ANL 2026 League Tutorial

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

The ANL 2026 competition challenges you to design and build a negotiation agent for bilateral negotiation that tries to mislead its opponent. The agent is rewarded for the agreement made in the negotiation, as well as for how well it is able to deceive its opponent. The challenge is to mislead the opponent while maximizing reward. For more details, see the CFP and the Tutorial.

A sample agent skeleton built with NegMAS can be downloaded here.

Quick Start

1. **Register & Download:** Register at ANL 2026 and download the skeleton (details)
2. **Install dependencies:** `uv sync` (details)
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:** `anl2026 run` and `anl2026 tournament` (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
```

```
│   ├── boa.py              # BOA (Bidding, Opponent modeling, Acceptance) agent
│   ├── map.py              # MAP (Maximum Aspiration Policy) agent
│   └── simple.py           # Simple negotiator example
├── scenarios/              # Negotiation scenarios
│   ├── Amsterdam/
│   ├── Camera/
│   ├── Car/
│   ├── Grocery/
│   ├── ISBTAcquisition/
│   ├── Laptop/
│   └── NiceOrDie/
├── main.py                 # CLI application entry point
├── mynegotiator.py         # Your agent implementation (RENAME & EDIT THIS!)
├── pyproject.toml          # Project configuration
└── README.md
```

0. Registration and Download

Before you can participate in the ANL 2026 competition, you need to register and download the skeleton code:

1. **Register for the competition** at https://anac.cs.brown.edu/register
   - Fill in your information
   - Choose "ANL 2026" as the league you want to participate in
2. **Download the skeleton code** from https://anac.cs.brown.edu/files/anl/y2026/anl2026.zip
   - Extract the zip file to your preferred location
   - This skeleton contains everything you need to get started

   [!TIP] You can also clone/download this repository directly from GitHub if you prefer using version control from the start.

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

Then install the project dependencies:

```
uv sync
```

To update NegMAS to the latest version:

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

Switching Between PyPI and GitHub Versions of NegMAS By default, the project uses the stable NegMAS release from PyPI. If you need the latest development version (e.g., for bug fixes not yet released), you can switch to the GitHub version:

1. **Edit `pyproject.toml`:**

```
dependencies = [
    # "negmas>=0.15.4",  # Comment out this line
    "negmas @ git+https://github.com/yasserfarouk/negmas.git@main",  # Uncomment this line
```

```
    "typer>=0.15.0",
]
```

2. **Sync dependencies:**

   ```
   uv sync
   ```

To switch back to the PyPI version, reverse the comments and run `uv sync` again.

> [!NOTE] For submissions, we recommend using the stable PyPI version unless instructed otherwise by the competition organizers.

Using pip

```
pip install -e .
```

2. 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 -> Rename
   - Enter your new name (e.g., `awsome.py`)
2. **Rename the class:**
   - Open the renamed file
   - Click on `MyNegotiator` class name
   - Press `F2` (or right-click -> Rename Symbol)
   - Enter your new class name (e.g., `AwesomeNegotiator`)
3. **Update references in `main.py`:**
   - Open `main.py`
   - Replace all occurrences of `mynegotiator` with your module name (e.g., `awesome`)
   - Replace all occurrences of `MyNegotiator` with your class name (e.g., `AwesomeNegotiator`)

PyCharm

1. **Rename the file:**
   - Right-click `mynegotiator.py` in Project view -> Refactor -> 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` -> Refactor -> Rename (or `Shift+F6`)
   - Enter your new class name (e.g., `AwesomeNegotiator`)
   - PyCharm will update all references automatically
3. **Verify imports in `main.py`** are updated correctly

Vim/Neovim (with LSP)

1. **Rename the file** (Linux/macOS/Git Bash):

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

   Or on Windows (Command Prompt/PowerShell):

   ```
   ren 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 references in `main.py`:**
   - Replace all occurrences of `mynegotiator` with your module name (e.g., `awesome`)
   - Replace all occurrences of `MyNegotiator` with your class name (e.g., `AwesomeNegotiator`)

Manual Renaming (Any Editor)

1. **Rename the file** (Linux/macOS/Git Bash):

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

   Or on Windows (Command Prompt/PowerShell):

   ```
   ren mynegotiator.py awesome.py
   ```

2. **Edit the renamed file:**
   - Change `class MyNegotiator` to `class AwesomeNegotiator` (or your chosen name)
3. **Edit `main.py`:**
   - Replace all occurrences of `mynegotiator` with your module name (e.g., `awesome`)
   - Replace all occurrences of `MyNegotiator` with your class name (e.g., `AwesomeNegotiator`)
4. **Verify the changes:**

   ```
   anl2026 run
   ```

3. 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 three example negotiator implementations that demonstrate different approaches to building agents:

BOANeg (`examples/boa.py`) A modular agent using the **BOA (Bidding, Opponent modeling, Acceptance)** architecture. This architecture separates the negotiation strategy into three independent components:

- **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

You can test it with:

```
anl2026 run --opponent examples.boa.BOANeg
```

You can see all available components from negmas using:

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

MAPNeg (`examples/map.py`) A **MAP (Maximum Aspiration Policy)** agent that also uses modular components but with a different base architecture allowing for multiple opponent models at once and other arbitrary components:

- Uses the same time-based offering and acceptance strategies as BOANeg
- Supports multiple opponent models (configured with `GSmithFrequencyModel`)
- Processes acceptance before offering (`acceptance_first=True`)

```
anl2026 run --opponent examples.map.MAPNeg
```

SimpleNegotiator (`examples/simple.py`) A minimal agent implemented in a single function, demonstrating the basics of negotiation:

- **Opponent Model:** Assumes the opponent's utility is the inverse of ours
- **Acceptance:** Accepts any offer with utility >= 0.8
- **Offering:** Proposes random outcomes with utility between 0.5 and 1.0

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

```
anl2026 run --opponent examples.simple.SimpleNegotiator
```

> [!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.

4. Usage from the command line

> **Note:** All commands below work on Linux, macOS, and Windows.
>
> **Important:** Before running any commands, ensure your virtual environment is activated: - **If installed with uv:** Either activate the venv (see below), OR prefix commands with `uv run` (e.g., `uv run anl2026 run`) - Linux/macOS/Git Bash: `source .venv/bin/activate` - Windows (Command Prompt): `.venv\Scripts\activate` - Windows (PowerShell): `.venv\Scripts\Activate.ps1` - **If installed with pip:** Activate the venv (same commands as above)

Running a single negotiation

To run a single negotiation:

```
anl2026 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 ANL 2026 score

Run command options

Option	Description
`--scenario TEXT`	Scenario name (default: random)
`--generate-scenario`	Generate random scenario
`--rational-fraction FLOAT`	Rational outcomes fraction (default: 1.0)
`--negotiator TEXT` or `--agent TEXT`	Your negotiator class (default: mynegotiator.MyNegotiator)
`--opponent TEXT`	Opponent class (default: random)
`--negotiator-first / --opponent-first`	Control first mover (default: random)
`--verbose`	Show full trace with utilities

Option	Description
`--plot / --no-plot`	Plot negotiation trace (default: plot)
`--show-trace`	Display trace as table in terminal
`--export-trace PATH`	Export trace to CSV file
`--trace-browser / --no-trace-browser`	Open trace in browser (default: no)

Examples:

```
# Run with a specific scenario
anl2026 run --scenario Camera

# Run with a generated scenario
anl2026 run --generate-scenario

# Run with a specific negotiator class
anl2026 run --negotiator examples.boa.BOANeg

# You can also use --agent instead of --negotiator
anl2026 run --agent examples.map.MAPNeg

# Control if your negotiator goes first
anl2026 run --negotiator-first

# Make the opponent go first
anl2026 run --opponent-first

# Run against a specific opponent
# (Any negmas/genius agent works)
anl2026 run --opponent negmas.sao.BoulwareTBNegotiator

# Run against an example agent
anl2026 run --opponent examples.boa.BOANeg

# Test two specific agents
anl2026 run \
  --negotiator examples.boa.BOANeg \
  --opponent examples.map.MAPNeg

# Test with specific order
anl2026 run \
  --negotiator examples.boa.BOANeg \
  --opponent examples.map.MAPNeg \
  --negotiator-first

# Run with verbose output and no plot
anl2026 run --verbose --no-plot

# Display trace as a formatted table in the terminal
anl2026 run --show-trace

# Export trace to a CSV file
anl2026 run --export-trace trace.csv

# Open trace in browser for interactive viewing
anl2026 run --trace-browser
```

```
# Combine multiple trace options
anl2026 run \
  --show-trace \
  --export-trace trace.csv \
  --trace-browser
```

Running a tournament

To run a complete tournament with all default competitors:

```
anl2026 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)
`--competitor TEXT`	Competitor classes (repeatable)
`--scenario TEXT`	Scenario names (repeatable, 'all' for all)
`--generate-scenarios N`	Generate N random scenarios
`--rational-fraction FLOAT`	Rational outcomes fraction (0.0-1.0)
`--verbosity INT`	Verbosity level 0-5 (default: 0)
`--parallel`	Run in parallel using all cores

Examples:

```
# Run tournament on specific scenarios
anl2026 tournament --scenario Camera --scenario Car

# Run tournament with generated scenarios
anl2026 tournament --generate-scenarios 10

# Run tournament in parallel with verbose output
anl2026 tournament --parallel --verbosity 1

# Run tournament with custom competitors
anl2026 tournament \
  --competitor mynegotiator.MyNegotiator \
  --competitor examples.boa.BOANeg
```

