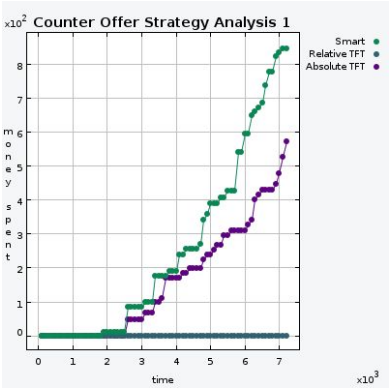
- Money spent grouped by buyer offer strategy (--bstrat)
  - Plot in analysis/snapshots/buyer_strat (line chart)
  - CSV data in analysis/csv/buyer_strat

- Absolute frequency distribution of the sellers' credibility (--credibility)
  - Plot in analysis/snapshots/credibility (histogram)
  - CSV data in analysis/csv/credibility

- Distribution of the products price in the simulated market (--price)
  - Plot in analysis/snapshots/product/{product_name} (histogram)
  - CSV data in analysis/csv/product/{product_name}







Initial configuration file - YAML*

Products:

```yaml
products:
 - name: pc
   price: 800
 - name: skate
   price: 150
 - …
```

General:

```yaml
wavesBuyers: 2
periodBuyers: 8000
```

Sellers:

```yaml
sellers:
 - scamFactor: 68
   elasticity: 14
   offerStrategy: SMART
   pickingStrategy: NAIVE
   products:
     - pc
     - skate
 - …
```

Buyers:

```yaml
buyers:
 - products:
     - pc
   counterOfferStrategy: ABSTFT
   patience: 80
 - products:
     - skate
   counterOfferStrategy: RELTFT
   patience: 63
```

- Defined offer/counter-offer strategies are ABSTFT (random absolute TFT), RELTFT (relative TFT) and SMART. Picking strategies are SMART and NAIVE.

* Initial file can also be in JSON format

Initial generation file - YAML

```yaml
product:
 name: vinil
 price: 100

numSellers: 8
numBuyers: 8
wavesBuyers: 1
periodBuyers: 8000

sellerStock: 10
buyerStock: 10
```

```yaml
scamFactors:
 - 90
 - 50
 - 50
 - 25
elasticities:
 - 30
 - 20
 - 10
patiences:
 - 100
 - 50
```

```yaml
pickingStrategies:
 - SMART
 - NAIVE

offerStrategies:
 - RELTFT
 - ABSTFT

counterOfferStrategies:
 - SMART
 - ABSTFT
```

- Defined offer/counter-offer strategies are ABSTFT (random absolute TFT), RELTFT (relative TFT) and SMART. Picking strategies are SMART and NAIVE.

Used libraries

- Repast v. 3.1 (and included auxiliary libraries)
- SAJas v. 0.92b
- Jade v. 4.5.0
- argparse4j v. 0.8.1
- jackson-dataformat-yaml v. 2.9.7
- jackson-databind v. 2.9.7