Project Report

Distributed Artificial Intelligence and Intelligent Agents

(ID2209)

KTH ROYAL INSTITUTE OF TECHNOLOGY

Group : 11

Name : Md Sakibul Islam
Name : Md Ahsanul Karim
Name : Singvalliyappa Velayutham

Date : 21 November, 2025

1. General Overview:

The simulation models a festival environment where guests and newly added merch auctions interact. Guests have budgets, and they move around the festival. Auctioneers appear during the simulation to sell merch items using a Dutch auction mechanism. Each auction starts at a high price and decreases until a buyer accepts or the reserve price is reached. All communication follows FIPA message standards, ensuring structured agent-to-agent interaction. The system demonstrates autonomous decision-making by both auctioneers and buyers, integrating preference matching, budget constraints, and coordinated message-based negotiation.

2. Running Instructions

To run the simulation model developed in this assignment, we need to follow these steps:

- Open GAMA Platform.
- Import the provided project files into GAMA by selecting File > Import, navigating to the 'General' folder, and choosing 'Existing Projects into Workspace'. Select the archive file and complete the import process.
- Locate the Main.gaml model within the imported project.
- Locate the Challenge1.gaml model within the imported project.
- Locate the Challenge2.gaml model within the imported project.
- Select the main experiment, which serves as the entry point for running the simulation.
- Press the 'Play' button seems like “Festival_Simulation” to initiate the simulation run.
- We coded in 3 gaml file where we first develop the basic part and then we added challenge-1 & challenge-2 later on.

3. Species

Agent Auctioneer

The auctioneer is the agent responsible for managing Dutch auctions inside the festival. It creates and prepares merchandise items, assigns each item a starting price, minimum threshold, and price-reduction step for each auction round. The auctioneer communicates exclusively using FIPA performatives, sending **CFP** messages to announce current prices and receiving **PROPOSE** messages from interested guests. Once a valid proposal is received, it confirms the winner with an **INFORM** message and

finalizes the sale. The auctioneer monitors each auction it conducts, recording whether it succeeded, how much revenue it generated, and how many rounds it required.

4. Section : Add Merch to Festival

4.1 : Explanation:

This part of the assignment required adding merchandise to the festival and creating a system where that merch could be sold through an auction. Instead of simple interactions, we had to use **FIPA message passing**, meaning agents communicate only through formal messages like CFP, PROPOSE, and INFORM. The goal was to learn how agents negotiate, respond to offers, and complete an auction inside the simulation.

We introduced a new agent, the **Auctioneer**, whose job is to handle merch sales. Each auctioneer creates an item and sets up everything needed for a Dutch auction: the starting price, the minimum threshold, and how much the price drops every round. During the auction, the auctioneer sends out **CFP messages** to all guests, each time announcing the new, lower price.

Guests check these offers based on their budget and interest. When the price becomes reasonable for one of them, they send a **PROPOSE message** back to the auctioneer. The auctioneer accepts the first valid proposal and completes the sale by sending an **INFORM message** to confirm the winner.

In this way, merchandise becomes part of the festival, and the simulation now includes realistic agent communication, decision-making, and negotiation through the Dutch auction process.

4.2 : Code:

- **Relevant Code Snippet(s) :**

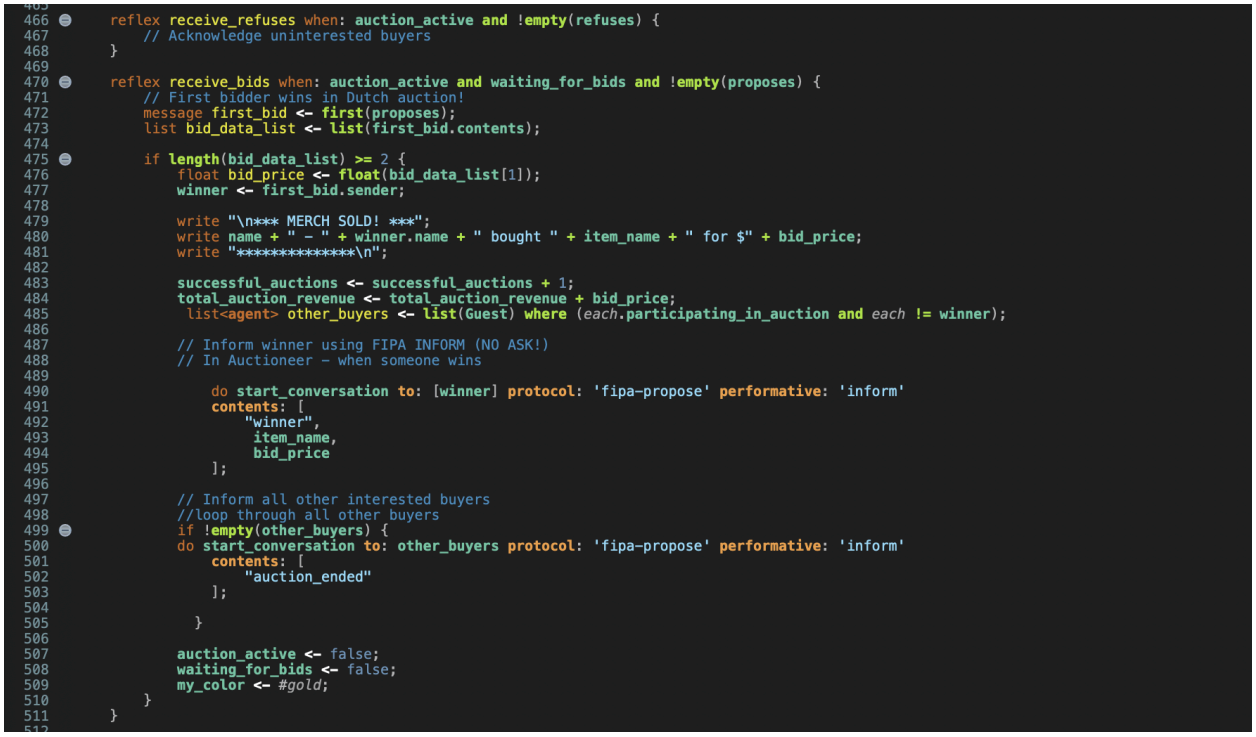
```
// --- Auctioneer starts a Dutch auction ---
action start_auction {
    auction_active <- true;
    item_name      <- one_of(merch_items);
    starting_price <- rnd(40.0, 100.0);
    current_price  <- starting_price;
    minimum_price  <- starting_price * 0.25;
    price_reduction <- starting_price * rnd(0.06, 0.12);
```

```
    // Broadcast CFP to all guests
    do start_conversation
        to: list(Guest)
        protocol: 'fipa-propose'
        performative: 'cfp'
        contents: [
            name, item_name, current_price, "auction_start"
        ];
}

// --- Guest receives CFP and sends PROPOSE when price is acceptable ---
reflex receive_cfp when: !empty(cfps) {
    loop msg over: cfps {
        list c <- list(msg.contents);
        float current_price <- float(c[2]);

        if current_price <= auction_budget {
            do propose message: msg contents: [name, current_price];
        }
    }
}

// --- Auctioneer accepts first bid and informs winner ---
reflex receive_bids when: auction_active and !empty(proposes) {
    message bid <- first(proposes);
    list info <- list(bid.contents);
    float bid_price <- float(info[1]);
    winner <- bid.sender;

    do inform message: winner contents: [
        "winner", item_name, bid_price
    ];

    auction_active <- false;
}
```

