HAN 2026 Tutorial

This document provides a quick tutorial to get you started in developing your agent for the HAN 2026 League.

The participants are challenged to build a negotiation strategy that uses the alternating offers protocol (AOP) augmented with text input/output to effectively negotiate with human subjects. Strategies should be compatible with standard AOP but may use multi-modal communication to influence or inform the human negotiator. For more information check the CFP and the Tutorial.

The Official HAN 2026 Template contains multiple examples of agent implementations for the HAN 2026 competition built with NegMAS and NegMAS-LLM. You can test your agent using the HANI interface.

Quick Start

1. **Install dependencies:** `uv sync` (details)
2. **Set up Ollama (for LLM agents):** `han2026 setup-ollama` (details) > Only needed if you plan to use LLM-based agents or want to try the LLM examples
3. **Rename your agent:** Change `mynegotiator.py` to `your_agent.py` and `MyNegotiator` to `YourAgent` (details)
4. **Implement your agent:** Edit your renamed file (details, examples)
5. **Test locally:** Run `han2026 run` for agent-vs-agent testing and `han2026 gui` for human-vs-agent testing (details)
6. **Submit:** Zip and upload to the competition site (details)

   [!NOTE] We **HIGHLY recommend** that you follow the whole process from installation to submission once you download the skeleton submitting the sample negotiator as your own submission to understand the whole process and save a lot of time later. The whole process should take no more than *5min* to try. If you face any issues in the submission you can email us here

Table of Contents

Project Structure

```
.
├── examples/               # Example negotiator implementations
│   ├── llm.py              # Pure LLM negotiator (standalone, no base negotiator)
│   ├── llm_adapter.py      # LLM-based adapter that wraps existing negotiators
│   ├── nollm_adapter.py    # Template-based adapter that wraps existing negotiators
│   └── nollm.py            # Non-LLM negotiators (BOA and simple SAO)
├── scenarios/              # Negotiation scenarios
│   ├── Grocery/
│   ├── Island/
│   └── Trade/
├── main.py                 # CLI application entry point
├── mynegotiator.py         # Your agent implementation (RENAME & EDIT THIS!)
├── pyproject.toml          # Project configuration
└── README.md
```

1. Registration and Download

Before you begin, register for the competition and download the skeleton:

1. **Register** for the HAN 2026 competition at https://anac.cs.brown.edu/register
2. **Download** the skeleton from https://anac.cs.brown.edu/files/han/y2026/han.zip
3. **Extract** the downloaded zip file to your preferred location

2. Installation

Using uv (Recommended)

First, install uv if you do not have it:

Linux/macOS:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Windows (PowerShell):

```
powershell -ExecutionPolicy ByPass -c `
  "irm https://astral.sh/uv/install.ps1 | iex"
```

Then install the project dependencies (all platforms):

```
uv sync
```

To update NegMAS to the latest version (all platforms):

```
uv sync --upgrade-package negmas
```

Using pip

Linux/macOS:

```
pip install -e .
```

Windows:

```
pip install -e .
```

Setting Up Ollama (for LLM Agents)

If you plan to use LLM-based agents or want to try the LLM examples, you need to install Ollama and pull the required model. Run:

```
han2026 setup-ollama
```

This command will:

1. Install Ollama (platform-specific)
2. Start the Ollama service if needed
3. Pull the required `qwen3:4b-instruct` model (~2-3 GB download)

   Note: If the automatic installation fails, see Manual Ollama Installation in the Troubleshooting section.

3. Getting Started: Rename Your Agent

Before you start developing, rename the agent module and class to match your submission name. This helps identify your agent in tournaments and is required for submission.

Naming conventions:

- **Module (file):** `snake_case` (e.g., `awesome.py`, `smart_negotiator.py`)
- **Class:** `TitleCase` (e.g., `AwesomeNegotiator`, `SmartNegotiator`)

VS Code

1. **Rename the file:**
   - Right-click `mynegotiator.py` in the Explorer, then Rename
   - Enter your new name (e.g., `awesome.py`)
2. **Rename the class:**
   - Open the renamed file
   - Click on `MyNegotiator` class name
   - Press `F2` (or right-click, then Rename Symbol)
   - Enter your new class name (e.g., `AwesomeNegotiator`)
3. **Update `main.py` (only 2 lines need changing):**
   - Open `main.py`
   - Line 25: Change `from mynegotiator import MyNegotiator` to `from awesome import AwesomeNegotiator`
   - Line 30: Change `MY_NEGOTIATOR = "mynegotiator.MyNegotiator"` to `MY_NEGOTIATOR = "awesome.AwesomeNegotiato`

   That's it! The `MY_NEGOTIATOR` variable is used throughout the application, so you only need to update it once.

