**Eco‑Simulator**

**Dual‑Layer Architecture for Data‑Driven Ecology**

**Purpose for Devs & Researchers**  — Run 10⁴+ macro runs per hour **and** explore emergent agent behaviour in a single code‑base.

**0. TL;DR**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Lines‑of‑Code** | **Typical batch size** | **Runtime (1 CPU)** | **Questions answered** |
| **Macro (ODE)** | ≈ 350 LoC | 10 k param sets | 0.3 s/run | *„Ist das System bei α ∈ [0.05‑0.3] stabil?“* |
| **Micro (ABM)** | ≈ 420 LoC | 10‑100 agents × 1‑5 k steps | 2‑5 s/run | *„Wie entsteht Schwarm‑Jagen bei Curiosity > 0.6?“* |

Both layers write a uniform metric schema → can be pushed into the same analytics pipeline / ML model.

**1️ Macro Layer — Predator‑Prey ODE Engine**

* **Model**: 6‑dim Lotka‑Volterra‑style system, incl. bacteria, virus, immune & nutrients.
* **Key params**:
  + **α** prey intrinsic growth (0.01‑0.5)
  + **β** predator attack rate (1e‑3‑0.1)
  + **φ** virus infection rate (1e‑3‑1e‑2)
* **Throughput**: ~3×10⁵ timesteps / sec on M1 Air.
* **Code**: predator\_prey\_sim/simulator.py, integrators (Euler, RK4, SciPy RK45).
* **Output**: ndarray → CSV (≤ 300 rows) in sensitivity\_results/.
* **Use‑cases**: Latin‑Hypercube sampling, Sobol sensitivity, Bayesian calibration.

**Dev Notes**

* Numerics guarded by overflow‑clamp (1e9) & NaN abort.
* Deterministic; perfect for CI regression tests.

**2️ Micro Layer — Grid ABM (Agent Ecosystem)**

* **Agent DNA**: {speed, sight, strength, camouflage, curiosity, mutation\_rate}
* **Environment**: toroidal grid, continuous nutrient field.
* **Behaviour loop**: perceive → decide (ε‑greedy exploration via Curiosity) → move → eat → reproduce (mutate) → age.
* **Key params**:
  + **mutation\_rate** ∈ [0‑0.2] (σ of Gaussian mutator)
  + **curiosity** ∈ [0‑1] (probability to explore vs exploit)
* **Scales**: 1‑200 agents @ 500 steps in 2‑5 s (NumPy vectorised where possible).
* **Code**: agent\_ecosystem/simulator.py, agent.py.
* **Output**: time‑series + run‑level KPIs (population, avg\_speed, avg\_curiosity).

**Dev Notes**

* Designed for plug‑in RL brains later (policy nets via PyTorch).
* Mutation & Curiosity fed back into analytics → can inform macro params.

**Cross‑Talk – Why 2 Layers?**

|  |  |  |
| --- | --- | --- |
| 📍 Question | Macro answers | Micro answers |
| Stability range for (α,β,φ)? | ✅ Fast grid‑search | 🚫 Too slow |
| Individual adaptation & emergent tactics? | 🚫 Aggregated | ✅ Visible |
| Parameter seed for GA/Bayes optimisation? | ✅ Provides priors | ✅ Refines priors |

**Bottom‑up**: agent metrics (e.g. mean prey kill rate) can be aggregated → feed into ODE model calibration.

**Top‑down**: macro sweeps narrow feasible region → micro runs only where interesting.

**Analytics Pipeline (already coded)**

1. **Batch** → tools/run\_sensitivity.py
2. **CSV** → analyze\_results.py (adds Stability label)
3. **Recommender** → recommend\_params.py (actionable tips)
4. **Dashboard** → tools/control\_dashboard.py
   * KPI tiles, correlation heatmap, trend scatters, filter “instable only”.
5. **ML** → ml\_predictor.py Random‑Forest stub → can be retrained on aggregated features.

**Key Metrics & Defaults**

* **ODE default**: time\_steps = 300, dt = 0.1 → 30 s simulated time.
* **ABM default**: grid = 50×50, agents = 20, mutation\_rate = 0.05.
* **Stability thresholds** (heuristic): prey > 20 & predator > 5 ⇒ Stable; prey < 5 or virus > 50 ⇒ Critical.

**Take‑aways for Devs**

* Use **Macro layer** for rapid param‑sweeps (10³‑10⁴ configs/hour).
* Drill down with **Micro layer** when you find a “hot region”.
* Unified CSV schema → can stack data and run cross‑model ML.
* Overflow/NaN safe by design; unit‑tests in tests/test\_simulator.py.

**Eco‑Simulator  — Zwei‑Ebenen‑Architektur (Makro + Mikro)**

**Zielgruppe**: Entwickler:innen, Data‑Scientists, Ökolog:innen.  
**Kurzfassung**: Kombiniert **10⁴ Makro‑Runs pro Stunde** mit **agentenbasierter Verhaltens‑Evolution** – alles in einem Repository.

