Faculty of Science of University of Lisbon

Multi-Agent Systems 2024/2025

Project - Zanzibar Bazaar

64465 - Duarte Gonçalves

64699 - Tommaso Tragno

Professor:
Luís Moniz

January 26, 2025

Contents

1 Introduction

1.1 Context and Motivation

The Zanzibar Bazaar game scenario models a historical (albeit fictionalized) 18th-century spice market in the port of Zanzibar, where multiple merchant agents negotiate trades, respond to market fluctuations, and try to maximize their profits. Multi-Agent System (MAS) concepts are particularly well-suited to modeling such a dynamic, distributed environment, where each agent must operate autonomously yet communicate effectively to achieve its goals.

1.2 Objectives

Our project focuses on designing, implementing, and demonstrating a multi-agent system that:

- Encapsulates strategic negotiation between autonomous merchant agents.
- Employs a BDI-like model (Beliefs, Desires, Intentions) for decision-making with cognitive architectures capable of:
  - Forming beliefs about the market based on historical and current information.
  - Establishing desires to maximize profits and capitalize on favorable market conditions.
  - Translating these beliefs and desires into actionable intentions, such as trade proposals, alliances, or inventory management.
- Reacts to random market events (storms, port taxes, trade routes) introduced by a Game Master agent (the `BazaarAgent`).
- Showcases alliances, rivalries, and sabotage behaviors among merchant agents.

1.3 Report Structure

This report is structured into seven sections:

- **Section 1** provides an introduction, motivation, and scope of the project.
- **Section 2** discusses relevant work in multi-agent systems with a focus on trade simulations and cognitive architectures.
- **Section 3** (the core) details the system design and agent architecture, including the BDI model.
- **Section 4** explains the implementation in JADE, covering agent behaviors and event handling.
- **Section 5** presents the results and evaluation of the system, including logs and performance observations.
- **Section 6** provides a discussion of the strengths, limitations, and possible extensions.

- **Section 7** concludes the report and outlines final remarks.

The project code is available on GitHub [4]
The simulation output and additional materials are provided in the appendices.

2 Related Work

2.1 Multi-Agent Systems in Trade Simulations

Multi-agent simulations of markets and trading have been employed in numerous research contexts to test negotiation protocols, dynamic pricing strategies, and agent learning mechanisms [2]. Classic examples include fish markets and electronic auctions [1], illustrating how MAS frameworks can capture realistic negotiation cycles and produce emergent phenomena like price convergence.

2.2 Cognitive Architectures for Intelligent Agents

BDI (Belief-Desire-Intention) architectures are a staple for modeling decision-making in cognitive agents [3]. Although JADE itself is not a BDI-specific framework, it supports the creation of agent behaviors that can effectively implement the BDI cycle through its behavior-based structure. Our agent's beliefs, desires, and intentions are stored as class variables, updated dynamically, and acted upon via negotiation or sabotage behaviors.

2.3 Contribution

Our implementation of the Zanzibar Bazaar demonstrates:

- A turn-based trade simulation using JADE, with dynamic events and flexible agent behavior.
- A BDI-like approach where each merchant updates its beliefs from the market signals, forms desires to maximize profit or sabotage rivals, and executes these desires through intentions (e.g., trade proposals).
- Integration of alliances and rivalries, which affect trade acceptance thresholds and sabotage attempts.

3 System Design and Architecture

3.1 Overview

Our MAS comprises:

1. **The `BazaarAgent`** (Game Master), which:
    - Manages rounds, from 1 to `MAX_ROUNDS`.
    - Announces market prices and random events each round.

   - Requests and processes sales from all merchants.
   - Adjusts prices (supply/demand + event effect) for the next round.

2. **Multiple `AdvancedPlayerAgent`**, each:
   - Registers in the Directory Facilitator (DF) as a `zanzibar-player`.
   - Maintains beliefs about current prices, events, and predictions.
   - Forms intentions to propose or accept trades, respond to alliances, sabotage rivals, and decide how many spices to sell at round's end.

3. **Multiple `SimplePlayerAgent`:**
   - Similar to the *AdvancedPlayerAgent*, registers as `zanzibar-player`;
   - Doesn't have any beliefs, their decision is solely random;
   - As their Intentions are purely randomized;

3.2 Agent Cognitive Architecture

While JADE does not enforce a specific BDI architecture, we have mapped:

- **Beliefs:**
  - `currentPrices`, `historicalPrices`, and `currentEvent` for market knowledge.
  - `inventory` and `coins` for self-knowledge.
  - `otherPlayers`, `allies`, `rivals` for social knowledge.
- **Desires:**
  - Maximize short-term profit.
  - Adapt to predicted price changes (sell if price will drop, hold if it will rise).
  - Seek alliances or sabotage rivals to gain advantage.
- **Intentions:**
  - Propose trades, accept or reject incoming proposals.
  - Finalize a "sell decision" at the end of each round.
  - Possibly sabotage a rival if beneficial.

