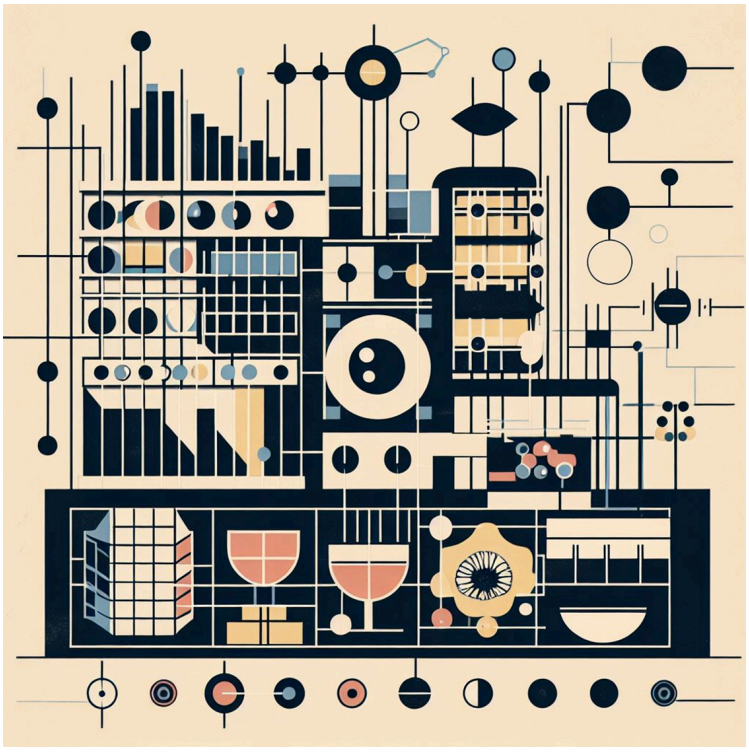
*Data Science und KI Infrastrukturen, Wintersemester 2025-2026

Projekt: Biology-inspired algorithms - emergent behavior



Lernziele:

- Umsetzung eines komplexen Programmierprojekts im Team.
- Entwicklung einer Python-Software unter Einhaltung aktueller Standards und "Best practices".
- Einblick in biologisch inspirierte Algorithmen.
- Erfahrung sammeln mit Python Laufzeitoptimierung.

Ablauf

- Projektmanagement über GitHub und Kanban-Board (inkl. Entwicklung in branches, Pull Requests und Code Reviews)
- Wöchentliche Update-Meetings (bis zum Ende des Vorlesungszeitraums, danach Treffen nach Absprache)
- Abgabe bestehend aus: Python-Software, Dokumentation
- **Zwischenprüfungen/Milestones**: Alle paar Wochen Präsentation der Zwischenergebnisse/Milestones
- **Projektprüfung** (Voraussichtlich: 25.02.2026)

Projekt: Particle Life Simulator

Ziel:

Entwickelt eine Python-Softwareanwendung, die ein dynamisches Partikelsystem simuliert, bei dem tausende Partikel basierend auf vordefinierten Regeln interagieren. Die Simulation soll verschiedene Partikeltypen mit einzigartigen Eigenschaften und Interaktionsmustern enthalten und dabei emergentes Verhalten und visuelle Komplexität demonstrieren.

Beispiele gibt es z.B. hier: https://webgl-particle-life.netlify.app/
Oder hier: https://particle-life.com/java-framework/overview.html

Projektübersicht:

Ihr arbeitet in Teams (idealerweise 4 Studierende), um eine Particle Life-Simulation zu implementieren. Das Hauptziel ist es, eine visuell ansprechende und recheneffiziente Simulation zu erstellen. Dem Kurs entsprechend spielen Performance-Optimierung, Codequalität, Testing und die effektive Nutzung von Versionskontrolle eine große Rolle.

Jedes Team wird:

1. Die Kernlogik der Simulation für die Partikelinteraktion implementieren.
2. Mehrere Partikeltypen und Interaktionsregeln definieren.
3. Die Performance der Simulation mit fortgeschrittenen Python-Techniken optimieren.
4. Tests für Kernfunktionen schreiben und CI (Continuous Integration) einrichten, um das Testing zu automatisieren.
5. Git und GitHub für die kollaborative Entwicklung verwenden.