**Übersicht (Was, Warum, Wie, Wo, Wer)**

| **Frage** | **Antwort** |
| --- | --- |
| **Was** | Zwei getrennte Simulatoren (ODE + ABM) arbeiten auf denselben Daten‑Pipelines. |
| **Warum** | ODE = superschnell für globale Stabilität; ABM = Detailtiefe für emergente Muster. |
| **Wie** | Python 3.11, NumPy, Plotly, Streamlit, Multiprocessing. CSV‑Schema ist identisch → eine Analyse‑Schicht. |
| **Wo** | predator\_prey\_sim/ (Makro) • agent\_ecosystem/ (Mikro) • tools/ (Analyse & Dashboard) |
| **Wer** | Dev‑Team, Forschende, Studierende. Makro‑Layer für schnelle Hypothesen, Mikro‑Layer für tiefe Verhaltens‑Studien. |

**1️ Makro‑Ebene (Populationsdynamik / Differentialgleichungen)**

**Kennzahlen**

* **LOC**: ≈ 350
* **Standardlauf**: 300 Zeitschritte × 0,1 dt → 30 simulierte s
* **Durchsatz**: ~3×10⁵ Schritte/s (M1‑Air, RK4)

**Parameter (α, β, φ …)**

| **Kürzel** | **Bedeutung** | **Range (Demo)** |
| --- | --- | --- |
| α | Beute‑Wachstum | 0,05 – 0,3 |
| β | Räuber‑Angriffsrate | 0,005 – 0,03 |
| φ | Virus‑Infektionsrate | 0,001 – 0,01 |

**Warum relevant?**

* **Schnelle Sensitivität** – bis 10⁴ Konfigurationen/h.
* Liefert **Stabilitätskorridore** als Heatmap (α vs β).

**Module & Pfade**

* predator\_prey\_sim/simulator.py – Kern‑ODEs
* tools/sensitivity\_engine.py – Multiprocessing‑Batch
* Ergebnisse: sensitivity\_results/results\_\*.csv

**2️ Mikro‑Ebene (Agentenbasierte Simulation)**

**Kennzahlen**

* **LOC**: ≈ 420
* **Default**: 50 × 50 Grid, 20 Agenten, 500 Steps → 2–5 s/Lauf.
* **Skalierbar**: bis 200 Agenten < 15 s.

**Agent‑DNA**

{speed, sight, strength, camouflage, curiosity, mutation\_rate}

**Parameterbeispiele**

| **Param** | **Rolle** | **Effekt bei Erhöhung** |
| --- | --- | --- |
| mutation\_rate | Evolutions‑Streuung | + Vielfalt, ± Stabilität |
| curiosity | Erkundungs‑Drang | + Ressourcenfindung, – Energieeffizienz |

**Warum relevant?**

* Zeigt **emergente Phänomene** (z. B. Schwarmjagd bei curiosity > 0,6).
* Erlaubt Studien zu **Mutation & Selektion**.

**Module & Pfade**

* agent\_ecosystem/agent.py, simulator.py
* Ergebnisse liegen ebenfalls in sensitivity\_results/ (gemeinsames Schema).

**Zusammenspiel (Bottom‑up ⇄ Top‑down)**

| **Ebene** | **Liefert** | **Nutzt** |
| --- | --- | --- |
| Makro | Globale Stabilität, Parameter‑Heatmaps | Startbereich für ABM‑Runs |
| Mikro | Mikrometriken (avg\_speed, kill\_rate) | Feature‑Engineering für ML‑Regressor |

**Pipeline:** Makro‑Sweep → stabile Insel → Mikro‑Runs → Verhalten validieren → ggf. Parameter zurück in Makro.

Mermaid‑Diagramm:

flowchart LR

A[param\_list.json] -->|Batch| B(Makro ODE)

A --> C(Mikro ABM)

B --> D[CSV\_Macro]

C --> E[CSV\_Micro]

D & E --> F[analyze\_results]

F --> G[control\_dashboard]

F --> H[ml\_predictor]

**Aktuelle Analytics‑Pipeline**

1. **Batch‑Run** → tools/run\_sensitivity.py
2. **Analyse** → analyze\_results.py (Stabilität + KPIs)
3. **Empfehlungen** → recommend\_params.py
4. **Dashboard** → tools/control\_dashboard.py (Dark‑Theme, Filter „nur instabile“)
5. **Vorhersage** → ml\_predictor.py (Random‑Forest‑Stub)

**Fazit für Dev‑Teams**

* **Makro** ⇒ Grob‑Scan, 10³–10⁴ Runs/h.
* **Mikro** ⇒ Tiefenanalyse, Evolution & Verhalten.
* Einheitliches CSV‑Schema ⇒ **eine** Analyse‑Schicht.
* Overflow/NaN‑Schutz & Unit‑Tests eingebaut.

**Kurz:** Maximale Geschwindigkeit **und** maximale Detailtiefe – ohne Code‑Duplikation.