README — RL Trading Agent in Synthetic Multi-Good Market

This repository contains a **synthetic multi-good market environment** with **endogenous price dynamics** and a **reinforcement learning (PPO) trading agent**. Training logs are exported to Excel files and can then be analyzed to generate plots.

---

1) Project structure

The project must follow this structure:

```
PROJECT_ROOT/
├─ game/                      # game / simulation code (Pygame)
│  ├─ MainFrame.py            # main entry point for the game
│  ├─ Solar_System.py
│  ├─ objects.py
│  ├─ camera.py
│  ├─ drawing.py
│  ├─ UI_manager.py
│  └─ Images/                 # icons, UI images (play, pause, etc.)
└─ ML/
   ├─ ML_main.py              # main training launcher (runs multiple
configs)
   ├─ prices_process.py       # price process parameters + market dynamics
   ├─ ML_nd_priceprocess_functions.py   # environment, logging, PPO helpers
   ├─ Results_analysis.py     # data loading, aggregation, plots
   └─ plots/                  # auto-generated figures
```

Important: all game-related scripts (`drawing.py`, `camera.py`, `UI_manager.py`, `Solar_System.py`, `objects.py`) **must be located in the same** `game/` **folder**. They rely on direct Python imports between each other.

---

2) Installation / requirements

This project has two parts: **(i) the game/simulation (Pygame)** and **(ii) the RL + analysis stack**.

External libraries (pip / conda)

Game (Pygame) - `pygame` filecite turn1file0 filecite turn1file5

RL training + environment - `numpy` filecite turn0file1 filecite turn0file2 - `pandas` filecite turn0file1 filecite turn0file2 - `gymnasium` filecite turn0file2 - `stable-baselines3`

filecite turn0file1 filecite turn0file2 filecite turn0file0 - `torch` (PyTorch) filecite turn0file1 filecite turn0file2

Plots / analysis - `matplotlib` (incl. `matplotlib.patheffects`) filecite turn0file2 filecite turn0file3 - `openpyxl` (needed to read/write `.xlsx` logs via pandas)

Python standard library modules used

- `sys` filecite turn1file5
- `time` filecite turn0file1 filecite turn0file2
- `math` filecite turn1file4 filecite turn0file2
- `random` filecite turn1file4
- `copy` (`deepcopy`) filecite turn0file1
- `pathlib` (`Path`) filecite turn0file3
- `dataclasses` filecite turn0file2
- `typing` filecite turn0file2

---

3) Running the game (simulation)

The game / simulation is implemented using **Pygame** and serves as the interactive visual environment.

How to run the game

From the project root:

```
python game/MainFrame.py
```

This is the **only file you need to execute** to run the game.

- `MainFrame.py` initializes Pygame, the camera, UI, and the first solar system
- All other game scripts are imported automatically and must **not** be run individually

Game scripts overview

- `MainFrame.py` — main loop, event handling, rendering
- `Solar_System.py` — star systems, planets, entity management
- `objects.py` — planet and star classes
- `camera.py` — zooming, panning, world-to-screen conversion
- `drawing.py` — low-level drawing helpers
- `UI_manager.py` — UI elements, buttons, panels

If any of these files are moved or renamed, imports will break.

---

4) Training the models (RL)

Training is launched from `ML/ML_main.py`.

Run training

From the project root:

```
python ML/ML_main.py
```

This will train several models sequentially and produce Excel files such as:

```
model_value_lenght_60000_raw.xlsx
model_value_lenght_40000_ent_coef_0.005_raw.xlsx
...
```

Each file contains detailed step-level logs of the agent–market interaction.

---

4) How to experiment with the model

There are **three main places** where you can modify the system.

A) Change RL / PPO hyperparameters

Edit the default values inside the `make_config()` function in `ML_main.py`.

Typical parameters you may want to adjust:

- PPO hyperparameters: `learning_rate`, `clip_range`, `ent_coef`, `gamma`, `gae_lambda`, `n_steps`, `batch_size`, `n_epochs`
- Training horizon: `train_episodes_chunks`
- Episode structure: `episode_len_t`, `window_size`
- Network architecture: `pi_arch`, `vf_arch`, `activation_fn`

Multiple experiments are defined via a list of configs:

```
runs = [
    make_config(train_episodes_chunks=200),
    make_config(train_episodes_chunks=300),
    make_config(train_episodes_chunks=600),

    make_config(learning_rate=3e-4),
    make_config(ent_coef=0.005),
    make_config(clip_range=0.2),
]
```

To add a new experiment, simply add another `make_config(...)` entry.

B) Change the market price process

Edit the function `build_market_matrices()` in `prices_process.py`.

This function defines the **data-generating process** of prices, including:

- Long-run mean log-prices
- Mean reversion matrix
- Instant price impact of trades
- Persistent (decaying) impact of past trades
- Cross-good delayed impact
- Shock covariance and correlation structure
- Market volume / depth parameters

Examples:

- Increase volatility → increase shock variances
- Faster mean reversion → reduce diagonal entries of the reversion matrix
- Stronger market impact → increase instant or persistent impact coefficients
- More cross-good interaction → adjust cross-impact matrices

This is the main place to experiment with different market environments.

C) Change policy architecture

Still inside `make_config()` in `ML_main.py`, you can modify:

- Number of hidden layers
- Size of each layer
- Activation function

Example:

```
pi_arch = (256, 256)
vf_arch = (256, 256)
activation_fn = torch.nn.Tanh
```

---

5) Analysis and plots

All post-training analysis is handled in `ML/Results_analysis.py`.

A) Register raw output files

Inside the function `import_files_raw()`, list all Excel files you want to analyze:

```
FILES = {
    "120k data": "model_value_lenght_120000_raw.xlsx",
    "60k data":  "model_value_lenght_60000_raw.xlsx",
    "40k data":  "model_value_lenght_40000_raw.xlsx",

    "higher learning rate":
"model_value_lenght_60000_learning_rate_0.0003_raw.xlsx",
    "lower entropy coefficient":
"model_value_lenght_60000_ent_coef_0.005_raw.xlsx",
    "higher clip range": "model_value_lenght_60000_clip_range_0.2_raw.xlsx",
}
```

The keys are the labels used in plots; the values are the Excel filenames.

B) Run analysis

From the project root:

```
python ML/Results_analysis.py
```

This script:

1. Loads all registered Excel files
2. Builds normalized variables (wealth, cash)
3. Aggregates data across time, episodes, and learning phases
4. Saves a clean long-format dataset
5. Generates plots into `ML/plots/`

---

6) Outputs

Training outputs

- Raw logs: `model_value_lenght_*_raw.xlsx`

Analysis outputs

- `CLEAN_ALL.csv` (cached aggregated dataset)
- Multiple `.png` figures in `ML/plots/`, including:
- Wealth and cash evolution
- Reward and penalty diagnostics
- Within-episode dynamics
- Cross-model comparisons

---

7) Typical workflow

1. Decide what to test (training horizon, PPO hyperparameters, architecture, market structure)
2. Modify:
3. `make_config()` and the runs list in `ML_main.py`
4. `build_market_matrices()` in `prices_process.py`
5. Run training: `python ML/ML_main.py`
6. Register new output files in `import_files_raw()`
7. Run analysis: `python ML/Results_analysis.py`
8. Inspect figures in `ML/plots/`

This setup is designed as a **controlled experimental testbed**: the market is fully synthetic, the agent is trained under fixed constraints, and all dynamics are logged for reproducible analysis.