PyCharm

1. **Rename the file:**
   - Right-click `mynegotiator.py` in Project view, then Refactor, then Rename (or `Shift+F6`)
   - Enter your new name (e.g., `awesome.py`)
   - Check "Search for references" and "Search in comments and strings"
2. **Rename the class:**
   - Open the renamed file
   - Right-click on `MyNegotiator`, then Refactor, then Rename (or `Shift+F6`)
   - Enter your new class name (e.g., `AwesomeNegotiator`)
   - PyCharm will update the import in `main.py` automatically
3. **Update the MY_NEGOTIATOR variable in `main.py`:**
   - Open `main.py`
   - Change `MY_NEGOTIATOR = "mynegotiator.MyNegotiator"` to `MY_NEGOTIATOR = "awesome.AwesomeNegotiator"`

This single variable update will propagate throughout the entire application.

Vim/Neovim (with LSP)

1. **Rename the file:**

   ```
   mv mynegotiator.py awesome.py
   ```

2. **Rename the class (using LSP rename):**
   - Open the file and place cursor on `MyNegotiator`
   - Use your LSP rename command (commonly `<leader>rn` or `:lua vim.lsp.buf.rename()`)
   - Enter the new name (e.g., `AwesomeNegotiator`)

3. **Update `main.py` (only 2 lines):**
   - Change `from mynegotiator import MyNegotiator` to `from awesome import AwesomeNegotiator`
   - Change `MY_NEGOTIATOR = "mynegotiator.MyNegotiator"` to `MY_NEGOTIATOR = "awesome.AwesomeNegotiator"`

Manual Renaming (Any Editor)

1. **Rename the file:**

   ```
   mv mynegotiator.py awesome.py
   ```

2. **Edit the renamed file:**
   - Change `class MyNegotiator` to `class AwesomeNegotiator` (or your chosen name)

3. **Edit `main.py` (only 2 lines):**
   - Line 25: Change `from mynegotiator import MyNegotiator` to `from awesome import AwesomeNegotiator`
   - Line 30: Change `MY_NEGOTIATOR = "mynegotiator.MyNegotiator"` to `MY_NEGOTIATOR = "awesome.AwesomeNegotiato`

     Note: The `MY_NEGOTIATOR` variable is used throughout `main.py` for default agent references, so updating it once updates all occurrences automatically.

4. **Verify the changes:**

   ```
   han2026 run
   ```

4. Implementing Your Agent

Your agent is implemented in your renamed module file. See the example agents below for different approaches to building agents.

Example Agents

The `examples/` folder contains example negotiator implementations that demonstrate different approaches to building agents:

HAN2026LLMNegotiator (`examples/llm.py`) A **pure LLM negotiator** that handles all negotiation decisions directly through an LLM. This is the simplest way to create an LLM-based agent - it extends `OllamaNegotiator` and lets the LLM make all decisions:

- **Standalone:** No base negotiator - the LLM handles everything
- **Customizable:** All prompts can be overridden via constructor parameters
- **Model:** Uses `qwen3:4b-instruct` (required for HAN 2026; only hyperparameters like temperature are configurable)

This is useful when you want the LLM to have full control over negotiation strategy and don't need a traditional negotiator as a fallback.

You can test it with:

```
han2026 run --opponent examples.llm.HAN2026LLMNegotiator
```

BoulwareBasedLLMNegotiator (`examples/llm_adapter.py`) An **LLM-based adapter** that wraps existing negotiators to add natural language communication capabilities. This example demonstrates how to use `LLMMetaNegotiator` to enhance any traditional negotiator with LLM-powered decision making:

- **Base Negotiator:** Uses `BoulwareTBNegotiator` as the underlying strategy (a time-based tough negotiator)
- **LLM Enhancement:** The LLM learns from and adapts the base negotiator's behavior
- **Hybrid Approach:** Combines the reliability of traditional strategies with LLM flexibility

This is useful when you want to leverage proven negotiation strategies while adding natural language reasoning.

You can test it with:

```
han2026 run --opponent examples.llm_adapter.BoulwareBasedLLMNegotiator
```

TemplateBasedAdapterNegotiator (`examples/nollm_adapter.py`) A **template-based adapter** that wraps existing negotiators to add context-aware natural language messages using predefined templates. This example demonstrates how to use `SAOMetaNegotiator` to enhance any traditional negotiator with natural language communication **without** requiring an LLM:

- **Base Negotiator:** Uses `BoulwareTBNegotiator` as the underlying strategy (like the LLM adapter)
- **Template Enhancement:** Generates messages from predefined templates that reference specific offer details
- **Context-Aware:** Messages mention actual issue values (e.g., "price of 20 is too high, I'm proposing 30 instead")
- **No LLM Required:** Fast and deterministic - great for testing and as a baseline

This is useful when you want natural language communication without the overhead and variability of LLM calls.