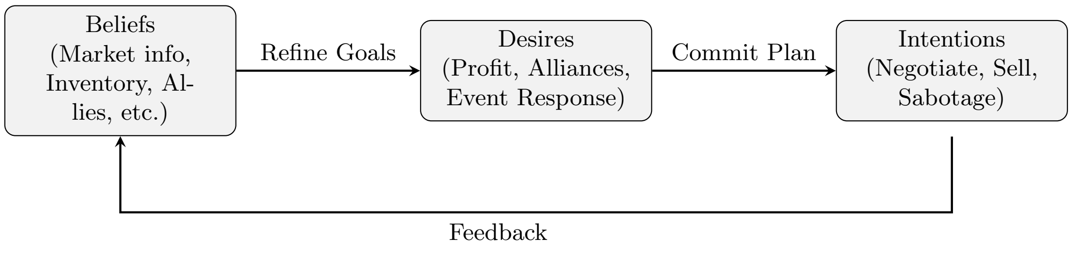

Beliefs (Market info, Inventory, Allies, etc.)
Refine Goals
Desires (Profit, Alliances, Event Response)
Commit Plan
Intentions (Negotiate, Sell, Sabotage)
Feedback

Figure 1: BDI-like cognitive flow in each `AdvancedPlayerAgent`.

3.3 Non-BDI SimplePlayerAgent

In addition to the `AdvancedPlayerAgent` (which uses a BDI-like model), we introduced a simpler `SimplePlayerAgent` to compare how an agent lacking explicit beliefs, desires, and intentions would perform. This `SimplePlayerAgent`:

- Registers as a `zanzibar-player` in the DF.
- Maintains a budget and a small inventory of spices but does not store historical prices or social relationships like alliances or rivalries.
- Randomly proposes trades to any discovered merchant. It may accept or reject incoming trade proposals based on a coin toss (i.e., 50% acceptance).
- Chooses to sell a spice or withhold sales using a similarly simplistic random approach when the Bazaar requests "sell decisions."

Whereas the `AdvancedPlayerAgent` aims to predict next-round prices and form alliances to improve negotiation outcomes, this `SimplePlayerAgent` uses limited reasoning and random acceptance criteria. In effect, the `SimplePlayerAgent` serves as a baseline or control agent, illustrating how a less "intelligent" merchant typically fares in the spice bazaar.

3.4 Bazaar Master Agent and Round Logic

The `BazaarAgent` orchestrates the overall gameplay by managing each round in a sequential fashion. It coordinates six main steps each round, shown in Figure 2. After processing these steps, it checks whether the current round number has reached (or exceeded) the `maxRounds` threshold. If it has, it triggers *game-over*, broadcasting a final termination message to all active merchants; otherwise, it starts a new round.

1. **UpdatePlayerList**: At the beginning of each round, the BazaarAgent queries the Directory Facilitator (DF) for all agents registered as `zanzibar-player`. It then adds new arrivals to the scoreboard (with an initial score of 0) and moves any departed players from the scoreboard to a *departedPlayers* record.

2. **RoundBegin**: Next, the BazaarAgent broadcasts a *round-start* message to all active players, indicating that a new round is about to commence. This message signals that negotiation and trading behaviors in each `AdvancedPlayerAgent` may now begin.

3. **AnnouncePrices**: The agent randomly generates and applies an event (Storm, Sultan's Port Tax, or New Trade Route) for the round. It then sends a *price-announcement* to inform each merchant of the current spice prices and the active event (e.g., "Storm in Indian Ocean: Next round Clove price +5"). This way, every player updates their internal *beliefs* about the market before trading starts.

4. **RequestSales**: The BazaarAgent issues a *sell-request* to each active player, asking how many units of each spice they wish to sell this round. From the agent's perspective, this marks the transition from negotiation into finalizing each merchant's intentions for the round.

5. **CollectSales**: The agent waits for *sell-response* messages. Upon receiving each response, it calculates how many coins the merchant has earned, applies any taxes if the event is a *Sultan's Port Tax*, and updates the scoreboard accordingly. At the same time, it accumulates the total quantity of each spice sold, needed for supply/demand price adjustments.

6. **EndRound**: Finally, it prints the updated scoreboard, adjusts prices for the next round (supply/demand + event effects), and then checks whether the current round has reached `maxRounds`. If `currentRound < maxRounds`, a new round is scheduled; otherwise, a *game-over* message is sent to all players, and the bazaar terminates.

Game-Over Action. When the condition `currentRound` $\geq$ `maxRounds` is met, the `BazaarAgent` sends a *game-over* `INFORM` message to all remaining `AdvancedPlayerAgents`, instructing them to finalize any internal states and call `doDelete()`. At this point, no further rounds or negotiations occur, and the `BazaarAgent` also terminates its own execution after a short grace period. This ensures a clean, coordinated shutdown for all participating agents.