- **Code Screenshots:**

```
281
282       // AUCTION REFLEXES - FIPA Protocol Communication
283
284
285     reflex receive_inform when: !empty(informs) {
286         loop inform_msg over: informs {
287             write"yessss";
288             list data <- inform_msg.contents;
289
290             string msg_type <- string(data[0]);
291             write data;
292             write msg_type;
293
294             if msg_type = "winner" {
295                 string item_name <- string(data[1]);
296                 float final_price <- float(data[2]);
297
298
299                 add item_name to: owned_merch;
300                 auction_budget <- auction_budget - final_price;
301                 participating_in_auction <- false;
302                 max_willing_to_pay <- 0.0; // Reset for next auction
303
304                 write name + " WON AUCTION! Bought " + item_name + " for $" + final_price + ". Remaining budget: $" + auction_budget;
305
306
307             } else if msg_type = "auction_ended" {
308                 participating_in_auction <- false;
309                 max_willing_to_pay <- 0.0; // Reset for next auction
310             }
311         }
312     }
313
```

```
313
314     reflex receive_cfp when: !empty(cfps) {
315         loop cfp_message over: cfps {
316             // FIPA contents come as a simple list [value1, value2, value3...]
317             list contents_list <- list(cfp_message.contents);
318
319             // Parse based on expected order: [auction_id, item_name, item_genre, starting_price, current_price, message_type]
320             if length(contents_list) >= 6 {
321                 string auction_id <- string(contents_list[0]);
322                 string item_name <- string(contents_list[1]);
323                 string item_genre <- string(contents_list[2]);
324                 float starting_price <- float(contents_list[3]);
325                 float current_price <- float(contents_list[4]);
326                 string msg_type <- string(contents_list[5]);
327
328                 if msg_type = "auction_start" {
329                     // Decide if interested based on genre and budget
330                     if item_genre in preferred_genres and auction_budget > starting_price * 0.2 and !participating_in_auction {
331                         participating_in_auction <- true;
332                         current_auction_id <- auction_id;
333                         max_willing_to_pay <- min(auction_budget * rnd(0.7, 0.95), starting_price * rnd(0.8, 1.1));
334
335                         write ">>> " + name + " INTERESTED in " + item_name + " (" + item_genre + "). Max willing: $" + max_willing_to_pay;
336                     } else {
337                         // Send REFUSE - not interested
338                         do refuse message: cfp_message contents: [name, false];
339                     }
340                 } else if msg_type = "price_update" and participating_in_auction {
341                     if auction_id = current_auction_id {
342                         // Dutch auction: Bid if price is acceptable
343                         if current_price <= max_willing_to_pay and current_price <= auction_budget {
344                             do propose message: cfp_message contents: [name, current_price];
345                             write name + " BIDS at price: $" + current_price;
346                         }
347                     }
348                 }
349             }
350         }
351     }
```



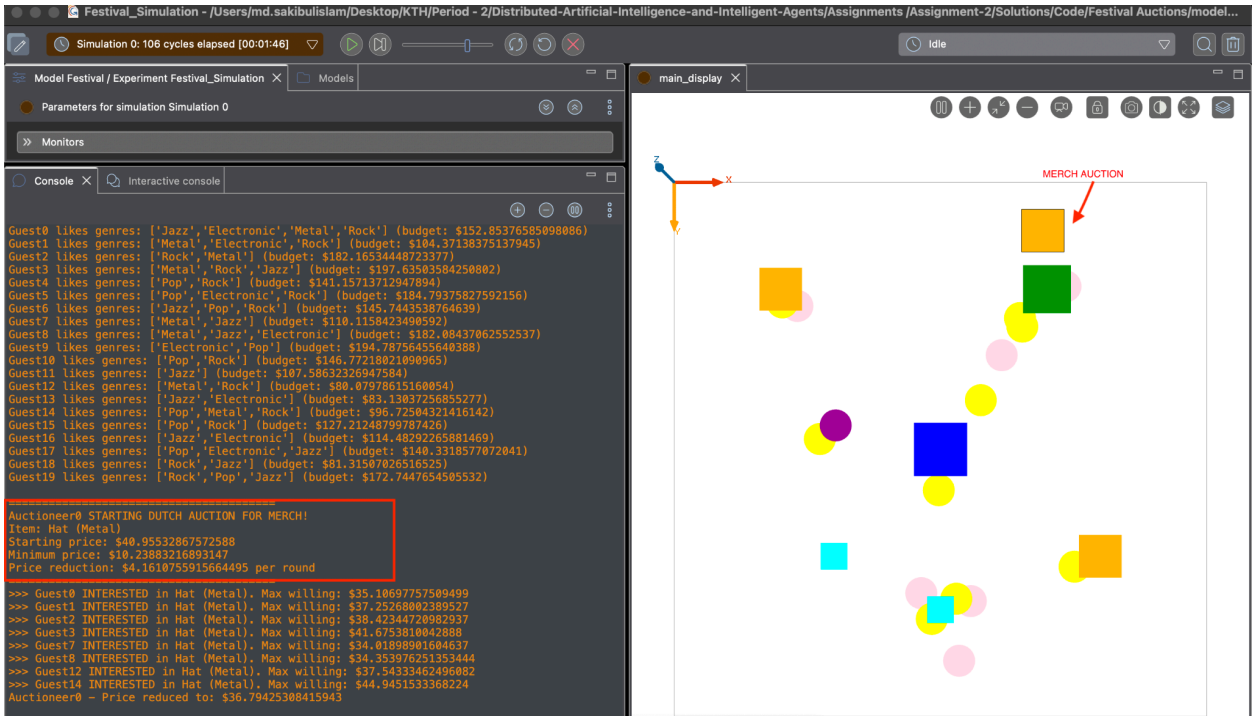
```
    reflex receive_refuses when: auction_active and !empty(refuses) {
        // Acknowledge uninterested buyers
    }

    reflex receive_bids when: auction_active and waiting_for_bids and !empty(proposes) {
        // First bidder wins in Dutch auction!
        message first_bid <- first(proposes);
        list bid_data_list <- list(first_bid.contents);

        if length(bid_data_list) >= 2 {
            float bid_price <- float(bid_data_list[1]);
            winner <- first_bid.sender;

            write "\n*** MERCH SOLD! ***";
            write name + " - " + winner.name + " bought " + item_name + " for $" + bid_price;
            write "**************\n";

            successful_auctions <- successful_auctions + 1;
            total_auction_revenue <- total_auction_revenue + bid_price;
             list<agent> other_buyers <- list(Guest) where (each.participating_in_auction and each != winner);

            // Inform winner using FIPA INFORM (NO ASK!)
            // In Auctioneer - when someone wins

                do start_conversation to: [winner] protocol: 'fipa-propose' performative: 'inform'
                contents: [
                    "winner",
                    item_name,
                    bid_price
                ];

            // Inform all other interested buyers
            //loop through all other buyers
            if !empty(other_buyers) {
            do start_conversation to: other_buyers protocol: 'fipa-propose' performative: 'inform'
                contents: [
                    "auction_ended"
                ];

              }

            auction_active <- false;
            waiting_for_bids <- false;
            my_color <- #gold;
        }
    }
```

- **Explanation of the code :** This code implements the complete merch auction system using FIPA communication. The **Auctioneer** creates a merch item, sets prices, and begins a Dutch auction by sending **CFP** messages to guests. It continuously lowers the price until a buyer responds. Guests who are interested and within budget wait until the price reaches an acceptable level and then send a **PROPOSE** message. The auctioneer accepts the **first** bid, sends an **INFORM** message declaring the winner, and ends the auction. Guests update their owned merch and budget based on the result.

4.3 : Demonstration

- **Use Case - 1 : Auctioneer Starts the Dutch Auction**
    - **Description :** The auctioneer initializes a Dutch auction for a merch item. They begin by sending a **CFP** message to all buyers with a starting price that is intentionally much higher than market value. Buyers receive the offer but take no action because the price is too high. The auctioneer then prepares to reduce the price in the next round.

