Fehleranalyse
Falsche oder doppelte Task/MDP-Registrierung

Problemstellung

Die Task-/MDP-Registrierung läuft über eine Registry, die Namen wie `pick_apples_v1` auf Klassen mappt. Die Registrierung passiert beim Import eines Moduls (Side-Effect). Wird ein Modul nicht importiert, ist die Task nicht registriert. Wird es mehrfach oder unter verschiedenem Modulnamen importiert, landet die Task doppelt in der Registry. Stimmen Registry-Name, Config und Startparameter nicht exakt überein, schlägt das Lookup fehl.

Typische Symptome

- `KeyError: task 'pick_apples_v1' not found`
- `ValueError: task 'pick_apples_v1' already registered`
- `ImportError: cannot import name ... from partially initialized module ...`
- Start bricht ab, obwohl der Code korrekt wirkt (Cache/Altpfad wird geladen).

Häufige Ursachen

- Registrierung an mehreren Stellen (z. B. in `__init__.py` und in `registry.py`).
- Unterschiedliche Modulnamen für denselben Code (doppelter Eintrag in `sys.modules`).
- Inkonsistenter Taskname: Registry, Config und CLI/Launcher verwenden leicht unterschiedliche Strings.
- Zirkuläre Imports führen zu `partially initialized module`.
- Alter Bytecode/Kit-Cache aus einer anderen Szene.

Minimalbeispiel für einen Zyklus

mdp/___init___.py

```
from .terminations import is_done # side-effect beim paketimport
from .mdp import PickApplesMDP # zieht mdp.py
```

mdp/terminations.py

```
from .mdp import PickApplesMDP # import zurueck nach mdp.py -> Zyklus
def is_done(state):
    ...
```

mdp/mdp.py

```
from .terminations import is_done # wiederum zurueck zu terminations.py
class PickApplesMDP:
    ...
```

Fix-Patterns (robuste Struktur)

Pattern A: Single Source of Truth

- Exakter Taskname an *einer* Stelle definiert, z. B. `TASK_NAME = "pick_apples_v1"`.
- `registry.py` führt einmalig die Registrierung aus; `__init__.py` nur Exporte, keine Side-Effects.

Pattern B: Ebenen entflechten

- `terminations.py` und `rewards.py` sind Leaf-Module ohne Rückimporte in `mdp.py` oder `task.py`.
- Gemeinsame Konstanten/Typen in `common.py` oder `types.py`.

Pattern C: Lazy-Imports und TYPE_CHECKING

- Optionale oder seltene Imports in Funktionskörper verlagern.
- Typannotationen mit `from typing import TYPE_CHECKING` kapseln:

```
from typing import TYPE_CHECKING
if TYPE_CHECKING:
    from .mdp import PickApplesMDP
```

Pattern D: Konsistenz mit Config/CLI

- CLI: `-task pick_apples_v1`
- YAML: `task:\ name:\ pick_apples_v1`
- Registry: `TASK_NAME = "pick_apples_v1"`

Beispiel: Saubere Registry-Struktur

core/registry.py

```
class TaskRegistry:
    def __init__(self):
        self._map = {}
    def register_task(self, name, cls, **kw):
        if name in self._map:
            raise ValueError(f"Task␣'{name}'␣already␣registered")
        self._map[name] = (cls, kw)
    def keys(self):
        return self._map.keys()
    def clear(self):
        self._map.clear()

task_registry = TaskRegistry()
```

tasks/pick_apples/registry.py

```
from .task import PickApplesTask
TASK_NAME = "pick_apples_v1"

def register(reg):
    reg.register_task(TASK_NAME, PickApplesTask)
```

tasks/pick_apples/___init___.py (nur Exporte, keine Registrierung)

```
from .task import PickApplesTask
from .registry import TASK_NAME
```

launch.py

```
from core.registry import task_registry
from tasks.pick_apples.registry import register, TASK_NAME

register(task_registry)
assert TASK_NAME in task_registry.keys()
# task_registry.create(TASK_NAME, cfg) # je nach Framework
```

Debug-Playbook (Schritt für Schritt)

1. Name abgleichen: CLI/Launcher, Config und `TASK_NAME` exakt identisch?
2. Registry inspizieren (früh im Start): Keys ausgeben, `assert "pick_apples_v1" in registry`.
3. Registrierungsstellen suchen:

   ```
   rg -n "register_task|register\(.*pick_apples" -g '!venv' -g '!**/__pycache__/**'
   ```

4. Zirkuläre Imports entschärfen: Stern-Imports vermeiden, TYPE_CHECKING/Lazy-Imports nutzen.
5. Duplicate Module/Pfade prüfen: `sys.path` und `pkg.__file__` ausgeben.
6. Caches löschen nach Änderungen:

   ```
   find . -type d -name "__pycache__" -exec rm -rf {} +
   rm -rf .pytest_cache .cache outputs
   rm -rf ~/.cache/ov ~/.local/share/ov/Kit*
   ```

Tests und Guards

```
# tests/test_registry.py
from core.registry import task_registry
from tasks.pick_apples.registry import register, TASK_NAME

def test_task_registered_once():
    task_registry.clear()
    register(task_registry)
    keys = [k for k in task_registry.keys() if k == TASK_NAME]
    assert len(keys) == 1
```

Quick-Checklist

- Genau eine Datei registriert die Task (keine Side-Effects in `__init__.py`).
- `TASK_NAME` als Single Source of Truth; identisch in CLI, Config, Registry.
- Keine zirkulären Imports zwischen `mdp`, `terminations`, `rewards`, `task`.
- Nach Strukturänderungen Caches löschen und Prozess neu starten.
- Früher Registry-Check: Keys loggen und auf Einmaligkeit testen.