DAIIA Assignment Report

Course: [Distributed Artificial Intelligence and AI Agents]
Assignment: Homework 2

Lorenzo Deflorian, Riccardo Fragale, Juozas Skarbalius
KTH Royal Institute of Technology

November 20, 2025

Contents

1. Running Instructions

Import the ZIP file given on GAMA and go to the folder assignment2/models. You will find a file called FestivalAuction.gaml where our model for the homework is implemented. Then, from the interface of GAMA, click on the play button and you will see a simulation. Use the tools provided by the GAMA simulation interface to adjust the speed, read the outputs and verify on screen that everything is working correctly. Note that changing the parameters *numCenter,numShop and numGuests* will change the number of agents that will appear on the screen and be part of the simulation. These parameters are still related to the implementation of the first homework but since we are introducing auctions in the previous model, they might be still relevant for simulation purposes. We also have the new parameter *numAuctioneers* that identifies the number of Dutch, English and Vickrey Auctioneers that appear in the map.

2. General Overview

2.1 New Agents

In addition to the previous model of the Festival we needed to add new agents that are repsonsible for running auctions on several different products. For simulation purposes and to ease the debug procedure we have removed as much as possible all the logs and the screen appearing of things related to the first homework. As far as the basic implementation and the first challenge we have an agent called **Auctioneer** that is responsible for the so-called Dutch auctions. It appears on the map in a random location and it is a blue square of dimension 5. For the purpose of the second challenge we implemented two new agents, called **EnglishAuctioneer** and **VickreyAuctioneer** that are responsible for English and Vickrey auctions. All auctioneers sell the same products to guests; alcohol, sugar and magic crystals (they in a bit recall the environment of a techno music festival).

2.2 Assumption

In our festival we would have only three auctioneers and, at least for the case of the Dutch Auctioneer, it is also able to showcase multiple products at the same time. As a general rule, all interactions between auctioneers and bidders will follow FIPA protocol, instead of pure ask commands. The auctioneers will appear on display at the beginning of the simulation, but they will start the auction procedure after a certain randomic time defined as *rnd(15,150)*. Each product will be sold just once by each auctioneer.

2.3 Goals

The four main goals for this homework are the following:

- More experience with agents in GAMA

- Introduction to message passing and FIPA protocol
- Experience working with agent negotiation
- Simulating and practicing in an auction

3. Basic implementation

3.1 Explanation

As said before in the general overview we were aske to implement a new agent, called **Auctioneer** that is responsible for holding Dutch auctions of certain products inside the festival. Just as a brief remainder, a Dutch auction starts with an offer by the auctioneer (in our case a randomic value) with much higher price than the expected market value. If no one wants to buy for the set price, the auctioneer reduces the price at selected interval. He is completely free to decide how much the price is reduced in every round. If the price gets reduced below a minimum threshold, the auction is cancelled and the product is not sold.

3.2 Code

First of all, following the guidelines of a Dutch auction we set up the market prices of the items (they will be valid also for challenge 1 and 2).

```
float alcohol_price <- 1000.0;
float sugar_price <- 750.0;
float astonishings_price <- 10000.0;
```

Then, considering the premises of the general overview the auctioneer must start from a higher price and must not sell the price lower than a baseline. In our case the initial price is randomized and is set between 110% and 150% of the market value of the item. He is also responsible of deciding of how much to reduce the price of things between rounds.

```
float min_inc_coeff <- 1.1;
float max_inc_coeff <- 1.5;
float baseline_min_price_coeff <- 0.5;

// randomize the initial prices
float auctioned_alcohol_price <-
    rnd(alcohol_price * min_inc_coeff,
        alcohol_price * max_inc_coeff);
float auctioned_sugar_price <-
    rnd(sugar_price * min_inc_coeff,
        sugar_price * max_inc_coeff);
float auctioned_astonishings_price <-
    rnd(astonishings_price * min_inc_coeff,
```

```
            astonishings_price * max_inc_coeff);

    // this will be the price that the auctioneer
    // will not accept to sell below
    float baseline_alcohol_price <-
        alcohol_price * baseline_min_price_coeff;
    float baseline_sugar_price <-
        sugar_price * baseline_min_price_coeff;
    float baseline_astonishings_price <-
        astonishings_price * baseline_min_price_coeff;
    float price_decrease_factor <- 0.9;
```