3.5 Communication Protocols

We use FIPA-compliant message exchange to:

- `INFORM` players of `round-start` and `price-announcement`.
- `REQUEST` each player to `sell-request`.
- `PROPOSE` and `ACCEPT_PROPOSAL` or `REJECT_PROPOSAL` for negotiations between merchants.
- `INFORM` for sabotage or rumor messages.

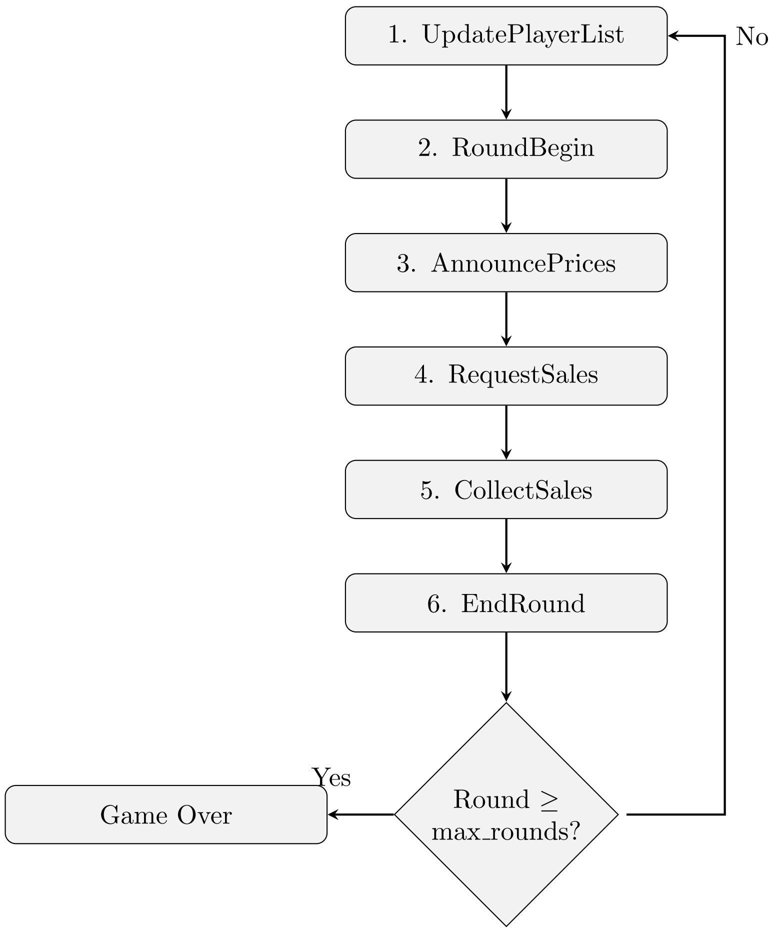


Figure 2: BazaarAgent round logic. After finishing the six main steps, it checks if the current round number is at least `maxRounds`. If not, the cycle repeats from *UpdatePlayerList*; otherwise, the system broadcasts a *game-over* and terminates.

4 Implementation

4.1 Platform and Tools

We used:

- **JADE 4.6.0**: Java Agent DEvelopment Framework for agent lifecycle and messaging.
- **Sniffer Agent**: Provided by JADE to monitor inter-agent communication (Figures shown in logs).
- **Java 11 or later**.

4.2 Agent Behaviors

Each `AdvancedPlayerAgent` includes the following behaviors:

1. **MarketMessageListener (Cyclic):** Receives `round-start`, `price-announcement`, and `sell-request` from `BazaarAgent`.
2. **NegotiationInitiatorBehavior (Ticker):** Periodically proposes trades to other players until a negotiation success or maximum attempts reached.
3. **NegotiationResponder (Cyclic):** Handles incoming `trade-offer` proposals (accept or reject).
4. **AllianceProposerBehavior (Ticker):** Occasionally tries to form alliances with random or strategic partners.
5. **AllianceResponder (Cyclic):** Evaluates incoming alliance proposals.
6. **SabotageBehavior (Ticker):** Optionally sends misleading or rumor messages to a rival at intervals.

4.3 Simplified SimplePlayerAgent Implementation

The `SimplePlayerAgent` code differs from the `AdvancedPlayerAgent` primarily in its lack of BDI-style data structures. Instead, it uses:

- A `budget` variable to track coins.
- A random negotiation acceptance or rejection scheme.
- Minimal logic for deciding how much to sell (often picks a random spice to sell and a random quantity).
- A `NegotiationMakerBehaviour` that sporadically sends `PROPOSE` messages, and a `NegotiationResponderBehaviour` that accepts or rejects trades by coin flip.

It still interacts with the same marketplace as the `AdvancedPlayerAgent`, but without the memory or forecasting typical of BDI-based strategies. The result is an agent more prone to unprofitable trades and random sales decisions.