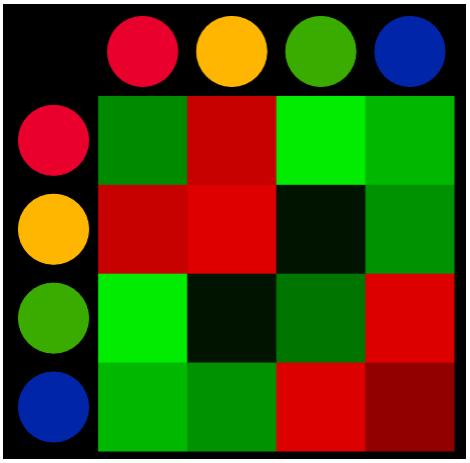
Projektumfang

1. Python Software 35%

Kernlogik der Simulation:

- Implementiert die grundlegende Simulationsschleife, bei der sich Partikel basierend auf den definierten Regeln bewegen und interagieren.
- Jeder Partikel sollte Eigenschaften wie Position, Geschwindigkeit und Typ haben.
- Die Partikel sollten basierend auf Abstand und typabhängigen Regeln interagieren, die in einer Matrix

definiert sind (z. B. Anziehung, Abstoßung, neutral). Zum Beispiel so:



Partikeltypen und Interaktionen:

- Implementiert mindestens 4 verschiedene Partikeltypen, die jeweils durch eine einzigartige Farbe visuell unterschieden werden.
- Definiert eine Interaktionsmatrix, bei der jede Kombination von Partikeltypen ein anderes Verhalten aufweist (z. B. Typ A zieht Typ B an, Typ B stößt Typ C ab).
- Zusätzlich nötige Parameter sind:
  Interaktionsstärke, maximale Einflussreichweite, Reibung, zusätzliche Zufallsbewegung.
- Die Simulation soll (falls technisch möglich) in Echtzeit dargestellt werden, mit > 2,000 Partikeln. Sollte dies nicht möglich sein, z.B: da einzelne Zeitschritte der Simulation zu lange zur Berechnung brauchen, dann soll ein Video der Simulation ausgegeben werden.
- Optional (**Pflicht für Teams mit 5 Studierenden**): GUI um die Simulationsparameter zu ändern.
- **Für Teams mit 5 Studierenden**: Es soll möglich sein live während der laufenden Simulation die Kräfte zu ändern.

2. Code quality 15%

Ziel des Projektes ist es u.a., eine Python-Software nach professionellen Standards zu erstellen. Dazu sollen viele der in diesem Modul eingeführten Techniken umgesetzt werden:

- Beachtung allgemeiner "Clean Code" und Formatierungsregeln. Also: gut strukturierter, gut dokumentierter, lesbarer Code der zumindest grundlegende Linter größtenteils zufrieden stellt (einzelne Linter-Regeln dürfen gerne umgangen werden...).
- Einsatz von `Unit Tests` (z.B: unter Verwendung von pytest), die > 70% des entwickelten Codes abdecken sollen.
- Automatischer Workflow ("CI") über GitHub Actions der die Software auf verschiedenen Systemen testet (Windows, MacOS, Ubuntu). Dieser Workflow soll einen Linter enthalten und die Unit Tests ausführen.
- Verwendung von `Docstrings` bei allen zentralen Funktionen, Klassen und Methoden.
- Übergeordnetes Code-Design, graphisch dargestellt (sollte bei der Abschlusspräsentation/Dokumentation gezeigt werden).

3. Performance Profiling & Optimierung 15%

Unterschiedliche Algorithmen und deren Implementierung unterscheiden sich manchmal drastisch in ihrer Laufzeit. Für diesem Projekt spielt die Laufzeit eine große Rolle, darum sollten grundlegende Schritte zur Messung und Optimierung unternommen werden.

- Nutzung von Profiling-Tools (z.B. time, timeit, cProfile) um die größten Schwachstellen (`Bottlenecks`) zu entdecken.
- Laufzeitverbesserung der Software durch den Einsatz passender Techniken (z.B. geeignete Algorithmenwahl, just-in-time Kompilierung, Parallelisierung, Nutzung optimierter Bibliotheken...)

4. Projektmanagement 15%

Es wird von allen Team-Mitgliedern erwartet, dass sie aktiv am Projekt mitarbeiten. Zur Planung innerhalb des Teams, wie auch zur Dokumentation soll die Software-Entwicklung zentral über GitHub laufen. Dies bedeutet u.a.