You can test it with:

```
han2026 run --opponent examples.nollm_adapter.TemplateBasedAdapterNegotiator

# Or test it against itself to see template-based communication
han2026 run --agent examples.nollm_adapter.TemplateBasedAdapterNegotiator \
            --opponent examples.nollm_adapter.TemplateBasedAdapterNegotiator
```

Non-LLM Negotiators (`examples/nollm.py`) This file contains two traditional (non-LLM) negotiators that don't require an LLM to run:

BOANeg - A modular agent using the **BOA (Bidding, Opponent modeling, Acceptance)** architecture:

- **Bidding Strategy:** Uses `TimeBasedOfferingPolicy` - makes concessions based on remaining time
- **Opponent Model:** Uses `GSmithFrequencyModel` - learns opponent preferences from their offers
- **Acceptance Strategy:** Uses `ACNext` - accepts if the offer is better than what we would offer next

```
han2026 run --opponent examples.nollm.BOANeg
```

SimpleNeg - A minimal agent implemented in a single function, demonstrating the basics:

- **Acceptance:** Accepts any offer with utility > 0.8
- **Offering:** Always proposes the best outcome for itself
- **Natural Text:** Includes simple response messages ("Thank you for this great offer", etc.)

This is a good starting point to understand the negotiation API before moving to more complex architectures.

```
han2026 run --opponent examples.nollm.SimpleNeg
```

You can see all available components from negmas using:

```
# Linux/macOS
python -c "from negmas.registry import component_registry as CR; \
  print(CR.keys());"
```

```
# Windows (single line)
# python -c "from negmas.registry import component_registry as CR; print(CR.keys());"
```

> [!NOTE] You can base your agent on any supported NegMAS negotiator (For some examples check this list. Other agents are available through negmas-negobog and negmas-geniusweb-bridge). You can explore available negotiators, their behavior, etc in the NegMAS GUI.

5. Customizing Your Agent

Modifying Prompts

Your agent uses several prompts defined as module-level variables at the top of your agent file. You can customize these to change how the LLM thinks about and approaches negotiation.

Available Prompt Variables In your agent file (e.g., `mynegotiator.py`), you'll find these module-level prompt variables:

- **`SYSTEM_PROMPT`** - Sets the overall behavior, personality, and role of the LLM negotiator
- **`PREFERENCES_PROMPT`** - Sent when preferences are first set at the start of negotiation
- **`PREFERENCES_CHANGED_PROMPT`** - Sent when preferences change during negotiation (rare)
- **`NEGOTIATION_START_PROMPT`** - Sent when negotiation begins, explains the response format
- **`ROUND_PROMPT`** - Sent each negotiation round with current state information

Each prompt can include **tags** (see Using Tags in Prompts) that get dynamically replaced with negotiation context.

Example: Creating an Aggressive Negotiator To make your agent more aggressive and less willing to compromise:

```
SYSTEM_PROMPT = """
You are a TOUGH negotiator who drives HARD bargains. Your primary goal
is to maximize your own utility, and you should be very reluctant to
make concessions. Only accept offers that give you excellent utility.

Be assertive in your text messages to the opponent. Make it clear you
expect favorable terms. Start with your best possible outcomes and
concede slowly and reluctantly only when time is running out.

Always respond in the exact JSON format requested.
"""
```

Example: Creating a Cooperative Negotiator To make your agent more cooperative and focused on mutual benefit:

```
SYSTEM_PROMPT = """
You are a COOPERATIVE negotiator focused on finding win-win solutions.
While you aim for good utility, you also value reaching agreements that
benefit both parties. Analyze the opponent's preferences when available
and look for outcomes that provide high joint utility.

Be friendly and collaborative in your text messages. Explain your
reasoning and show willingness to compromise when it makes sense.

Always respond in the exact JSON format requested.
"""
```

Example: Adding Opponent Modeling to Negotiation Start You can enhance the `NEGOTIATION_START_PROMPT` to encourage the LLM to model the opponent:

~~~
NEGOTIATION_START_PROMPT = """
# Negotiation Started

The negotiation has now started. For each round, you should:

1. **Model your opponent**: Track which outcomes they offer and accept
   to infer their preferences. Use {{history:text(k=5)}} to see recent moves.
2. **Analyze offers**: When you receive an offer, check its utility for you
   using {{utility:text(outcome={{opponent-last-offer}})}}
3. **Make strategic decisions**: Choose to ACCEPT, REJECT (with counter-offer), or END
4. **Communicate effectively**: Provide persuasive text that advances your position

Respond in this JSON format for each decision:

```json
{
    "response_type": "accept" | "reject" | "end" | "wait",
    "outcome": [value1, value2, ...] | null,
    "text": "optional persuasive message to send to your opponent",
    "reasoning": "brief explanation of your decision (not sent to opponent)"
}
~~~

Using Tags in Prompts

Tags allow you to dynamically insert negotiation context into prompts. They are written as `{{tag-name}}` or `{{tag-name:format(param=value)}}`.

Common Tags and Examples To see all available tags, run:

```
han2026 tags
```

Here are some frequently used tags:

Outcome Space Tags:

- `{{outcome-space:json}}` - Get the full outcome space in JSON format
- `{{outcome-space:text}}` - Get human-readable outcome space description
- `{{n-outcomes}}` - Number of possible outcomes

Utility Tags:

- `{{utility:text(outcome={{opponent-last-offer}})}}` - Compute utility of opponent's last offer
- `{{utility-function:text}}` - Your utility function description
- `{{reserved-value}}` - Your walk-away utility value
- `{{opponent-utility-function:text}}` - Opponent's utility (if available)

History Tags:

- `{{history:text}}` - Full negotiation history
- `{{history:text(k=5)}}` - Last 5 negotiation steps
- `{{opponent-last-offer}}` - Opponent's most recent offer
- `{{my-last-offer}}` - Your most recent offer

Context Tags:

- `{{nmi:text}}` - Negotiation mechanism information (time limit, steps, etc.)
- `{{step}}` - Current negotiation step number
- `{{relative-time}}` - How much time has elapsed (0.0 to 1.0)
- `{{running}}` - Whether negotiation is still active

Example: Rich Round Prompt with Multiple Tags

```
ROUND_PROMPT = """
# Round {{step}}

**Progress**: Step {{step}} | Time: {{relative-time:.1%}} | Running: {{running}}

## Current Situation
{{history:text(k=3)}}

## Opponent's Last Offer
{{opponent-last-offer}}

**Utility for you**: {{utility:text(outcome={{opponent-last-offer}})}}

## Your Last Offer
{{my-last-offer}}

## Analysis
- Your reserved value (walk-away point): {{reserved-value}}
- Total possible outcomes: {{n-outcomes}}

What is your decision? Remember to maximize your utility while considering
the time pressure and opponent's apparent preferences.

Respond with JSON.
"""
```

Customizing Example Agents

Adapter Example (`examples/llm_adapter.py`) The adapter wraps a traditional negotiator with LLM capabilities. You can customize:

1. Change the Base Negotiator:

Replace `BoulwareTBNegotiator` (tough negotiator) with a different strategy:

```python
from negmas.sao import ConcederTBNegotiator  # More willing to concede

class CooperativeLLMNegotiator(LLMMetaNegotiator):
    def __init__(self, **kwargs):
        super().__init__(
            negotiator=ConcederTBNegotiator(),  # Changed from BoulwareTBNegotiator
            system_prompt=SYSTEM_PROMPT,
            **kwargs
        )
```

2. Modify the System Prompt:

The `SYSTEM_PROMPT` in `llm_adapter.py` controls how the LLM generates text. Customize it to change the communication style:

```
SYSTEM_PROMPT = """
You are assisting in automated negotiation by generating natural language
messages. Be BRIEF and DIRECT - keep messages under 20 words when possible.

Available information:

- {{outcome-space:text}}
- {{history:text(k=3)}}
```

```
Generate concise, persuasive text based on the negotiation context.
"""
```

3. Adjust LLM Parameters:

```
class MyCustomAdapter(LLMMetaNegotiator):
    def __init__(self, **kwargs):
        super().__init__(
            negotiator=BoulwareTBNegotiator(),
            system_prompt=SYSTEM_PROMPT,
            temperature=0.9,            # More creative responses
            max_tokens=512,             # Shorter responses
            **kwargs
        )
```

> **Important:** For the HAN 2026 league, changing the LLM model is **not allowed**. All agents must use the default model (qwen3:4b-instruct). You may only customize prompts, temperature, and other parameters.

Non-LLM Examples (examples/nollm.py) **BOANeg** uses modular components you can swap:

```
from negmas.sao import AspirationNegotiator, GaussianAcceptanceStrategy

# More flexible bidding strategy
bidding_strategy = AspirationNegotiator(max_aspiration=0.9, aspiration_type='linear')

# Different acceptance strategy
acceptance_strategy = GaussianAcceptanceStrategy()

class MyBOANeg(SAONegotiator):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.add_capabilities(
            offering=bidding_strategy,
            acceptance=acceptance_strategy,
            opponent_model=GSmithFrequencyModel()
        )
```

SimpleNeg can be modified to be more strategic:

```
def respond(state, offer):
    my_utility = self.ufun(offer) if offer else 0.0

    # More sophisticated acceptance threshold
    time_pressure = state.relative_time
    threshold = 0.7 + (0.2 * time_pressure)  # Accept lower offers as time runs out

    if my_utility >= threshold:
        return SAOResponse(ResponseType.ACCEPT_OFFER, offer)

    # Find best counter-offer
    outcomes = self.nmi.outcome_space.enumerate()
    best = max(outcomes, key=lambda o: self.ufun(o))

    return SAOResponse(
        ResponseType.REJECT_OFFER,
        best,
        f"I propose this alternative (utility for me: {self.ufun(best):.2f})"
    )
```

For detailed tag documentation, use:

```
han2026 tags <tag-name>
```

6. Usage from the command line

> **Note:** All commands below work on Linux, macOS, and Windows. On Windows, use Command Prompt, PowerShell, or Windows Terminal.

Running a single negotiation

To run a single negotiation:

```
han2026 run
```

This will run your agent against a random opponent on a random scenario and report:

- **Advantage:** utility - reserved-value
- **Deception:** How well you confuse your opponent's model of you
- **Score:** The final HAN 2026 score

Run command options

Option	Description
`--scenario TEXT`	Scenario name to use (default: random from scenarios folder)
`--generate-scenario`	Generate a random scenario instead of loading one
`--rational-fraction FLOAT`	Fraction of rational outcomes in generated scenarios (0.0-1.0, default: 1.0)
`--opponent TEXT`	Opponent class to negotiate against (default: random)
`--verbose`	Show full negotiation trace with utilities
`--plot / --no-plot`	Plot the negotiation trace (default: plot)

Examples:

```
# Run with a specific scenario
han2026 run --scenario Grocery

# Run with a generated scenario
han2026 run --generate-scenario

# Run against a specific opponent (Any negmas/genius agent can be used this way)
han2026 run --opponent negmas.sao.BoulwareTBNegotiator


# Run against a specific opponent (Any example can be used this way)
han2026 run --opponent examples.boa.BOANeg

# Run with verbose output and no plot
han2026 run --verbose --no-plot
```

Running a tournament

To run a complete tournament with all default competitors:

```
han2026 tournament
```

This will run your agent against multiple opponents across all scenarios and report the final scores.

Tournament command options

Option	Description
`--name TEXT`	Tournament name (default: auto-generated)
`--competitor TEXT`	Competitor classes to include (can be repeated, default: built-in competitors)
`--scenario TEXT`	Scenario names or paths (can be repeated, use 'all' for all scenarios)
`--generate-scenarios N`	Generate N random scenarios
`--rational-fraction FLOAT`	Fraction of rational outcomes in generated scenarios (0.0-1.0)
`--verbosity INT`	Verbosity level 0-5 (default: 0 = silent)
`--parallel`	Run tournament in parallel mode using all cores

Examples:

```
# Run tournament on specific scenarios
han2026 tournament --scenario Grocery --scenario Island

# Run tournament with generated scenarios
han2026 tournament --generate-scenarios 10

# Run tournament in parallel with verbose output
han2026 tournament --parallel --verbosity 1

# Run tournament with custom competitors (Linux/macOS)
han2026 tournament \
  --competitor mynegotiator.MyNegotiator \
  --competitor examples.boa.BOANeg