4.4 Market Dynamics

4.4.1 Storm, Sultan's Tax, New Trade Route

- **Storm:** Raises price of a specified spice in the next round.
- **Sultan's Port Tax:** Applies a random percentage tax on sales for the current round.
- **New Trade Route:** Multiplies the future price of a specified spice by a factor (0.3 to 0.7 range).

4.4.2 Supply/Demand Logic

At the end of each round, the `BazaarAgent` adjusts prices:

- If no units of a spice were sold, its price is increased slightly for scarcity.
- If many units (e.g., >10) were sold, price decreases slightly.
- Price is bounded at a minimum of 1 coin.

4.5 Code Organization

- **BazaarAgent.java** implements the master logic
- **AdvancedPlayerAgent.java** holds the merchant code (BDI-like data structures)
- **SimplePlayerAgent.java** implements the random-driven agent without BDI data structures.

(**NOTE:** The full code is provided on GitHub [4].)

5 Results and Evaluation

5.1 Experimental Setup

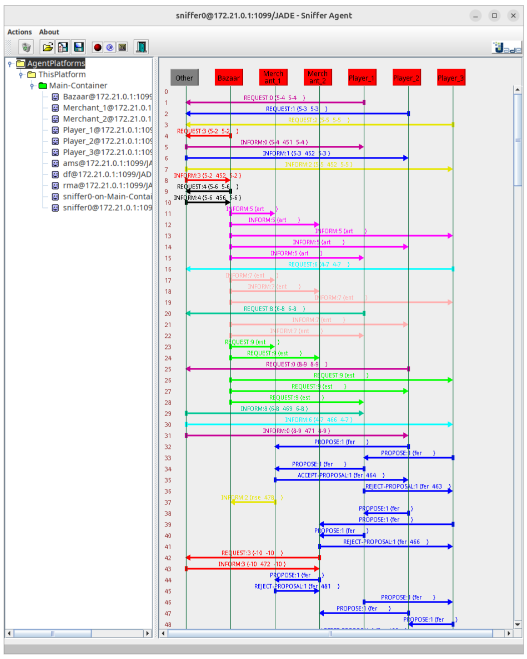
We ran a simulation with:

- **1 BazaarAgent** controlling 5 rounds (`maxRounds=5`).
- **3 AdvancedPlayerAgents**, a BDI-like merchants each with randomly generated initial inventories and coins.
- **2 SimplePlayerAgents**, a non BDI-like merchants, random-based, used to validate the BDI agent.
- Each merchant attempts trades every couple of seconds, up to 3 attempts per round.
- Alliances and sabotage behaviors also attempt periodically (10 seconds interval).

5.2 Simulation Results

Figure 3 shows a snippet of the Sniffer Agent logs (sequence diagram), capturing `PROPOSE` and `REQUEST` messages among merchants and the `BazaarAgent`. Each round:

1. `BazaarAgent` announces an event (e.g., "New Trade Route: Clove next round price x0.55").
2. Merchants scramble to propose trades ("OFFER:Cinnamon=2 → REQUEST:Clove=1").
3. Most proposals are rejected due to unfavorable exchange ratios or rivalry constraints.
4. Eventually, merchants hit their negotiation fallback and send their `sell-response`.



```
REQUEST:0 (5-4  5-4   )
REQUEST:1 (5-3  5-3   )
REQUEST:2 (5-5  5-5   )
REQUEST:3 (5-2  5-2   )
INFORM:0 (5-4  451  5-4 )
INFORM:1 (5-3  452  5-3 )
INFORM:2 (5-5  452  5-5 )
INFORM:3 (5-2  452  5-2 )
REQUEST:4 (5-6  5-6   )
INFORM:4 (5-6  456  5-6 )
INFORM:5 (art      )
INFORM:5 (art      )
INFORM:5 (art      )
INFORM:5 (art      )
INFORM:5 (art      )
REQUEST:6 (4-7  4-7   )
INFORM:7 (ent      )
INFORM:7 (ent      )
INFORM:7 (ent      )
REQUEST:8 (6-8  6-8   )
INFORM:7 (ent      )
INFORM:7 (ent      )
REQUEST:9 (est      )
REQUEST:9 (est      )
REQUEST:0 (8-9  8-9   )
REQUEST:9 (est      )
REQUEST:9 (est      )
REQUEST:9 (est      )
INFORM:8 (6-8  469  6-8 )
INFORM:6 (4-7  466  4-7 )
INFORM:0 (8-9  471  8-9 )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
ACCEPT-PROPOSAL:1 (fer  464   )
REJECT-PROPOSAL:1 (fer  463   )
INFORM:2 (nse  478   )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
REJECT-PROPOSAL:1 (fer  466   )
REQUEST:3 (-10  -10   )
INFORM:3 (-10  472  -10 )
PROPOSE:1 (fer      )
REJECT-PROPOSAL:1 (fer  481   )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
PROPOSE:1 (fer      )
```

Figure 3: Snippet of JADE Sniffer capturing trade messages between Merchant Agents and the Bazaar.