- Aufgaben in möglichst überschaubare `Issues` einteilen.
- `issues` untereinander zuweisen (assign)
- Arbeit am Projekt über `branches` und `pull requests`
- Gegenseitige `Code reviews`, zum einen zur Verbesserung der Code-Qualität, aber auch um sicherzustellen, dass alle Code-Teile von mehreren Team-Mitgliedern verstanden werden
- Nutzung eines `GitHub Projects` als Kanban-Board für die kontinuierliche Visualisierung des Projekfortschritts.

5. Projektpräsentation (& Dokumentation) 20%

Zum Abschluss soll das Projekt vom gesamten Team präsentiert werden. Dazu gehört:

- Nutzer*innen Dokumentation + Developer-Dokumentation in der `README.md` auf GitHub.
  Hier bitte beschreiben wie die Software genutzt werden kann und wie der Code strukturiert ist.
- Vortrag + Demo.
  In einem Vortrag die Software und die zugrundeliegenden Konzepte/Ideen vorstellen, sowie natürlich die damit erzielten Ergebnisse. Dazu gehört auch eine live-Vorführung der fertigen Software.
- Fachgespräch/Diskussion
  Zu den präsentierten Ergebnissen und der eigenen Software gibt es im Anschluss noch ein Fachgespräch, d.h. Fragen und Diskussionen (ca. 20-25 Minuten).

Milestones

Milestone 1: Projektsetup - 19.11.2025

- Teams sind gefunden, GitHub Repo ist eingerichtet + Team Mitglieder*innen hinzugefügt
- Vorstellung Software Architektur (Diagramm)
- README.md mit Projektidee & Aufgabenbeschreibung
- CI ist eingerichtet
- Aufgaben als Issues angelegt und Kanban Board aufgesetzt (mit Zeitplanung)

Milestone 2: Basis-Simulation - 17.12.2025

- Particles Klasse mit Position, Geschwindigkeit, Typ vorhanden (nicht für einzelne Teilchen sondern alle/viele)
- Simulation mit Update-Schritte, Bewegung, Interaktion vorhanden
- Interaktionsmatrix vorhanden
- grundlegende Simulation funktoniert über Konsole

Milestone 3: Visualisierung - 17.12.2025

- (Echtzeit-)Visualisierung mit 4 Partikeltypen funktioniert
- Parameter (z.B. Reibung) können definiert werden
- Tests für wichtigste Komponenten vorhanden + passen

Milestone 4: Optimierung

- Funktionen wurden geprofiled und sind performant
- Bottlenecks wurde behoben und >=1000 Partikel können simuliert werden
- Tests passen und Coverage passt

Milestone 5: Projektpräsentation (& Dokumentation) - 25.02.2026

- Alles ist dokumentiert und ihr könnt euer Projekt vorstellen.

Verwendung von Python Bibliotheken

Es dürfen natürlich externe Python Bibliotheken eingesetzt werden. Ausgeschlossen sind Bibliotheken (sowie externer Code), die bereits die eigentlichen biologischen Algorithmen umsetzten.

Laufzeitoptimierung

Wie im Modul besprochen (Dezember), eignen sich hier Bibliotheken wie z.B. `numpy`, `numba`, `dask`.

Graphisches Interface & dynamische Visualisierungen

Python ist eigentlich nicht dafür bekannt, dass man damit besonders gut Graphische Schnittstellen bauen kann (GUIs, Graphical User Interface). Aber es ist für unsere Zwecke ausreichend gut möglich. Zum Beispiel über Bibliotheken wie:

- `VisPy` (dynamische Plots)
  Sehr performante Bibliothek für dynamische/interaktvie Visualisierungen, nutzt dafür OpenGL. `OpenGL` lässt sich auch direkt nutzen, z.B. über `PyOpenGL`, aber `VisPy` wirkt zugänglicher. Lässt sich (zur Steuerung) auch mit PyQt verbinden.
- `tkinter` (GUI + dynamische Plots)
  Relativ einfach zu nutzen, viele Tutorials, reicht für einfache Interfaces und einfache graphische Umsetzungen. **Achtung: nicht sehr performant!**
- `pygame` (GUI + dynamische Plots)

Viele Tutorials, komplexer als tkinter, dafür aber deutlich mehr Möglichkeiten (aber nicht so performant wie vispy).