- **Screenshot of program execution/output :**

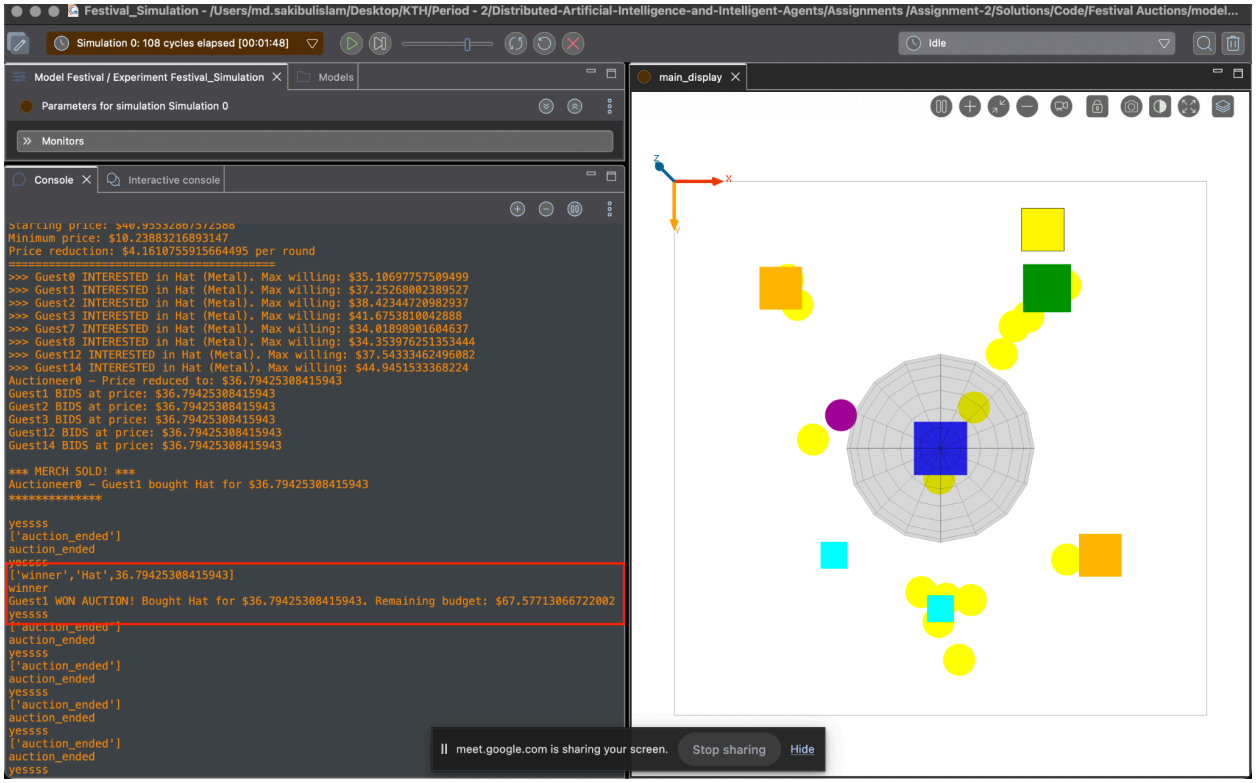


  - **Short interpretation of the result :** The auctioneer correctly starts a Dutch auction, announces the item, and lowers the price each round. Multiple guests show interest with their maximum willing price. Price reduction works, and buyers are evaluating the offers as intended.

- **Use Case - 2 : Buyer Wins the Dutch Auction**

  - **Description :** The auctioneer lowers the price each round and sends **CFP** messages. When the price becomes acceptable, a buyer sends a **PROPOSE** message. The auctioneer accepts the first valid proposal, sends an **INFORM** message confirming the win, and transfers the item. The auction ends immediately.

- **Screenshot of program execution/output :**



- **Short interpretation of the result : Guest1** places the first valid bid when the price drops.The auctioneer accepts it, marks the auction as ended, and logs that Guest1 bought the hat for **$36.79**, with the remaining budget updated. All follow-up messages confirm proper FIPA communication and auction termination.

5. Challenge-1: Multiple Auctions in the Festival

5.1 : Explanation:

Challenge 1 required enabling **multiple auctioneers** to run Dutch auctions at the same time. Guests needed to handle CFP messages from different auctioneers, recognize which auction each message belonged to, and respond correctly through the FIPA protocol. Each auction had to function independently without message conflicts.

We solved this by creating several auctioneers, each running its own Dutch auction. Every auctioneer generates a **unique auction ID**, which is included in all CFP, PROPOSE, and INFORM messages. Guests use this ID to decide which auction they join and ignore others. Each auctioneer processes only proposals matching its own ID,

allowing multiple auctions to run in parallel while maintaining clean communication, proper routing, and correct winner selection.

5.2 : Code:

- **Relevant Code Snippet(s) :**

```
reflex trigger_auction when: time >= next_auction_time {
        // Trigger auction on a random available auctioneer
        if (!auction_active) {
                do start_auction;
                auction_interval <- rnd(50.0,60.0);
                next_auction_time <- time + auction_interval;

        }
    }

    action start_auction {
        if !auction_active {
            auction_active <- true;
            current_auction_id <- name + "_" + string(time);

            // Generate random merchandise item with GENRE
            item_name <- one_of(merch_items);
            item_genre <- one_of(merch_genres);
            starting_price <- rnd(40.0, 100.0);
            current_price <- starting_price;
            minimum_price <- starting_price * 0.25;
            price_reduction <- starting_price * rnd(0.06, 0.12);
            last_reduction_time <- time;

            participants <- [];
            winner <- nil;
            waiting_for_bids <- true;
            my_color <- #orange;

            total_auctions <- total_auctions + 1;

            write "\n=======================================";
            write name + " AUCTION #" + total_auctions;
            write "Item: " + item_name + " | Genre: " + item_genre;
```

```
            write "Starting: $" + starting_price + " | Min: $" + minimum_price;
            write "========================================";

            // Send CFP to all guests using FIPA protocol
            do start_conversation to: list(Guest) protocol: 'fipa-propose' performative:
'cfp' contents: [
                current_auction_id,
                item_name,
                item_genre,
                starting_price,
                current_price,
                "auction_start"
            ];
        }
    }
```

- **Code Screenshots:**

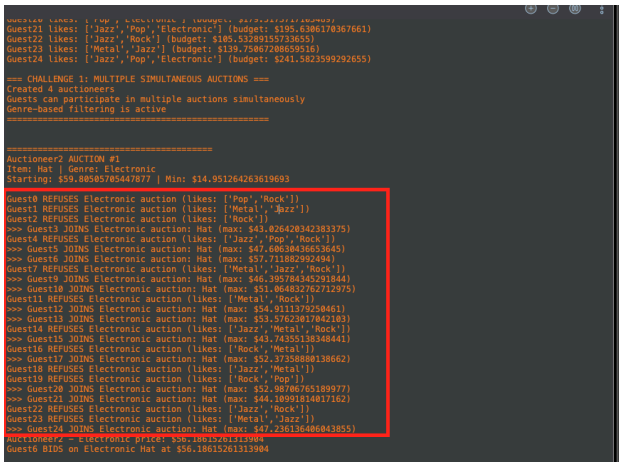
```
274
275     // CHALLENGE 1: AUCTION REFLEXES - Support MULTIPLE SIMULTANEOUS AUCTIONS
276
277     reflex receive_cfp when: !empty(cfps) {
278         loop cfp_message over: cfps {
279             list contents_list <- list(cfp_message.contents);
280
281             // Parse: [auction_id, item_name, item_genre, starting_price, current_price, message_type]
282             if length(contents_list) >= 6 {
283                 string auction_id <- string(contents_list[0]);
284                 string item_name <- string(contents_list[1]);
285                 string item_genre <- string(contents_list[2]);
286                 float starting_price <- float(contents_list[3]);
287                 float current_price <- float(contents_list[4]);
288                 string msg_type <- string(contents_list[5]);
289
290                 if msg_type = "auction_start" {
291                     // CHALLENGE 1: Check if interested in THIS GENRE
292                     if item_genre in preferred_genres and auction_budget > starting_price * 0.2 and !(auction_id in active_auction_ids) {
293                         // Join this auction!
294                         if(!(auction_id in ended_auction_ids)){
295
296
297                         add auction_id to: active_auction_ids;
298                         float max_willing <- min(auction_budget * rnd(0.6, 0.9), starting_price * rnd(0.7, 1.0));
299                         auction_max_prices[auction_id] <- max_willing;
300                         write ">>> " + name + " JOINS " + item_genre + " auction: " + item_name + " (max: $" + max_willing + ")";
301
302                         //NOT BIDDING HERE DIRECTLY - ONLY THE FIRST GUEST MIGHT GET A CHANCE
303                         //WAITING FOR THE PRICE UPDATE SO POTENTIAL GUESTS JOIN THE AUCTION FIRST
304
305                         }
306
307
308                     } else {
309                         // Not interested in this genre or already in this auction
310                         do refuse message: cfp_message contents: [name, false];
311                         if !(item_genre in preferred_genres) {
312                             write name + " REFUSES " + item_genre + " auction (likes: " + preferred_genres + ")";
313                         }
314                     }
315                 } else if msg_type = "price_update" and (auction_id in active_auction_ids and !(auction_id in ended_auction_ids) ) {
316                     // Get max willing to pay for THIS specific auction
317                     float max_willing <- float(auction_max_prices[auction_id]);
318
319                     // Dutch auction: Bid if price is acceptable
320                     if current_price <= max_willing and current_price <= auction_budget {
321                         do propose message: cfp_message contents: [name, current_price];
322                         write name + " BIDS on " + item_genre + " " + item_name + " at $" + current_price;
323                     }
324                 }
325             }
326         }
327     }
328
```