5.2.1 Scoreboard Evolution

Below is the final scoreboard after Round 5 in this mixed scenario, corresponding to the logs we collected:

- **Player_1: 612 coins**
- **Player_2: 420 coins**
- **Player_3: 259 coins**
- **Merchant_1: 132 coins**
- **Merchant_2: 116 coins**

Notably, the three `AdvancedPlayerAgent` merchants generally accumulated higher final scores than the two `SimplePlayerAgent` merchants, consistent with the BDI-based forecasting and negotiation tactics employed by the `AdvancedPlayerAgent`s. The `SimplePlayerAgent`s tended to accept or reject trades randomly and often missed opportunities to exploit events like *Storm in Indian Ocean* or *New Trade Route*.

5.3 Analysis

This outcome highlights the advantage of a more informed decision-making process. The BDI-like approach to forecasting and alliance formation (even if partial) translates into better overall

profits. Meanwhile, the `SimplePlayerAgents`, operating with minimal strategy and random trade acceptance, typically fall behind in the scoreboard.
The contrast between these two agent types underscores how modeling mental states (or at least storing and acting upon explicit beliefs) can significantly impact a merchant's performance in a dynamic marketplace.
The system demonstration also confirms that:

- **Price Forecasting** influences selling decisions: agents attempt to hold a spice if it is predicted to increase, or dump it if a new route indicates a price drop. However, the simplified approach often leads to bulk-selling when the fallback is triggered.
- **Dynamic Events** significantly shape agent decisions (e.g., avoid selling if a Storm on `Clove` is predicted to raise that price soon).

6 Discussion

6.1 Strengths and Innovations

- **Flexible BDI-Like Design:** Each merchant's beliefs, desires, and intentions should be extended, for example by improving forecasting with Machine Learning.
- **Social Interactions:** Alliances and rivalries, while simple in this prototype, show how agent attitudes can affect negotiation outcomes.
- **Event-Driven Market Fluctuations:** The random event generator adds variety and unpredictability, encouraging adaptive strategies.

6.2 Challenges and Limitations

- **Frequent Rejections:** Our threshold logic for trade acceptance is conservative, leading to many unsuccessful negotiations.
- **Limited Alliance Impact:** Agents do not store or recall enough about successful/unfruitful trades, so alliances are somewhat superficial.
- **Sabotage Behavior is Minimal:** We occasionally send misleading rumors, but most agents ignore them. More complex deception logic or rumor acceptance would be interesting.
- **Scalability:** With more agents and longer rounds, more advanced coordination or computational strategies would be needed.

6.3 Future Work

Possible extensions include:

- **Adaptive Negotiation Strategies:** Machine learning or game-theoretic approaches to compute dynamic acceptance thresholds based on market states.

- **Deeper Social Reasoning:** Agents could maintain more nuanced records of trust, forming stable alliances or permanent rivalries.
- **Enhanced Forecasting:** Replace naive forecasting with time-series methods or Bayesian updates based on events and price history.
- **User Interaction:** Provide a GUI for human players to join as merchants alongside AI-driven agents.

7 Conclusion

We have presented a multi-agent simulation of the Zanzibar Bazaar using JADE, illustrating cognitive agent design BDI concepts. Through sequential rounds, the `BazaarAgent` regulated prices, introduced disruptive events, and tallied merchant profits, while the `AdvancedPlayerAgents` and the `SimplePlayerAgents` negotiated trades or alliances to maximize gains. The results demonstrate how dynamic events, alliances, and sabotage can shape outcomes in a competitive marketplace, and highlight the advantages of a BDI model.
This project highlights the power of multi-agent methodologies for simulating realistic trading environments with complex agent interactions, setting the stage for deeper research into negotiation protocols, strategic deception, and advanced forecasting in MAS contexts.

References

[1] S. Fike, T. Korzh, and B. Pagel. "An Auction-Based Approach for Multi-Agent Resource Allocation in E-Markets". In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, 2008, pp. 665–672.

[2] M. He, H.-F. Leung, and N. R. Jennings. "Agent-Based E-Market for Commodity Trading: Experimental Evaluation". In: *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2003, pp. 456–459.

[3] Anand S. Rao and Michael P. Georgeff. "BDI Agents: From Theory to Practice". In: *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS)*. San Francisco, USA, 1995, pp. 312–319.

[4] Tratom. *fcul-sma-project*. `https://github.com/tratom/fcul-sma-project`. Accessed: January 26, 2025.