Each auction will have an ID displayed on the screen and a state (either *init*, *running*, *idle*). After a product is sold, or the auction is aborted the state will be moved to idle and with a 10%probability the auctioneer will decide whether to restart the selling of that product in a certain moment. This logic will be valid also for the types of auctions of challenge 2.

Regarding the FIPA protocol, I'll show you below the interactions that are hsppening between an auctioneer and the guests that are bidding. Since the two categories of agents must exchange messages in order we setup a loop where auctioneers send CFP during even cycles and guests receive and processes proposals in odd cycle. To "measure" even and odd we used the time defined by GAMA simulation platform.

```
    reflex alcohol_auction_iteration when: (auction_state[0] = "running") {
    if (even(time)) {
        do auction_iteration_even(0, "alcohol", baseline_alcohol_price);
    } else {
        do auction_iteration_odd(0, "alcohol");
    }
}
```

This reflex is called alcohol-auction as it is separated from the other auctions due to needs of challenge 1. When the auctioneer starts an auction it sends a proposal to allGuests with the following auction:

```
    action sendProposalToGuests(int idx, string item, float price) {
    write '(Time ' + time + '):' + name + ' sent a CFP for ' + item + ' at price ' + p
    string id <- auction_id[idx];
    do start_conversation to: list(Guest) protocol: 'fipa-contract-net'
        performative: 'cfp'
        contents: [item, price, "dutch", id];
}
```

As it can be seen we are using FIPA protocol. The guest is receiving a CFP and processes it thanks to this method

```
    reflex receiveCFP when: !empty(cfps) and hunger < hungerThreshold and thirsty < th
                        and onTheWayToShop = false and targetShop = nil and beingAttacl
```

The logic inside is is that if the price_auction is below the maximum price it accepts immediately with "PROPOSE" as it can be seen in the following snippet.

```
    // In Dutch, accept if price is below valuation
        if (auction_type = "dutch") {
            current_auction_state <- "busy";
            current_auction_item <- auction_item;
            write '(Time ' + time + '):' + name
                  + ' accepts DUTCH price ' + auction_price
                  + ' for ' + auction_item + ' (max: ' + max_price + ')';
            do propose message: cfpMsg contents: [auction_item, auction_price];
        }
```

The auctioneer then processes the proposals and if there are one (or many) of them he selects the first one arriving. If there are no proposals it sends again a CFP with a reduced price (also check if the new price is below the acceptance baseline).

```
    action auction_iteration_odd(int idx, string item) {
    do receiveProposals(idx, item);

    // Lower price for next iteration
    current_auction_price[idx] <- current_auction_price[idx] * price_decrease_factor;
}
[....]
if (current_auction_price[idx] < baseline_price) {
        auction_state[idx] <- "abort";
        return;
    }

    do sendProposalToGuests(idx, item, current_auction_price[idx]);
```

Just as a final remark, when an auction is completed or aborted its state is moved to "idle" and the auction will restart with 10% probability as said before.

```
    // Restart idle auctions with probability
    reflex restart_idle_auctions {
        loop i from: 0 to: 2 {
            if (auction_state[i] = "idle") {
                if (flip(0.1)) { // 10% probability to restart per cycle
                    auction_state[i] <- "init";
                    write "Restarting auction " + i + "
                    (transition from idle to init)";
                }
            }
        }
    }
```

3.3 Demonstration

I will provide two use cases for this basic implementation:

- A user wins an auction for alcohol
- An auctioneer starts the auction for sugar

User wins alcohol auction.

Auctioneer starts auction for sugar.