# Run tournament with custom competitors (Windows CMD)
# han2026 tournament ^
#   --competitor mynegotiator.MyNegotiator ^
#   --competitor examples.boa.BOANeg
```

Testing with Human Negotiators (HANI GUI)

The HANI (Human-Agent Negotiation Interface) allows you to test your agent in real human-agent negotiations through an interactive GUI. This is valuable for:

- **Understanding agent behavior:** See how your agent communicates and negotiates in real-time
- **Identifying issues:** Spot problems with decision-making or message generation
- **Evaluating user experience:** Test if your agent's messages are clear and persuasive
- **Practice:** Experience negotiating against your own agent to find weaknesses

Quick Start To launch the GUI with your agent in guest mode (no authentication required):

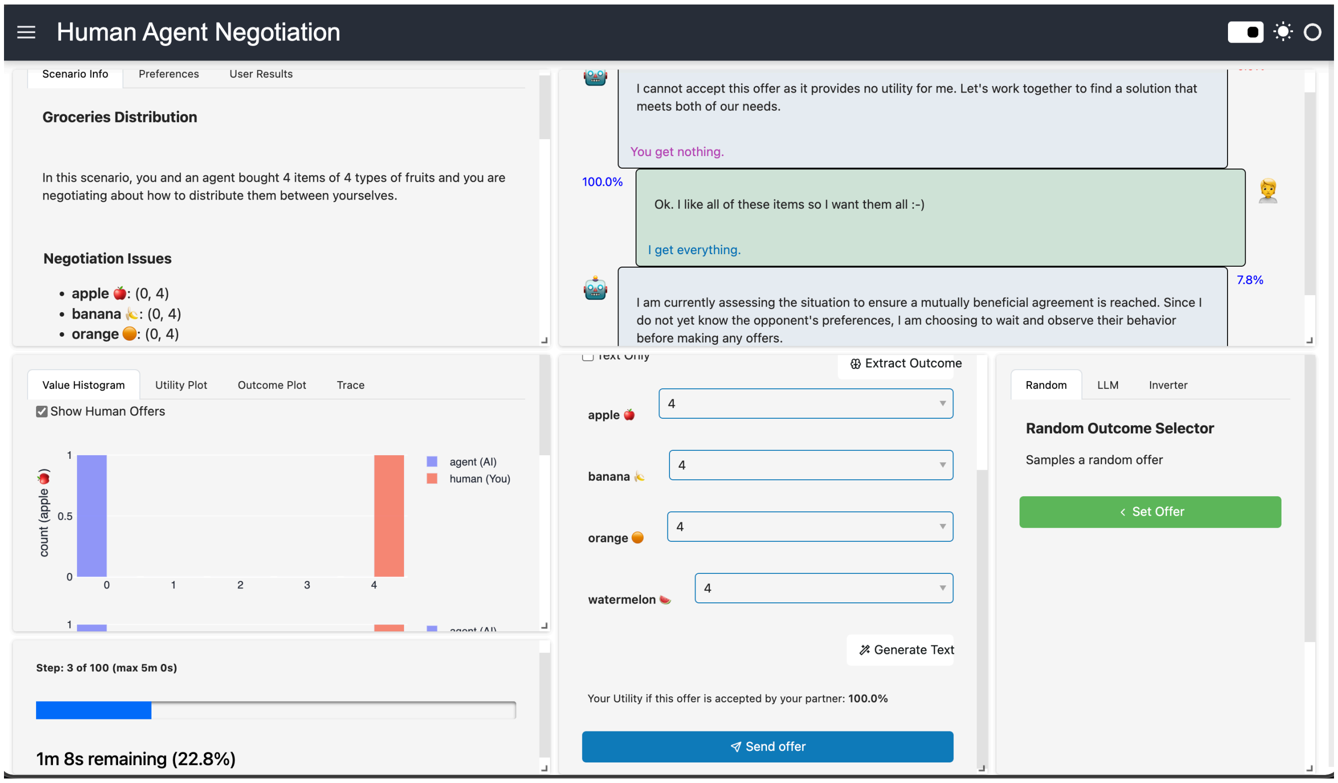
```
han2026 gui
```

This will start the HANI interface in guest/playground mode with your default agent (`mynegotiator.MyNegotiator`). The GUI will open in your browser automatically at `http://localhost:5008`. You can then negotiate against your agent as a human player.

GUI Components The HANI interface is divided into several panels:

Left Panel - Scenario Information:

- **Scenario Info tab:** Displays the scenario name (e.g., "Groceries Distribution") and description explaining what you're negotiating about



Groceries Distribution

In this scenario, you and an agent bought 4 items of 4 types of fruits and you are negotiating about how to distribute them between yourselves.

Negotiation Issues

- **apple** 🍎: (0, 4)
- **banana** 🍌: (0, 4)
- **orange** 🍊: (0, 4)

☑ Show Human Offers

I cannot accept this offer as it provides no utility for me. Let's work together to find a solution that meets both of our needs.

You get nothing.

100.0%

Ok. I like all of these items so I want them all :-)

I get everything.

7.8%

I am currently assessing the situation to ensure a mutually beneficial agreement is reached. Since I do not yet know the opponent's preferences, I am choosing to wait and observe their behavior before making any offers.

apple 🍎	4
banana 🍌	4
orange 🍊	4
watermelon 🍉	4

Your Utility if this offer is accepted by your partner: **100.0%**

Random Outcome Selector

Samples a random offer

Step: 3 of 100 (max 5m 0s)

1m 8s remaining (22.8%)

Figure 1: HANI GUI Interface

- **Negotiation Issues:** Lists all issues being negotiated with their value ranges (e.g., apple: 0-4, banana: 0-4, orange: 0-4)
- **Preferences tab:** View your utility function and preferences
- **User Results tab:** See your negotiation history and performance

Left Panel - Visualization (bottom):

- **Value Histogram:** Bar chart comparing offers from agent (AI) vs human (You) across different values
- **Utility Plot:** Track utility values over the negotiation
- **Outcome Plot:** Visualize the outcome space
- **Trace:** View the full negotiation trace
- **Progress bar:** Shows current step (e.g., "Step 3 of 100") and remaining time (e.g., "2m41s remaining")

Center Panel - Chat and Offers:

- **Message history:** Shows the conversation between you and the agent, with utility percentages displayed next to each offer
- **Agent messages:** Displayed with a robot icon, showing both the text message and the formal offer (e.g., "You get nothing.")
- **Your messages:** Displayed with a human icon on the right side
- **Utility indicators:** Green percentages show how good each offer is for the respective party (e.g., "100.0%", "7.8%")

Right Panel - Offer Controls:

- **Issue dropdowns:** Select values for each issue in your offer (apple, banana, orange, watermelon, etc.)
- **Extract Outcome button:** Parse an outcome from text using AI
- **Generate Text button:** Generate a message to accompany your offer
- **Your Utility display:** Shows what utility you'll get if your current offer is accepted
- **Send offer button:** Submit your offer to the agent
- **Offer helpers (top right):**

  - **Random:** Generate a random offer
  - **LLM:** Use an LLM to suggest an offer
  - **Inverter:** Invert the current offer values

Testing Different Agents To test a specific agent (including example agents):

```
# Test your main negotiator (default)
han2026 gui --agents file:mynegotiator.MyNegotiator