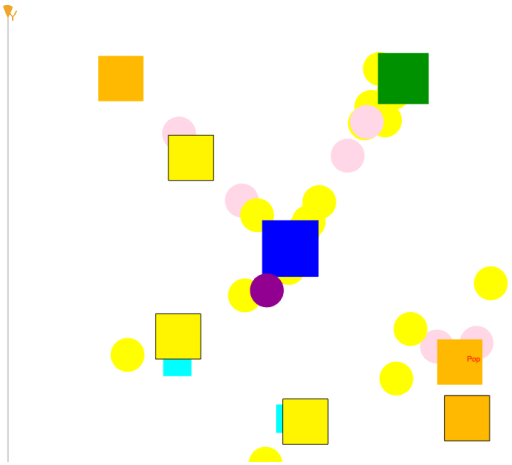
- **Explanation of the code :** This reflex allows each guest to process CFP messages from several auctioneers at once by reading the auction details, checking if the item's genre matches their preferences, and joining only auctions they are interested in and can afford. When an auction starts, the guest stores its own maximum price for that specific auction using the auction ID, and when a price update arrives, it bids only if the reduced price is below that stored limit. If uninterested, the guest sends a REFUSE message. Overall, this code enables clean and independent participation in multiple simultaneous Dutch auctions using FIPA communication.

5.3 : Demonstration:

- **Use Case - 1 : Guest Joins a Genre-Matching Auction**
    - **Description :** A guest receives CFP messages from multiple auctioneers and joins only the auctions whose merch genre matches its "preferred_genres" and whose starting price fits its budget. When this happens, the guest adds the "auction_id" to "active_auction_ids" and stores a "max_willing_to_pay" value for that specific auction. If the genre is not preferred, the guest immediately sends a REFUSE message and ignores that auction.

    - **Screenshot of program execution/output :**

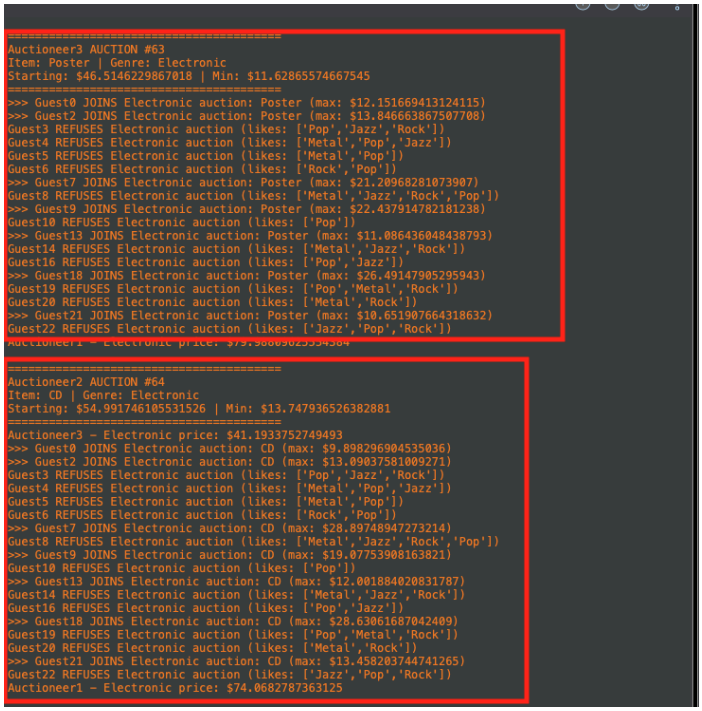




    - **Short explanation of the results:**
      The output shows the guests' decisions during a simultaneous auction. Each guest receives the Electronic-genre CFP and either REFUSES if they don't like Electronic or JOINS if the genre matches their preferences and their budget allows it. Their printed "max" value shows how much they

are willing to pay in this specific auction. As the auctioneer lowers the price, only guests who joined the auction begin placing bids. This confirms that genre-based filtering, multi-auction participation, and Dutch auction bidding behavior are working correctly.

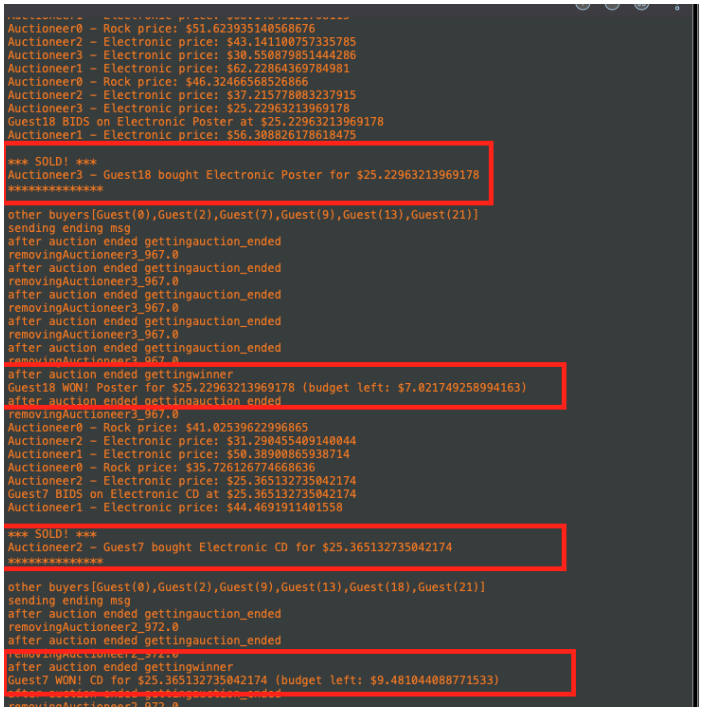
- **Use Case - 2 : Guest Participates in Multiple Auctions Simultaneously**
    - **Description :** Several auctioneers are running at the same time with different genres. A guest likes, for example, both Rock and Metal. The guest receives multiple CFP messages with different auction_ids and item_genres. For each matching genre and budget, the guest joins and adds the auction_id to active_auction_ids, potentially being active in 2–3 auctions at once. The guest later reacts independently to each auction's price updates using the stored auction_max_prices[auction_id].
    - **Screenshot of program execution/output :**

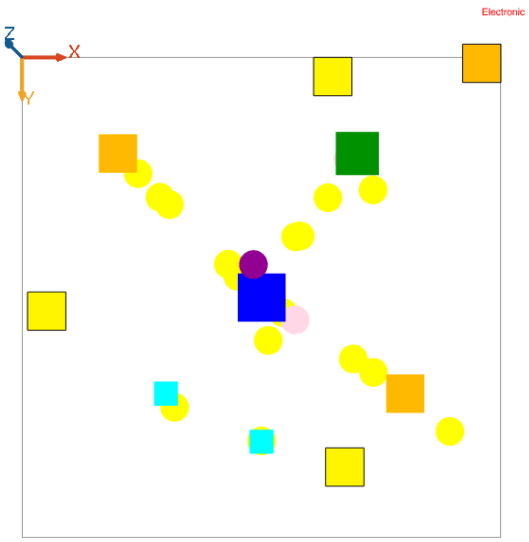


    - **Short explanation of the results:** The results show multiple auctioneers running Electronic-genre auctions at the same time, and each guest evaluates every CFP independently. Guests whose preferred genres include Electronic join the auction and print their personal maximum price, while others immediately refuse. Because two auctions are active simultaneously, guests may join one, both, or neither depending on their preferences and budgets. This confirms that genre-based filtering,

simultaneous auction handling, and parallel guest participation all work correctly.