4. Challenge 1: Multiple Auctions in the Festival

4.1 Explanation

In this first challenge we were asked to allow multiple auctions at the same time. In our case we decided to allow a multiple auctioneer to setup multiple auctions at the same time of different products. We would also show that guests are interested into different products and so they would bid offers only for what they are really interested.

4.2 Code

The Dutch auctioneer can three independent auctions simultaneously (alcohol, sugar, astonishings) using vectorized state management and separate reflexes per item. All auction state is stored in lists indexed by item (0=alcohol, 1=sugar, 2=astonishings). This is done because each item can be in a different state at the same time. Instead of a generic loop, each auction has its own reflex.

```
    reflex start_alcohol_auction when: (auction_state[0] = "init" and time >= start_ti
    reflex start_sugar_auction when:
        (auction_state[1] = "init" and time >= start_time)
```

Since GAML's reflex scheduler fires all matching reflexes in the same cycle, only three separate reflexes allow all auctions to start independently whenever conditions match. We also opted for a *Centralised proposal collection* so that proposals are collected once per cycle into a persistent map.

```
    map<string, list<message>> pending_proposals <- map(
    ["alcohol"::[], "sugar"::[], "astonishings"::[]]);

reflex collect_proposals when: !empty(proposes) {
    loop proposeMsg over: proposes {
        list contents_list <- list(proposeMsg.contents);
```

```
        string item <- string(contents_list[0]);

        if (item = "alcohol") {
            add proposeMsg to: pending_proposals["alcohol"];
        } else if (item = "sugar") {
            add proposeMsg to: pending_proposals["sugar"];
        } else if (item = "astonishings") {
            add proposeMsg to: pending_proposals["astonishings"];
        }
    }
}
```

This is done because FIPA mailbox is consumed immediately; storing proposals by item prevents errors when processing multiple auctions. On elast thing is that actions taken an *idx* parameter to work on the correct item.

Now we need to explain how guests can decide whether to take part in an auction. They participate in auctions only for items needed. Interest is determined by item preferences and verified before sending proposals. The auctioneer never receives proposals for unwanted items. Each guest has a list of items they care about:

```
    list<string> interest_items <- ["alcohol", "sugar", "astonishings"];
```

This is initialized randomly during guest creation. When a guest receives a CFP, it first checks interest (inside the reflex called *receiveCFP*) before deciding to propose:

```
    bool is_interested <- interest_items contains auction_item;
        if (!is_interested) {
            write '(Time ' + time + '):' + name +
                ' is not interested in ' + auction_item;
            do refuse message: cfpMsg contents:
                ['Not interested in this item'];
            continue;  // Skip to next CFP
        }
```

Only after confirming interest, the check whether the auction price is acceptable. The auctioneer only receives proposals for items guests are interested as shown below.

Figure 1: Collect proposals reflex

4.3 Demonstration

I will provide the following four use cases

- Two auctions starting at the same time
- Two auctions running at the same time
- A user decides not to take place in an auction for a certain item since it is not interested
- An auction is completed while another one is running

Two auctions starting at the same time.

Two auctions running at the same time.

User doesn't take place in an auction.

Auction completed while another one is running.

5. Challenge 2: Different auction settings

4.1 Explanation

For the purpose of this second challenge we were aske to implement at least two more type of auctions. We decided to implement English auctions and Vickrey auctions.

An English auction is an open ascending-bid auction where bidders continuosly raise prices; the auction continues until no higher bids are made and the winner is thus identified. It encourages competitive bidding and transparent price discovery.

A Vickrey auction is a sealed-bid auction where bidders submit private bids and the higher bidder wins the auction. However, they pay the second-highest bid, not their own. This format incentivizes truthful bidding and reduces strategic manipulation.