A Simulation Output

A simulation output log from the end of round 4 to the end of round 5 is provided below, showing the sequence of messages exchanged between 3 `PlayerAgent`, 2 `MerchantAgent` and the `BazaarAgent`.

```
  --- End of Round 4 ---
  Scoreboard:
    Player_1: 611
    Player_2: 419
    Player_3: 258
    Merchant_1: 132
    Merchant_2: 116
  Prices after Round 4 adjustments: {Nutmeg=19, Clove=42, Cardamom=1,
Cinnamon=13}

  === Starting Round 5 ===
  Bazaar: Sent 'round-start' to players.
  [Merchant_1] Budget:[237] | Inventory: {Nutmeg=0, Clove=0, Cardamom=8,
Cinnamon=4}
  Player_2 received 'round-start'. Re-discovering players and enabling
negotiation...
  Player_3 received 'round-start'. Re-discovering players and enabling
negotiation...
  [Merchant_2] Budget:[218] | Inventory: {Nutmeg=0, Clove=0, Cardamom=4,
Cinnamon=6}
  Player_1 received 'round-start'. Re-discovering players and enabling
negotiation...
  Bazaar: Announcing event for Round 5 => Storm in Indian Ocean: Next round
Clove price +6
  Player_1 discovered 4 other player(s).
  > Sent price-announcement: Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm
in Indian Ocean: Next round Clove price +6
  [Merchant_2]: Received price-announcement:
    Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm in Indian Ocean:
Next round Clove price +6
  Bazaar: Requesting sales...
  Player_2 discovered 4 other player(s).
  Player_1 received price-announcement:
    Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm in Indian Ocean:
Next round Clove price +6
  Player_2 received price-announcement:
    Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm in Indian Ocean:
Next round Clove price +6
```

```
  Player_1: My naive forecast for next-round prices => {Nutmeg=19, Clove=48,
Cardamom=1, Cinnamon=12}
  Player_2: My naive forecast for next-round prices => {Nutmeg=19, Clove=48,
Cardamom=1, Cinnamon=12}
  [Merchant_1]: Received price-announcement:
    Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm in Indian Ocean:
Next round Clove price +6
  Player_2 received sell-request from Bazaar
  [Merchant_1]: Received sell-request from [Bazaar]
  [Merchant_2]: Received sell-request from [Bazaar]
  Player_1 received sell-request from Bazaar
  Player_3 discovered 4 other player(s).
  Player_3 received price-announcement:
    Nutmeg=19;Clove=42;Cardamom=1;Cinnamon=13;|EVENT:Storm in Indian Ocean:
Next round Clove price +6
  Player_3: My naive forecast for next-round prices => {Nutmeg=19, Clove=48,
Cardamom=1, Cinnamon=12}
  Player_3 received sell-request from Bazaar
  Player_1 -> Player_2: PROPOSE OFFER:Cinnamon=3 -> REQUEST:Clove=3
  Player_2 rejected trade from Player_1: OFFER:Cinnamon=3 -> REQUEST:Clove=3
  Player_3 -> Merchant_2: PROPOSE OFFER:Cinnamon=1 -> REQUEST:Clove=1
  [Merchant_2]: Received a Trade Proposal: OFFER:Cinnamon=1 -> REQUEST:Clove=1
  [Merchant_2]: Accepted the Trade Proposal
  [Merchant_2]: Accepted trade with Player_3: OFFER:Cinnamon=1 ->
REQUEST:Clove=1
  [Merchant_2]: Doesn't Want to Sell
  Merchant_2 sold {Nutmeg=0, Clove=0, Cardamom=0, Cinnamon=0} => 0 coins
(total 116)
  Player_2 -> Merchant_1: PROPOSE OFFER:Cinnamon=2 -> REQUEST:Clove=2
  [Merchant_1]: Received a Trade Proposal: OFFER:Cinnamon=2 -> REQUEST:Clove=2
  [Merchant_1]: Accepted the Trade Proposal
  [Merchant_1]: Accepted trade with Player_2: OFFER:Cinnamon=2 ->
REQUEST:Clove=2
  [Merchant_1]: Doesn't Want to Sell
  Merchant_1 sold {Nutmeg=0, Clove=0, Cardamom=0, Cinnamon=0} => 0 coins
(total 132)
  Player_1 -> Player_3: PROPOSE OFFER:Cinnamon=3 -> REQUEST:Clove=3
  Player_3 -> Player_2: PROPOSE OFFER:Cinnamon=1 -> REQUEST:Clove=1
  Player_2 -> Player_1: PROPOSE OFFER:Cinnamon=3 -> REQUEST:Clove=3
  Player_1 -> Merchant_1: PROPOSE OFFER:Cinnamon=2 -> REQUEST:Clove=2
  Player_3 -> Merchant_1: PROPOSE OFFER:Cinnamon=1 -> REQUEST:Clove=1
  [Merchant_1]: Received a Trade Proposal: OFFER:Cinnamon=1 -> REQUEST:Clove=1
  [Merchant_1]: Rejected the Trade Proposal
  Player_2 -> Player_3: PROPOSE OFFER:Cinnamon=3 -> REQUEST:Clove=3
  Player_1 reached max negotiation attempts without success. Fallback: ending
negotiation phase and forcing sell if pending.
  Player_3 reached max negotiation attempts without success. Fallback: ending
negotiation phase and forcing sell if pending.
  Player_3 -> sell-response: Nutmeg=0;Clove=0;Cardamom=1;Cinnamon=0
  Player_1 -> sell-response: Nutmeg=0;Clove=0;Cardamom=1;Cinnamon=0
  Player_3 inventory update: {Nutmeg=1, Clove=1, Cardamom=1, Cinnamon=1}
  Player_1 inventory update: {Nutmeg=1, Clove=1, Cardamom=1, Cinnamon=8}
  Player_3 sold {Nutmeg=0, Clove=0, Cardamom=1, Cinnamon=0} => 1 coins (total
259)
  Player_1 sold {Nutmeg=0, Clove=0, Cardamom=1, Cinnamon=0} => 1 coins (total
612)
  Player_2 reached max negotiation attempts without success. Fallback: ending
negotiation phase and forcing sell if pending.
  Player_2 -> sell-response: Nutmeg=0;Clove=0;Cardamom=1;Cinnamon=0
  Player_2 inventory update: {Nutmeg=1, Clove=1, Cardamom=1, Cinnamon=6}
  Player_2 sold {Nutmeg=0, Clove=0, Cardamom=1, Cinnamon=0} => 1 coins (total
420)

  --- End of Round 5 ---
  Scoreboard:
    Player_1: 612
    Player_2: 420
    Player_3: 259
    Merchant_1: 132
    Merchant_2: 116
  Prices after Round 5 adjustments: {Nutmeg=21, Clove=50, Cardamom=1,
Cinnamon=15}
  Reached MAX_ROUNDS = 5. Ending game...
  Bazaar: Sent 'game-over' to remaining players.
  [Merchant_2]: Received 'game-over' from GameMaster [Bazaar]
  [Merchant_1]: Received 'game-over' from GameMaster [Bazaar]
  [Merchant_2]: Ended Game with 218 points
  [Merchant_1]: Ended Game with 237 points
```