- **Use Case - 3 : Guest Wins an Auction and Updates State**
    - **Description :** The price of an item drops below the guest's max_willing_to_pay for a specific auction_id. The guest sends a PROPOSE message with its bid. The auctioneer accepts the first valid proposal, sends an INFORM message with "winner", auction_id, item_name, final_price, and ends that auction. The guest receives the INFORM, adds the item to owned_merch, subtracts final_price from auction_budget, and removes the corresponding auction_id from active_auction_ids and auction_max_prices.
    - **Screenshot of program execution/output :**

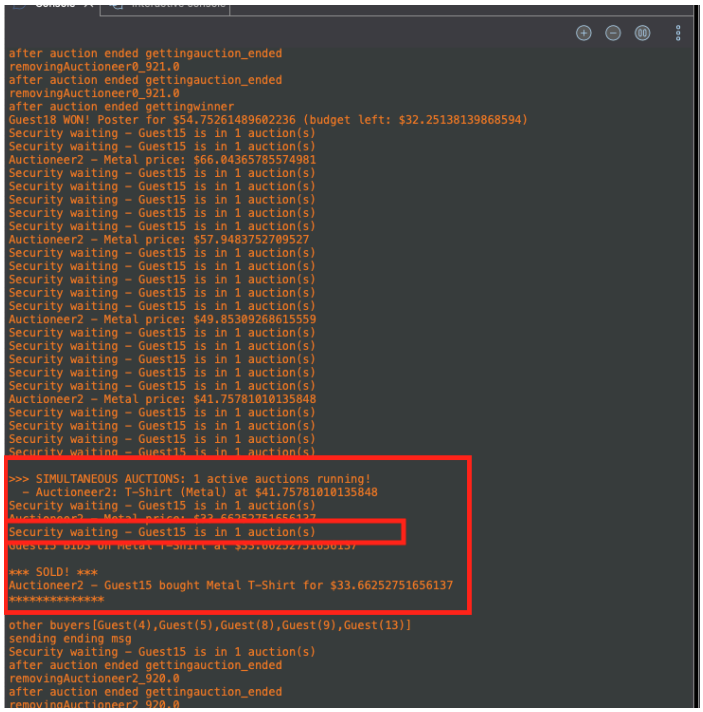




    - **Short explanation of the results:** The results show that two different auctioneers complete their Electronic-genre auctions independently. In the first auction, Guest18 places the first valid bid and wins the Electronic Poster, and all other participants are notified that the auction has ended. Shortly after, another auctioneer finishes a separate auction where Guest7 wins the Electronic CD. Each guest's budget is updated correctly, and all joined auction IDs are removed, confirming that multiple Dutch auctions

run in parallel, winners are selected properly, and guests clean up their state after each auction.

- **Use Case - 4 : Security Guard Waits for Guest to Finish Active Auctions**
    - **Description :** A guest who is currently participating in one or more auctions is flagged by the Security agent as a rule-breaker. When the security guard tries to approach and remove that guest, it checks the guest's active_auction_ids. Since the guest is still involved in active auctions, the guard does not intervene immediately and waits until all those auctions end. As soon as the guest receives INFORM messages that the auctions have finished and their active_auction_ids list becomes empty, the security guard resumes pursuit and removes the guest normally. This ensures that ongoing auctions are not disrupted by external agent behavior.
    - **Screenshot of program execution/output :**



    - **Short explanation of the results:** The output shows that Guest15 is involved in an active auction, and as a result the Security agent keeps waiting instead of acting. Every cycle, the system prints "Security waiting – Guest15 is in 1 auction(s)", confirming that security correctly checks the guest's active auctions before intervening. Only after Guest15 wins the Metal T-Shirt and the auction finishes does the auctioneer send the end messages, allowing Guest15's auction list to clear. This demonstrates that

auctions are not interrupted by external agents and that the security guard waits until the guest is no longer participating in any auction.

6. Challenge 2:

6.1 : Explanation: Challenge 2 is about running different auction mechanisms in the same festival and comparing their behaviour. We needed to support Dutch, English, and Sealed-bid auctions in parallel and then analyse which type sells more items, generates more revenue, and how much “buyer saving” (difference between willingness to pay and final price) each type gives.

We created several Auctioneer agents and assigned each one an auction_type ("Dutch", "English", or "Sealed"). The auctioneer logic then branches on this type: Dutch reduces the price until the first bid; English keeps raising the current price and tracks the highest bidder until the time limit; Sealed collects all bids once and selects the highest at the end. All communication still uses FIPA CFP/PROPOSE/INFORM messages, so guests can participate in any type with the same interface. In the global section we maintain separate counters for each type (total, successful, revenue, buyer_savings lists) and periodically print/plot success rate, average revenue, and average buyer savings to compare the three mechanisms.

6.2 : Code:

- **Relevant Code Snippet(s) :**

```
species Auctioneer skills: [fipa] {
    rgb my_color <- #gold;
    string auction_type <- "Dutch";

    bool auction_active <- false;
    string current_auction_id;
    string item_name;
    string item_genre;
    float starting_price;
    float current_price;
    float minimum_price;
    float price_change;
    float action_interval <- 5.0;
    float last_action_time;
```