4.2 Code

To implement the two new auction types we created two new agents, called **EnglishAuctioneer** and **VickreyAuctioneer**. Both agents share similar structure and logic to the Dutch auctioneer but with differences in auction mechanics. In code snippet provided in 2, we can see how the Vickrey auction is concluded. Since Vickrey auctions require sealed bids, all proposals are collected without revealing bid amounts to other bidders. Once the bidding phase ends, the auctioneer identifies the highest and second-highest

```
// Find highest and second-highest bids
float highest <- -1.0;
float second  <- -1.0;
message winnerMsg <- nil;

loop proposeMsg over: item_proposes {
    list cont <- list(proposeMsg.contents); // [item, bid]
    float bid <- float(cont[1]);

    if (bid > highest) {
        second  <- highest;
        highest <- bid;
        winnerMsg <- proposeMsg;
    } else if (bid > second) {
        second <- bid;
    }
}

// Check reserve
if (highest < reserve_price) {
    write '(Time ' + time + '):' + name
          + ' VICKREY highest bid ' + highest
          + ' below reserve ' + reserve_price
          + ', aborting.';
    pending_proposals[item] <- [];
    auction_state[idx] <- "aborted";
    return;
}

// Second-price payment
float final_price <- reserve_price;
if (second >= 0.0) {
    final_price <- max(second, reserve_price);
}

winner of type Guest is assigned a value of type agent, which will be casted to Guest
Guest winner <- agent(winnerMsg.sender);
```

Figure 2: Code snippet: Vickrey auction price determination process

```
action auction_iteration_odd(int idx, string item) {
    // Only resolve if we actually sent a CFP
    if (auction_round[idx] = 0) {
        return;
    }
    do resolveAuction(idx, item);
}
```

Figure 3: Code snippet: Vickrey auction conclusion process

bids to determine the winner and the price they will pay. As seen it the code snippet, the auctioneer enforces that the second-highest bid price is paid by the winner. Unlike Dutch and English auction, Vickrey auctions are not iterative (i.e., single shot). This means that our logic for Vickrey auction is changed in a way that odd cycles determine the start of the auction and even cycles determine the end of it (as implemented by 3). With English auctions, the auctioneer starts with a low initial price and bidders openly bid higher amounts. The code snippet in 4 depicts how the highest local bid is updated.

4.3 Demonstration and comparison of prices

While it is hard to demonstrate the full auction process in a static report, we can provide some screenshots of the auctions in action.

English Auction in action. Figure 5 demonstrates an English auction in progress, where bidders are actively raising their bids for the item on sale.

```
float local_highest <- highest_bid[idx];
message winnerMsg <- nil;

loop proposeMsg over: item_proposes {
    list cont <- list(proposeMsg.contents);
    // contents: [item, bid]
    float bid <- float(cont[1]);
    if (bid > local_highest) {
        local_highest <- bid;
        winnerMsg <- proposeMsg;
    }
}
```

Figure 4: Code snippet: English auction highest bid update process

Figure 5: English auction in action

Figure 6: Vickrey auction in action

Vickrey Auction in action. Figure 6 demonstrates a Vickrey auction in progress, where conclusions of auctions are visible.

Price Comparison. For the comparison of all three auction types, we can achieve that by comparing the social welfare generated by each auction type. To keep things simple, we express social welfare as the sum of two components: the utility gained by the winning bidder and the revenue earned by the auctioneer. We hold that the utility gained by the bidder is the difference between their valuation of the item and the price they paid. We don't calculate the sum of utilitiies (i.e., total social welfare), however we implemented two dynamic real-time graphs in GAMA that depict surplus of all bidders and auctioneer's revenues of all auction types. Figure 7 shows the comparison the surpluses and revenues of all three auction types. From the graphs, we can observe that the Vickrey auction tends to generate higher bidder surpluses (72%) compared to the Dutch and English auctions. This is because bidders in Vickrey auctions are incentivized to bid their true valuations, leading to more efficient outcomes. The English auction shows moderate bidder surpluses, as the open bidding process allows bidders to adjust their bids based on competition. The Dutch auction, on the other hand, often results in lower bidder surpluses due to its descending price mechanism, which may lead to quicker sales but at potentially lower prices. This might also be due to the implementation fact that single-shot Vickrey auctions happen significantly more frequently than any other auction types combined. Due to the same reasons, the auctioneer's revenue is highest in the Vickrey auction (41%) and in the Dutch auction (41%), and lowest in the English auction (19%).