# Test your renamed agent
han2026 gui --agents file:awesome.AwesomeNegotiator

# Test example agents using file: prefix for local files
han2026 gui --agents file:examples/nollm_adapter.TemplateBasedAdapterNegotiator
han2026 gui --agents file:examples/nollm.SimpleNeg
han2026 gui --agents file:examples/nollm.BOANeg
han2026 gui --agents file:examples/llm_adapter.BoulwareBasedLLMNegotiator
```

> **Important:** Always use the `file:` prefix followed by the file path (without `.py` extension) and class name separated by a dot. For example: - `file:mynegotiator.MyNegotiator` – loads `MyNegotiator` class from `mynegotiator.py` - `file:examples/nollm.SimpleNeg` – loads `SimpleNeg` class from `examples/nollm.py` - `file:awesome.AwesomeNegotiator` – loads `AwesomeNegotiator` class from `awesome.py`

Using HANI Directly You can also use HANI commands directly for more control:

```
# Guest mode - simple playground without authentication (recommended for testing)
hani-guest --agents file:mynegotiator.MyNegotiator

# Development mode - allows editing code while running
hani --dev --agents file:mynegotiator.MyNegotiator

# Test different example agents
hani-guest --agents file:examples/nollm_adapter.TemplateBasedAdapterNegotiator
hani-guest --agents file:examples/nollm.SimpleNeg
hani-guest --agents file:examples/nollm.BOANeg

# Multiple agent types (Linux/macOS)
hani-guest --agents \
  file:mynegotiator.MyNegotiator,\
file:examples/nollm.BOANeg,\
file:examples/nollm_adapter.TemplateBasedAdapterNegotiator

# Multiple agent types (Windows CMD) - use ^ for line continuation
# hani-guest --agents ^
#   file:mynegotiator.MyNegotiator,^
#   file:examples/nollm.BOANeg,^
#   file:examples/nollm_adapter.TemplateBasedAdapterNegotiator
```

> **Tip:** The `file:` prefix tells HANI to load the agent from a local Python file in your project directory. Use forward slashes (/) for subdirectories, even on Windows.

GUI Features The HANI interface provides:

- **Interactive negotiation:** Make offers, accept/reject proposals, and send messages
- **Real-time feedback:** See your agent's responses and reasoning
- **Scenario selection:** Choose from available scenarios or create custom ones
- **Utility visualization:** Track utilities and negotiation progress

- **Message history:** Review the full conversation and offers exchanged

Tips for Testing

1. **Start simple:** Test with straightforward scenarios like `Grocery` or `Trade` first
2. **Observe patterns:** Watch how your agent responds to different offer sequences
3. **Test edge cases:** Try extreme offers, quick acceptance, or delayed responses
4. **Check messages:** Ensure your agent's text is appropriate and persuasive
5. **Verify decisions:** Confirm your agent makes rational acceptance/rejection decisions

For more information about HANI, visit the HANI repository.

Viewing Development and Submission Info

To view development workflows and submission instructions in your terminal:

```
han2026 info
```

Viewing Available Prompt Tags

To see all available tags for customizing LLM prompts:

```
han2026 tags
```

To get detailed documentation for a specific tag:

```
han2026 tags utility
han2026 tags outcome-space
```

Tags can be used in custom prompts to dynamically insert negotiation context such as outcome space, utilities, negotiation history, and opponent information.

7. Development Workflows

VS Code

1. Open the project folder in VS Code
2. Install the recommended Python extension
3. Select the Python interpreter from `.venv`:
   - Press `Ctrl+Shift+P` (Windows/Linux) or `Cmd+Shift+P` (macOS)
   - Type "Python: Select Interpreter"
   - Choose the interpreter from `.venv/bin/python` (Linux/macOS) or `.venv\Scripts\python.exe` (Windows)
4. Edit your agent file (renamed from `mynegotiator.py`) to implement your agent
5. Run tests with `pytest` in the integrated terminal
6. Run your agent with `han2026 run`

Vim/Neovim

1. Ensure you have a Python LSP configured (e.g., pyright, pylsp)
2. Activate the virtual environment or use `uv run` prefix for commands
3. Edit your agent file (renamed from `mynegotiator.py`) to implement your agent
4. Run commands from the terminal:

   ```
   han2026 run
   pytest
   ```

PyCharm / Other IDEs

1. Open the project folder
2. Configure the Python interpreter to use `.venv`:
   - Go to Settings / Preferences > Project > Python Interpreter
   - Add the interpreter from `.venv/bin/python` (Linux / macOS) or `.venv\Scripts\python.exe` (Windows)
3. Edit your agent file (renamed from `mynegotiator.py`) to implement your agent
4. Use the built-in terminal to run:

```
han2026 run
pytest
```

8. Submission

To submit your agent to the HAN 2026 competition:

1. Ensure you renamed your agent and followed the instruction for development and have a runnable agent. To test that you have a runnable agent, use:

```
han2026 run
```

and

```
han2026 tournament
```

2. Test your agent locally using `han2026 run` and `han2026 tournament`
3. Add any extra dependencies you need for your agent in requirements.txt. Try to always use the most recent version of each library to maximize the re-usability of your agent in the future.
4. Zip the code for your agent, excluding: `.*` (dotfiles / folders), `*.sh` / `*.bat` scripts, `__pycache__/`, `examples/`, `report/`, `scenarios/`, and `tests/`:

   Using the provided scripts (recommended):

   Linux / macOS:

```
./make_submission.sh
```

   Windows:

```
make_submission.bat
```

   Or manually:

   Linux / macOS:

```
zip -r submission.zip . -x ".*" -x "*.sh" -x "*.bat" \
  -x "__pycache__/*" -x "examples/*" -x "report/*" \
  -x "scenarios/*" -x "tests/*"
```

   Windows (PowerShell):

```
Get-ChildItem -Exclude .*,*.sh,*.bat,__pycache__,examples,report,scenarios,tests |
  Compress-Archive -DestinationPath submission.zip -Force
```

   Windows (Command Prompt with tar):

```
tar -a -cf submission.zip --exclude=".*" --exclude="*.sh" ^
  --exclude="*.bat" --exclude=__pycache__ --exclude=examples ^
  --exclude=report --exclude=scenarios --exclude=tests .
```

5. Submit your agent following the competition guidelines at HAN 2026 Competition Page

   5.1. [First time] Register for the competition here

5.2. Login for the submission site here

5.3. Go to your home page "Your Home" and choose "Submissions"

5.4. Under "HAN", click "New Agent" and fill the form. We will assume that your main agent class is called `AwesomeNegotiator` implemented in a file called `awesome.py` at the root of this folder:

- Agent Name: Awesome Negotiator
- Agent Alias: Awesome Negotiator
- Agent Module: awesome
- Agent Class: AwesomeNegotiator
- Dependencies: <any dependencies you had to `pip install` or `uv add` yourself in a semicolon-separated list, no spaces>
- Code: submission.zip
- Requirements File: requirements.txt
- Report: . This is only required for the final submission but it is safe to submit drafts of it at any time. No one will read them before the final submission deadline.

  [!NOTE] If you need to use data files (e.g. trained models, any files that your agent READs in real time for successful operation), make sure they are included in the zipped submission file and read this FAQ entry: Accessing Data Files. This is written for the SCML league but the process is EXACTLY the same.

You can submit your agent as many times as you want until the competition deadline. Please submit early and frequently. When you submit your agent, we run tests on it and you will get feedback if it is failing in the competition. Moreover, we keep a running leaderboard here which allows you to judge different improvements you make against your real opponents in the competition.

9. Troubleshooting

HANI GUI Issues

If the `hani-guest` or `hani` commands are not found:

```
# Install HANI from git repository
uv pip install 'hani @ git+https://github.com/autoneg/hani.git@main'

# Or with pip
pip install 'hani @ git+https://github.com/autoneg/hani.git@main'

# Verify installation
pip show hani
```

If you get errors when running `han2026 gui`:

Try running HANI directly to see more detailed error messages:

```
# Using guest mode (recommended)
hani-guest --agents mynegotiator.MyNegotiator

# Or using dev mode
hani --dev --agents mynegotiator.MyNegotiator
```

Common issues:

1. **"Cannot choose from an empty sequence" error:**

   - This occurs when HANI's image assets are missing
   - **Workaround:** Use `han2026 gui --use-dev` or run `hani --dev` directly
   - **Fix:** Reinstall HANI:

```
uv pip install --force-reinstall \
  'hani @ git+https://github.com/autoneg/hani.git@main'
```

- **Alternative:** Clone the HANI repository and install from source

2. **Port already in use:**
   - Another instance of HANI might be running
   - Close it or use `Ctrl+C` to stop the server
3. **Browser doesn't open:**
   - Manually navigate to `http://localhost:5008`
4. **Agent import errors:**
   - Make sure your agent module is importable
   - Check for syntax errors, missing dependencies, etc.
   - Verify your agent works with: `han2026 run`

Manual Ollama Installation

If `han2026 setup-ollama` fails or you prefer to install manually, follow these steps:

1. Install Ollama:

- **Linux:**

  ```
  curl -fsSL https://ollama.com/install.sh | sh
  ```

- **macOS:** Download and install from https://ollama.com/download, or use Homebrew:

  ```
  brew install ollama
  ```

- **Windows:** Download and install from https://ollama.com/download

2. Start the Ollama service:

```
ollama serve
```

> **Note:** On macOS and Windows, Ollama typically runs automatically after installation. On Linux, you may need to start the service manually.

3. Pull the required model:

The HAN 2026 competition requires the `qwen3:4b-instruct` model. Pull it before running LLM-based agents:

```
ollama pull qwen3:4b-instruct
```

This download is approximately 2-3 GB. You can verify the model is installed with:

```
ollama list
```

> **Important:** The model does not auto-download when running agents. You must pull it manually before testing LLM-based negotiators.

Common Ollama issues:

1. **"Connection refused" errors:**
   - The Ollama service may not be running
   - Start it with: `ollama serve`
2. **Model not found:**
   - Ensure you pulled the correct model: `ollama pull qwen3:4b-instruct`
   - Check installed models: `ollama list`
3. **Slow responses:**
   - The model runs locally and performance depends on your hardware
   - First request may be slow as the model loads into memory