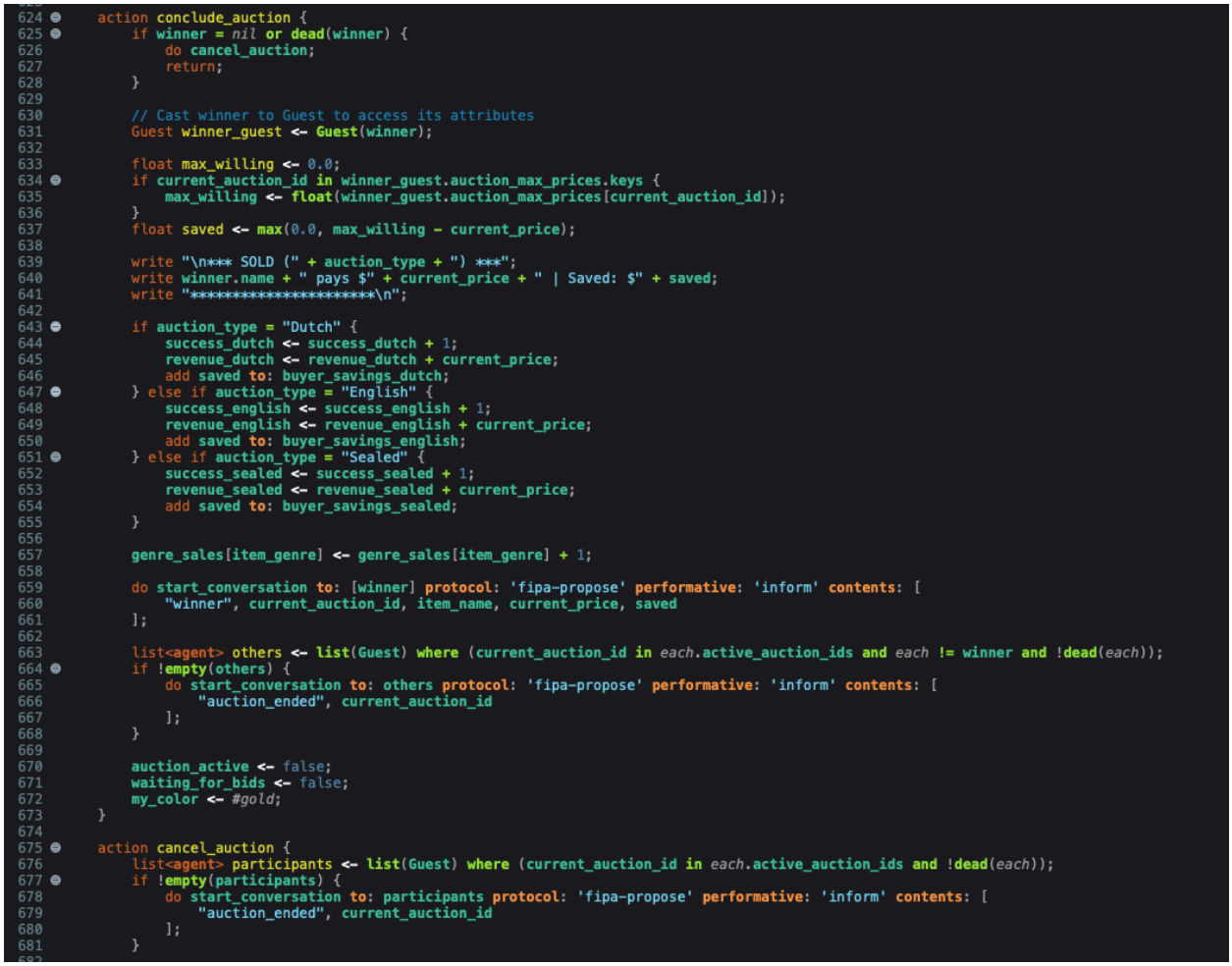
```
float auction_duration <- 25.0;
float auction_start_time;

map<agent, float> all_bids <- map([]);
agent current_highest_bidder <- nil;
float current_highest_bid <- 0.0;
agent winner <- nil;
bool waiting_for_bids <- false;

action start_auction {
    if !auction_active {
        auction_active <- true;
        current_auction_id <- name + "_" + string(time);

        item_name <- one_of(merch_items);
        item_genre <- one_of(merch_genres);
        starting_price <- rnd(40.0, 100.0);
        minimum_price <- starting_price * 0.25;
        last_action_time <- time;
        auction_start_time <- time;

        all_bids <- map([]);
        current_highest_bidder <- nil;
        current_highest_bid <- 0.0;
        winner <- nil;
        waiting_for_bids <- true;

        if auction_type = "Dutch" {
            current_price <- starting_price;
            price_change <- starting_price * rnd(0.06, 0.12);
            my_color <- #orange;
            total_dutch <- total_dutch + 1;
        } else if auction_type = "English" {
            current_price <- starting_price * 0.4;
            price_change <- 5.0;
            my_color <- #cyan;
            total_english <- total_english + 1;
        } else if auction_type = "Sealed" {
            current_price <- starting_price;
            auction_duration <- 15.0;
```

```
            my_color <- #darkgreen;
            total_sealed <- total_sealed + 1;
        }

        write "\n========== " + auction_type + " AUCTION ==========";
        write name + ": " + item_name + " (" + item_genre + ") - $" + starting_price;
        write "====================================";

        do start_conversation to: list(Guest) protocol: 'fipa-propose' performative: 'cfp'
contents: [
            current_auction_id, item_name, item_genre, starting_price, current_price,
auction_type, "auction_start"
        ];
    }
  }
```

- **Code Screenshots:**

```
    reflex receive_refuses when: auction_active and !empty(refuses) {
        // Acknowledge
    }

    reflex receive_bids when: auction_active and waiting_for_bids and !empty(proposes) {
        loop bid_msg over: proposes {
            list bid_data <- list(bid_msg.contents);
            if length(bid_data) >= 2 {
                agent bidder <- bid_msg.sender;
                float bid_price <- float(bid_data[1]);

                if auction_type = "Dutch" {
                    winner <- bidder;
                    current_price <- bid_price;
                    do conclude_auction;
                    return;
                } else if auction_type = "English" {
                    if bid_price > current_highest_bid {
                        current_highest_bidder <- bidder;
                        current_highest_bid <- bid_price;
                        current_price <- bid_price;
                        last_action_time <- time;
                        write name + " - High bid: $" + bid_price;
                    }
                } else if auction_type = "Sealed" {
                    if !(bidder in all_bids.keys) or bid_price > all_bids[bidder] {
                        all_bids[bidder] <- bid_price;
                    }
                }
            }
        }
    }
```

```
    reflex manage_auction when: auction_active and waiting_for_bids {
        float elapsed <- time - last_action_time;
        float total_elapsed <- time - auction_start_time;

        if auction_type = "Dutch" and elapsed >= action_interval {
            current_price <- current_price - price_change;

            if current_price < minimum_price {
                write name + " - Dutch CANCELLED";
                do cancel_auction;
            } else {
                write name + " - Price: $" + current_price;
                list<agent> interested <- list(Guest) where (current_auction_id in each.active_auction_ids and !dead(each));
                if !empty(interested) {
                    do start_conversation to: interested protocol: 'fipa-propose' performative: 'cfp' contents: [
                        current_auction_id, item_name, item_genre, starting_price, current_price, auction_type, "price_update"
                    ];
                }
                last_action_time <- time;
            }
        } else if auction_type = "English" {
            if total_elapsed >= auction_duration {
                if current_highest_bidder != nil {
                    winner <- current_highest_bidder;
                    current_price <- current_highest_bid;
                    do conclude_auction;
                } else {
                    write name + " - English CANCELLED";
                    do cancel_auction;
                }
            } else if elapsed >= 3.0 {
                list<agent> interested <- list(Guest) where (current_auction_id in each.active_auction_ids and !dead(each));
                if !empty(interested) {
                    do start_conversation to: interested protocol: 'fipa-propose' performative: 'cfp' contents: [
                        current_auction_id, item_name, item_genre, starting_price, current_price, auction_type, "price_update"
                    ];
                }
                last_action_time <- time;
            }
        } else if auction_type = "Sealed" and total_elapsed >= auction_duration {
            if !empty(all_bids) {
                float highest <- max(all_bids.values);
                loop bidder over: all_bids.keys {
                    if all_bids[bidder] = highest {
                        winner <- bidder;
                        current_price <- highest;
                        break;
                    }
                }
                write name + " - Sealed: " + length(all_bids) + " bids";
                do conclude_auction;
            } else {
                write name + " - Sealed CANCELLED";
                do cancel_auction;
            }
        }
    }
```