Viewing Development and Submission Info

To view development workflows and submission instructions in your terminal:

```
anl2026 info
```

5. 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 from `.venv/bin/python` or `.venv\Scripts\python.exe`

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 `anl2026 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:

   ```
   anl2026 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 from `.venv/bin/python` or `.venv\Scripts\python.exe`
3. Edit your agent file (renamed from `mynegotiator.py`)
4. Use the built-in terminal to run:

   ```
   anl2026 run
   pytest
   ```

6. Submission

To submit your agent to the ANL 2026 competition:

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

`anl2026 run`

and

`anl2026 tournament`

2. Test your agent locally using `anl2026 run` and `anl2026 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. Create your submission zip file:

   Using the provided scripts (recommended for all platforms):

   Linux/macOS/Git Bash:

   `./make_submission.sh`

   Windows (Command Prompt or PowerShell):

   `make_submission.bat`

   > **Note:** The Windows script requires PowerShell, which is included by default in Windows 7 and later.

   Both scripts will:

   - Create `submission.zip` in your project root

- Include all necessary files (your renamed negotiator, requirements.txt, any additional Python/data files)
- Exclude development files, examples, tests, and configuration files
- Show the contents of the created zip file

Or manually:

Linux/macOS/Git Bash:

```
zip -r submission.zip . \
  -x ".*" \
  -x ".git/*" \
  -x ".venv/*" \
  -x "examples/*" \
  -x "scenarios/*" \
  -x "report/*" \
  -x "tests/*"
```

Windows (Command Prompt with tar):

"'cmd tar -a -cf submission.zip ^ –exclude=.* ^ –exclude=.git ^ –exclude=.venv ^ –exclude=examples ^ –exclude=scenarios ^ –exclude=report ^ –exclude=tests ^ .

You can use any app to generate this zip file but, remember to exclude all folders starting with "." (e.g. .git, .venv), and the scenarios, report and tests folders to avoid exceeding the submission size limit. These are not needed in your submission anyway.

5. Submit your agent following the competition guidelines at ANL 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 "ANL", click "New Agent" and fill the form. Example assuming your agent class is `AwsomeNegotiator` in `awsome.py`:

   - **Agent Name:** Awsome Negotiator
   - **Agent Alias:** Awsome Negotiator
   - **Agent Module:** awsome
   - **Agent Class:** AwsomeNegotiator
   - **Dependencies:** Any packages you installed via `pip install` or `uv add` (semicolon-separated, no spaces)
   - **Code:** submission.zip
   - **Requirements File:** requirements.txt
   - **Report:** Upload your report PDF (required for final submission only; drafts can be submitted anytime)

     [!NOTE] If you need data files (e.g., trained models), ensure they're in your submission zip. See Accessing Data Files for details (written for SCML but process is identical).

You can submit your agent multiple times before the deadline. Submit early and frequently! We test submissions and provide feedback on failures. Track your progress on the leaderboard.

7. Troubleshooting

Clearing Python Cache

If you're experiencing unexpected behavior after modifying your code (e.g., old code still running), Python's bytecode cache (`.pyc` files) may be stale. Clear the cache with:

Linux/macOS/Git Bash:

```
find . -type d -name "__pycache__" -exec rm -rf {} + 2>/dev/null
find . -type f -name "*.pyc" -delete 2>/dev/null
```

Windows (PowerShell):

```
Get-ChildItem -Recurse -Directory -Filter "__pycache__" | Remove-Item -Recurse -Force
Get-ChildItem -Recurse -Filter "*.pyc" | Remove-Item -Force
```

Reinstalling Dependencies

If you're having issues with dependencies or want to ensure a clean environment:

Using uv (Recommended):

```
# Remove the virtual environment and reinstall everything
rm -rf .venv  # Linux/macOS/Git Bash
# OR on Windows: rmdir /s /q .venv

# Recreate and sync
uv sync
```

Using pip:

```
# Remove and recreate the virtual environment
rm -rf .venv  # Linux/macOS/Git Bash
# OR on Windows: rmdir /s /q .venv

python -m venv .venv
source .venv/bin/activate  # Linux/macOS/Git Bash
# OR on Windows: .venv\Scripts\activate

pip install -e .
```

Reinstalling the anl2026 Command

If the `anl2026` command is not working correctly or running outdated code:

Using uv:

```
# Reinstall the package (this also reinstalls the CLI command)
uv sync --reinstall-package anl2026
```

Using pip:

```
# Reinstall in editable mode
pip install -e . --force-reinstall --no-deps
```

If the command still doesn't work, verify it's pointing to the correct location:

```
# Check where the command is installed
which anl2026  # Linux/macOS/Git Bash
# OR on Windows: where anl2026

# Verify the Python environment
anl2026 --help
```

> [!TIP] If you have multiple Python environments, make sure your virtual environment is activated before running `anl2026`. You can also use `uv run anl2026` to ensure the correct environment is used.