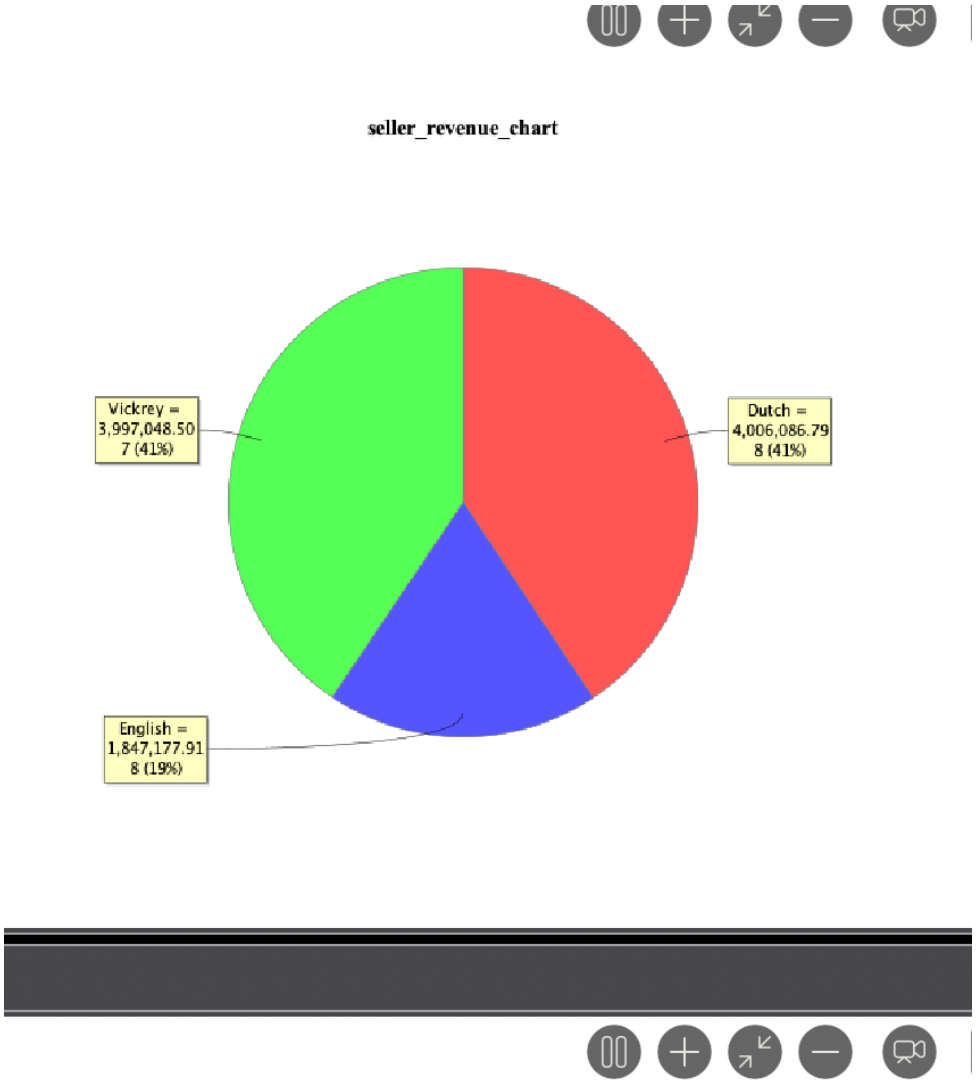

Figure 7: Social welfare comparison of auction types

6. Final Remarks

In conclusion, this assignment provided us a second oppurtunity to learn the basics of GAMA simulation modelling. Moreover, we were also to "dirty" our hands or harder tasks such as communicsating using FIPA protocol and establishing auctions. This homework was definitely more interesting and challenging and thus we are very happy of having been able to correctly deliver it.