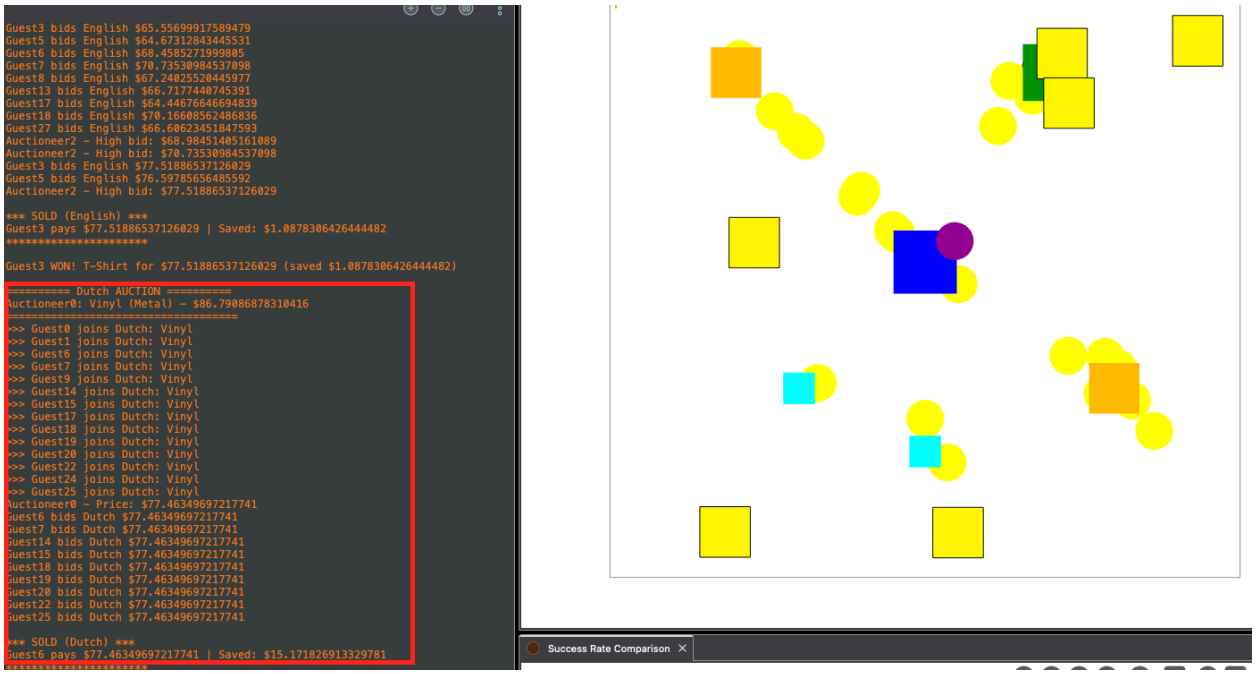
```
    action conclude_auction {
        if winner = nil or dead(winner) {
            do cancel_auction;
            return;
        }

        // Cast winner to Guest to access its attributes
        Guest winner_guest <- Guest(winner);

        float max_willing <- 0.0;
        if current_auction_id in winner_guest.auction_max_prices.keys {
            max_willing <- float(winner_guest.auction_max_prices[current_auction_id]);
        }
        float saved <- max(0.0, max_willing - current_price);

        write "\n*** SOLD (" + auction_type + ") ***";
        write winner.name + " pays $" + current_price + " | Saved: $" + saved;
        write "**********************\n";

        if auction_type = "Dutch" {
            success_dutch <- success_dutch + 1;
            revenue_dutch <- revenue_dutch + current_price;
            add saved to: buyer_savings_dutch;
        } else if auction_type = "English" {
            success_english <- success_english + 1;
            revenue_english <- revenue_english + current_price;
            add saved to: buyer_savings_english;
        } else if auction_type = "Sealed" {
            success_sealed <- success_sealed + 1;
            revenue_sealed <- revenue_sealed + current_price;
            add saved to: buyer_savings_sealed;
        }

        genre_sales[item_genre] <- genre_sales[item_genre] + 1;

        do start_conversation to: [winner] protocol: 'fipa-propose' performative: 'inform' contents: [
            "winner", current_auction_id, item_name, current_price, saved
        ];

        list<agent> others <- list(Guest) where (current_auction_id in each.active_auction_ids and each != winner and !dead(each));
        if !empty(others) {
            do start_conversation to: others protocol: 'fipa-propose' performative: 'inform' contents: [
                "auction_ended", current_auction_id
            ];
        }

        auction_active <- false;
        waiting_for_bids <- false;
        my_color <- #gold;
    }

    action cancel_auction {
        list<agent> participants <- list(Guest) where (current_auction_id in each.active_auction_ids and !dead(each));
        if !empty(participants) {
            do start_conversation to: participants protocol: 'fipa-propose' performative: 'inform' contents: [
                "auction_ended", current_auction_id
            ];
        }
```

- **Explanation of the code :** This code handles bidding and auction progression for all three auction types—Dutch, English, and Sealed Bid—inside the same Auctioneer. When a PROPOSE message arrives, the auctioneer checks the auction type: Dutch immediately accepts the first bid, English updates the current highest bid and bidder, and Sealed stores each bidder's offer without revealing anything. The `manage_auction` reflex then updates prices or decides when each auction type should end. Dutch reduces the price until someone bids or the minimum price is reached; English keeps broadcasting updates until its time limit expires; Sealed waits until its full duration is over, then selects the highest stored bid. The `conclude_auction` action finalizes the sale, updates success and revenue statistics for the correct auction type, calculates buyer savings, and sends INFORM messages to both the winner and all other participants. This structure lets all three auction mechanisms run correctly and independently using the same FIPA communication flow.

6.3 : Demonstration:

- **Use Case - 1 : Dutch Auction – First Bidder Wins Immediately**
    - **Description :** The auctioneer starts a Dutch auction with a high initial price and gradually reduces it. Guests who joined the auction wait for the price to drop. As soon as the price becomes acceptable for one guest, that guest sends a PROPOSE message. Because Dutch auctions award the item to the first bidder, the auctioneer immediately accepts the proposal, sends an INFORM “winner”, and ends the auction.

    - **Screenshot of program execution/output :**

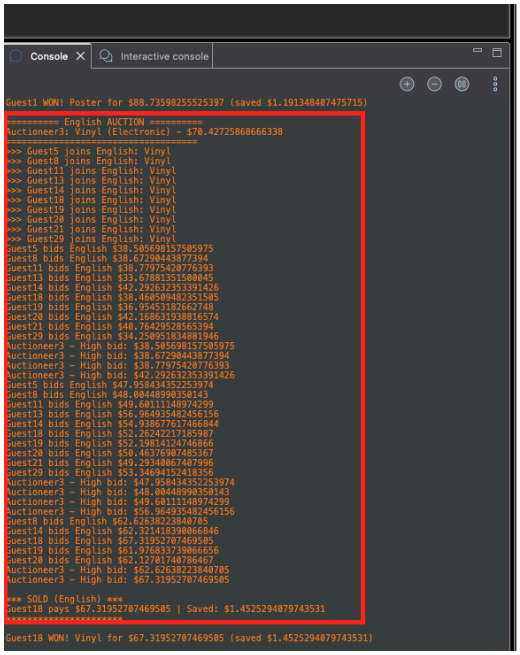


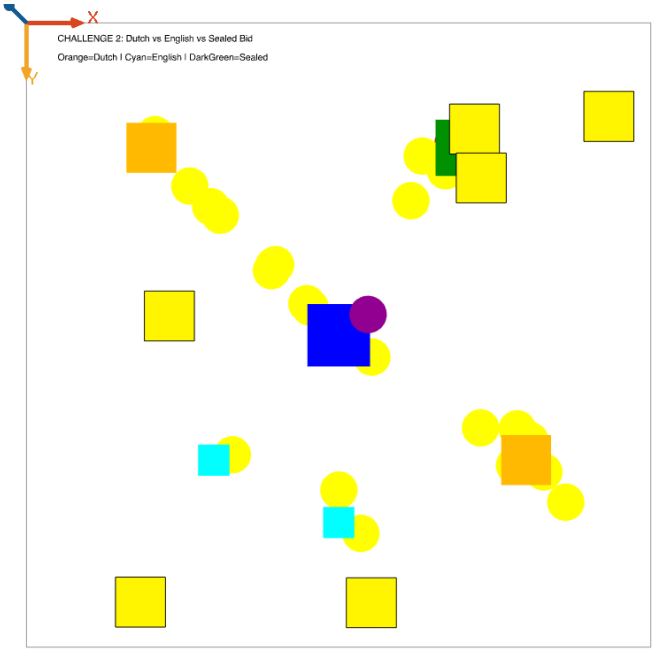
    - **Short explanation of the results:** The simulation shows two different auction types running one after another. First, the English auction completes: multiple guests place progressively higher bids, the price rises, and Guest3 becomes the highest bidder by the time the auction ends, winning the T-Shirt for $77.51. Immediately after, a Dutch auction begins for a Vinyl item. Many guests join based on genre preference, and as the auctioneer lowers the price, Guest7 sends the first acceptable bid and wins the item at $77.46. Both auction types operate correctly, with guests reacting according to the rules of each mechanism and the system logging the winner, price paid, and their savings.