B Sniffer Agent Screenshot

The communication messages captured from the Sniffer Agent during the simulation are shown below:

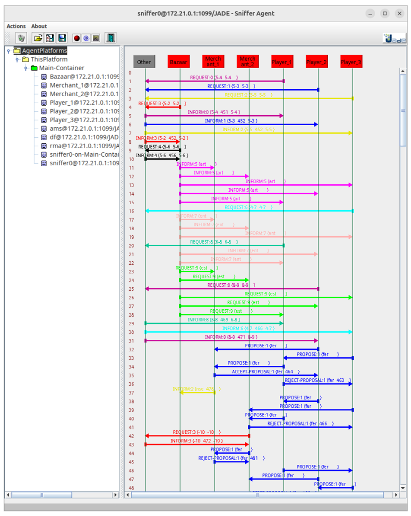


Figure 4: Sniffer Agent Output 1

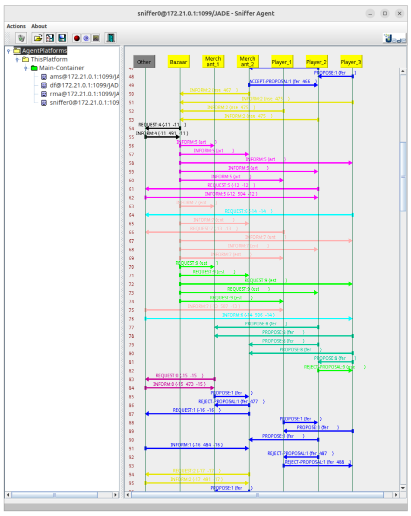


Figure 5: Sniffer Agent Output 2

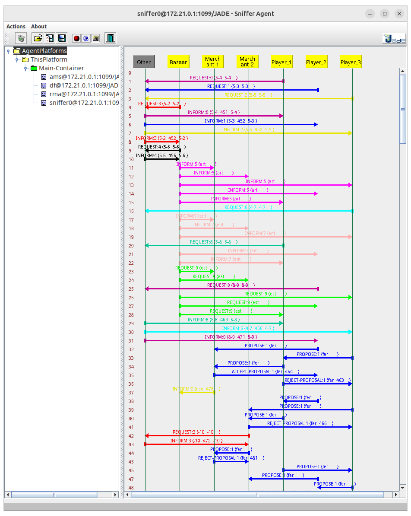


Figure 6: Sniffer Agent Output 3