- **Use Case - 2 : English Auction – Highest Bidder Wins After Time Limit**

- **Description :** An English auctioneer broadcasts a starting price and lets guests place bids above the current highest bid. Whenever a guest bids higher, the auctioneer updates the current price and highest bidder. When the auction duration ends, the auctioneer checks the final highest bid and sends an INFORM “winner” to that guest. If no one bids in time, the auction is cancelled.

- **Screenshot of program execution/output :**



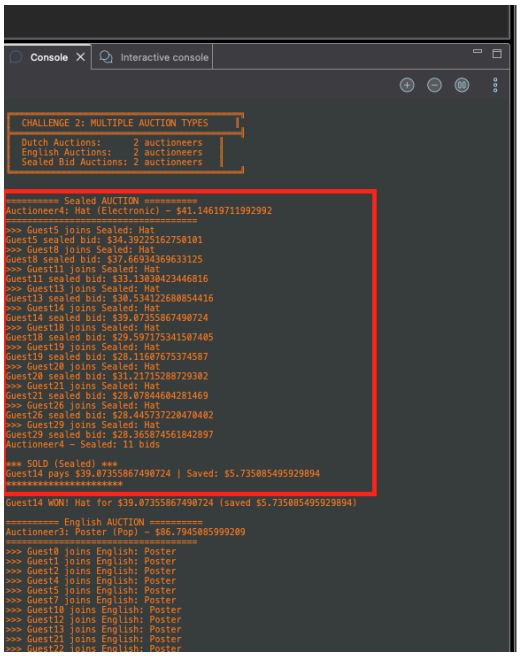


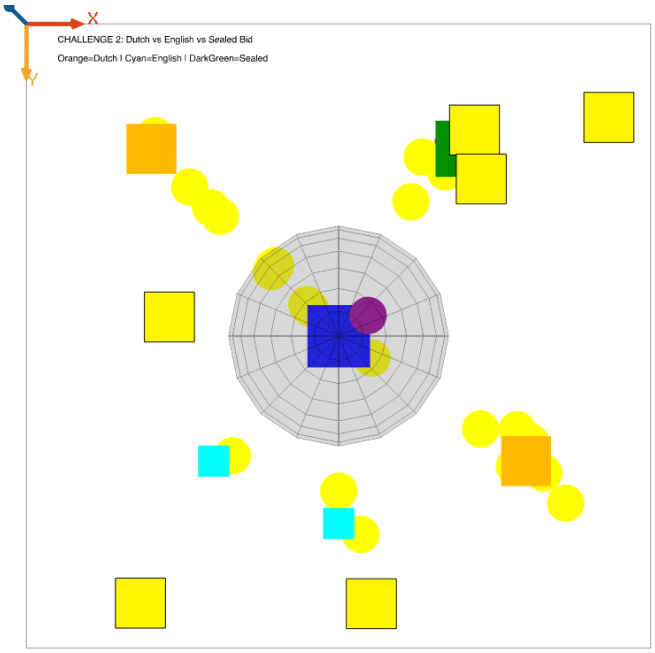
- **Short explanation of the results:**

- **Use Case - 3 : Sealed-Bid Auction – Winner Chosen After All Bids Collected**

  - **Description :** In a sealed-bid auction, guests submit one private bid without seeing others’ offers. The auctioneer collects all submitted bids during the auction duration but does not reveal any information. When time expires, the auctioneer evaluates all recorded bids, selects the highest bidder as the winner, sends an INFORM “winner”, and ends the auction. All losing bidders receive an “auction_ended” message.

- **Screenshot of program execution/output :**

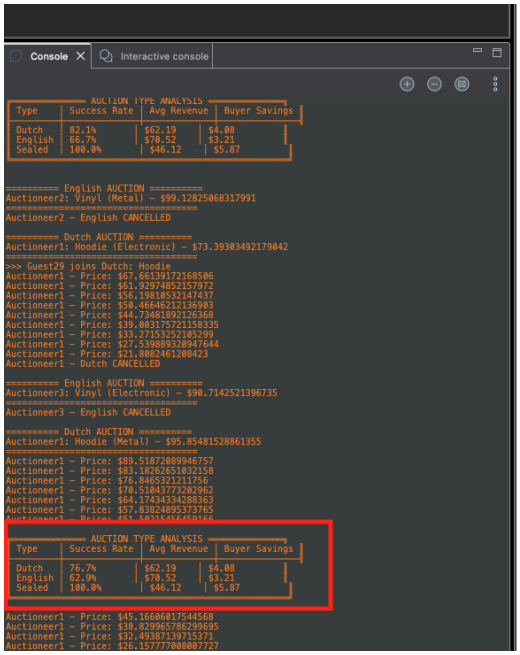


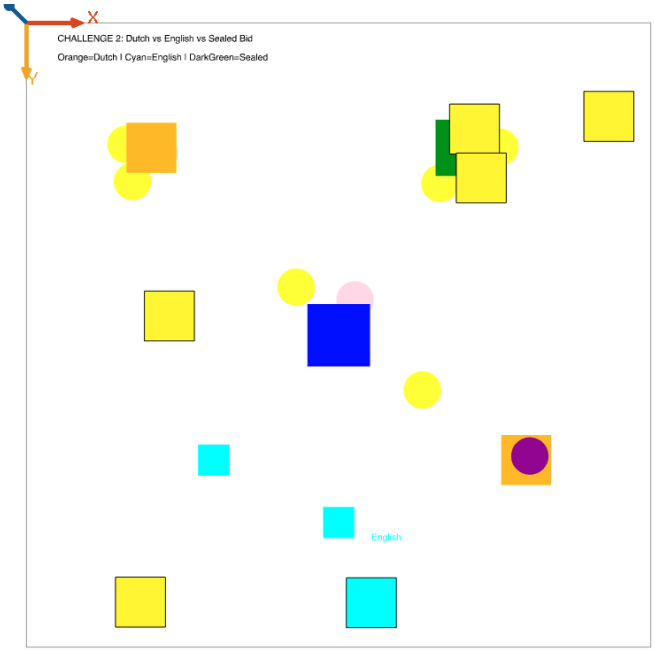


- **Short explanation of the results:**

- **Use Case - 4 : Auction Type Analysis**
  - **Description :** The auction type analysis compares the performance of the three implemented mechanisms—Dutch, English, and Sealed-Bid—over the entire simulation. For each auction type, the model tracks how many auctions were conducted, how many succeeded, the average revenue generated, and how much buyers saved compared to their maximum willingness to pay. These statistics are aggregated during the run and periodically displayed as text tables and charts. This allows us to observe which auction type sells items more reliably, which earns higher revenue, and which gives buyers better deals. In short, the analysis provides a direct performance comparison of the three auction formats under identical festival conditions.

- **Screenshot of program execution/output :**





- **Short explanation of the results:** The results table compares the performance of Dutch, English, and Sealed-bid auctions based on three metrics: success rate, average revenue, and buyer savings. In this run, Dutch auctions sell most items consistently with a high success rate and strong revenue. English auctions show lower success because some expire without bids, but still yield good revenue when they do succeed. Sealed-bid auctions have the highest success rate because all bids are collected silently, and a winner is chosen even when bidding is sparse. Buyer savings differ across types: Dutch gives moderate savings, English slightly lower, and Sealed-bid gives the highest savings since bidders often bid below their maximum willingness. Overall, the table summarizes how each mechanism performs under identical festival conditions.

7. Final Remarks :

- **Summarize :** The project successfully implemented a festival simulation enriched with intelligent agent interactions, FIPA-based communication, and multiple auction mechanisms (Dutch, English, Sealed). Guests made autonomous decisions based on preferences and budgets, while auctioneers

conducted parallel auctions reliably. The system demonstrated realistic negotiation, agent cooperation, and dynamic environment behavior.

- **Limitations or Improvements :**
  Guest decision rules can be expanded (e.g., learning or adaptive bidding).
  Better visualization and logs could help interpret complex auction flows.
  Some auctions may still overlap heavily, causing message congestion.