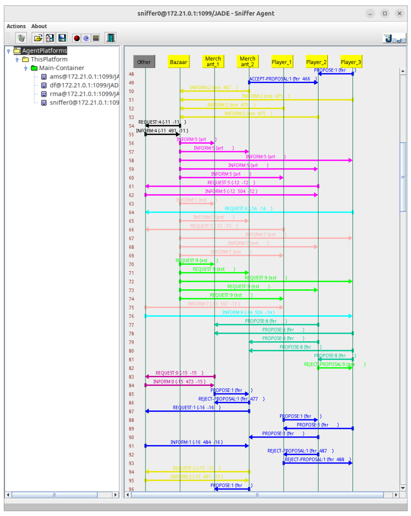


Figure 7: Sniffer Agent Output 4

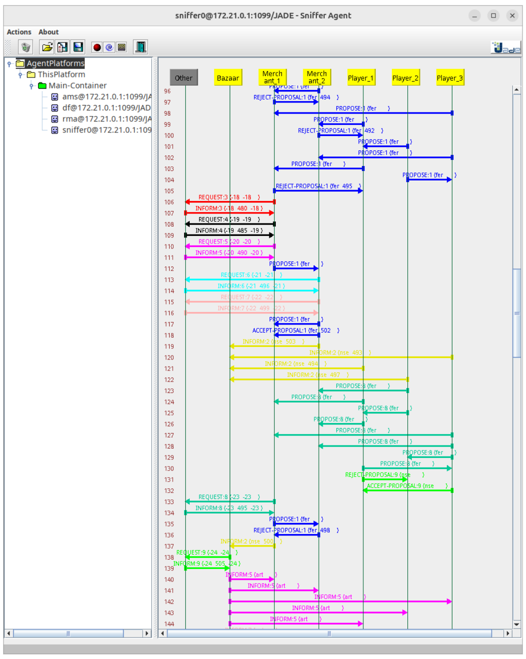


Figure 8: Sniffer Agent Output 5

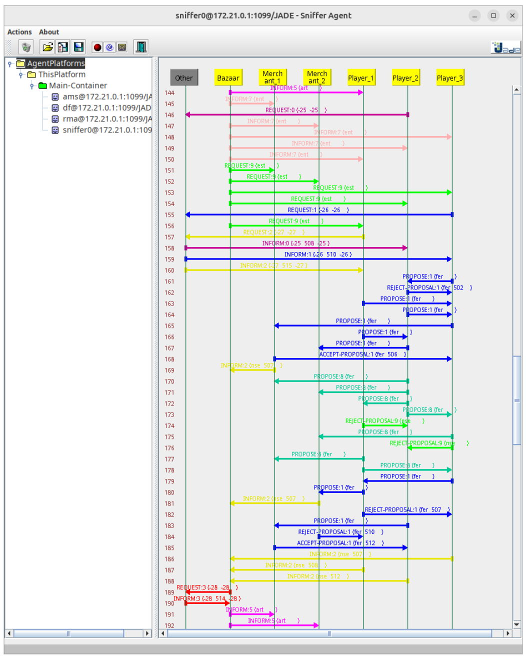


Figure 9: Sniffer Agent Output 6

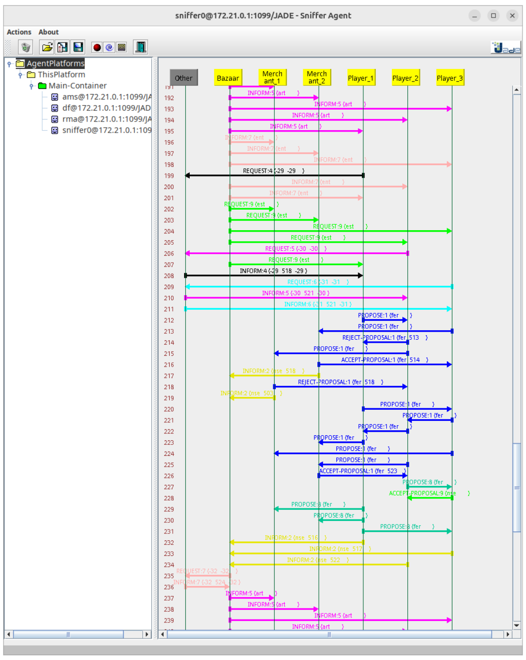


Figure 10: Sniffer Agent Output 7

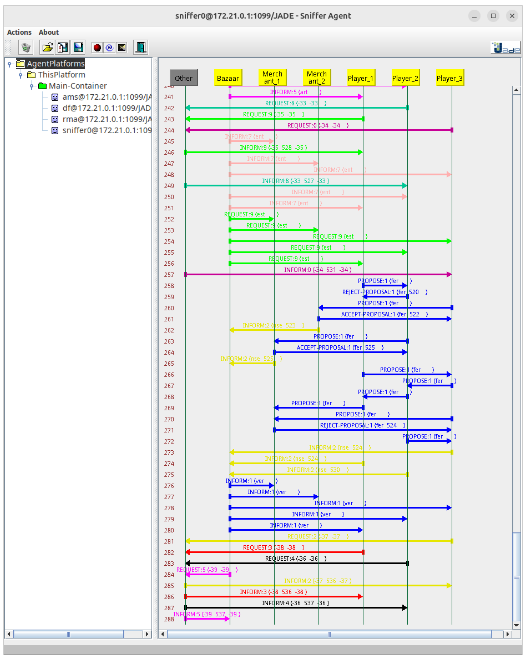


Figure 11: Sniffer Agent Output 8