Maturitätsarbeit an der Kantonsschule Uster

Fachschaft Informatik

Evolution und kooperatives Verhalten von Einzellern

Eine computerbasierte Simulation

Julian Heer (2008)

07. Januar 2026

Betreuerin: Theresa Luternauer

Expertin: Seung Hee Ma

Abstract

In dieser Arbeit wird die Entwicklung von Einzellern in einer computerbasierten Simulation untersucht. Ziel ist es, zu erforschen, ob evolutionäre Vorgänge simuliert werden können und dabei kooperative Verhaltensmuster unter den Einzellern entstehen. Dafür wird eine Simulationsumgebung entwickelt, in der sich stark vereinfachte Einzeller fortpflanzen, Liganden binden und Plasmide übertragen können. Die Simulationen werden unter verschiedenen Parametereinstellungen ausgeführt, um die Auswirkungen auf die Evolution und Kooperation zu analysieren. Die Ergebnisse zeigen, dass evolutionäre Vorgänge erfolgreich modelliert werden konnten. Kooperative Verhaltensmuster konnten jedoch unter den gezeigten Bedingungen nicht beobachtet werden.

Inhaltsverzeichnis

1 Einleitung

Charles Darwin veränderte mit seinem 1859 veröffentlichten Buch *On the Origin of Species* die Sicht auf die Entstehung und Entwicklung des Lebens wesentlich. Allein die biologische Evolution sollte für die diverse Entwicklung des Lebens auf der Erde verantwortlich sein. Bis zu seinem Tod im Jahre 1882 war die Evolutionstheorie in wissenschaftlichen Kreisen bereits weitgehend anerkannt und stellte die Vorstellung einer göttlichen Kreation zunehmend infrage [12]. Keine intelligente Schöpfer*in, vielmehr Zeit und Zufall bestimmen die Richtung der Entwicklung. Strukturen, Verhaltensweisen und sogar komplexe Strategien entstehen nicht durch Planung, sondern indem sich zufällige Variationen als vorteilhaft erweisen und dadurch über Generationen hinweg erhalten bleiben. Es ist daher kaum verwunderlich, dass die Informatik fast 100 Jahre später die natürliche Selektion als Grundlage für die Lösung diverser Probleme nahm. Diese sogenannten Evolutionären Algorithmen (EA) reichen von Stundenplanern in Schulen über industrielle Designverbesserungen bis hin zu Algorithmen für den Aktienhandel. Ein Hauptmerkmal evolutionärer Algorithmen ist, dass sie lediglich durch „Trial-and-Error“ entstehen. [5, S. 13–24]

In den 1960er Jahren verwendeten erste Wissenschaftler EA im biologischen Sinne. Damals lag in der evolutionären Programmierung (EP) die künstliche Intelligenz (KI) im Fokus [5, S. 103]. Der Schwerpunkt dieses Projekts in der EP liegt jedoch auf einer naturbezogenen Darstellung evolutionärer Prozesse. Deshalb wird in dieser Arbeit untersucht, inwiefern sich möglichst realistische Einzeller durch rein evolutionäre Entwicklung gewisse Merkmale aneignen können. Insbesondere stellt sich die Frage, ob sich in einem egoistisch wirkenden System kooperative Verhaltensmuster entwickeln können. Ausgehend davon lässt sich folgende Fragestellung

formulieren:

Kann eine biologische Evolution simuliert werden und entstehen dabei kooperative Verhaltensmuster unter simulierten Einzellern?

Dafür müssen die Einzeller stark vereinfacht werden. Um deutlich zu machen, in welchen Aspekten die anschliessende Evolutionssimulation von der natürlichen Biologie abweicht, werden im Abschnitt 3.1 sämtliche Eigenschaften der simulierten Einzeller definiert. Auf diese Weise werden alle Vereinfachungen und Abweichungen gegenüber den in Abschnitt 2.1 beschriebenen biologischen Merkmalen der realen Einzeller transparent und nachvollziehbar dargestellt.

2 Theorie

Im folgenden Kapitel wird die theoretische Grundlage dieser Arbeit erläutert. Zunächst werden die zellbiologischen Grundlagen am Prokaryoten *Escherichia coli* vorgestellt, bevor anschliessend zentrale Konzepte der Evolutionsbiologie behandelt werden.

2.1 Zellbiologische Grundlagen

Die wichtigsten Bestandteile und Eigenschaften einer prokaryotischen Zelle werden im Folgenden erläutert.

2.1.1 Genom

Das genetische Material (Genom) von Prokaryoten liegt in Form von einem ringförmigen Desoxyribonukleinsäure-(DNA)-Molekül vor, auch Bakterienchromosom genannt. Das Genom bestimmt alle Eigenschaften der Zelle und ist die Vorlage für alle gebildeten Proteine. Im Cytoplasma synthetisieren die Ribosomen die Proteine, darunter auch Rezeptorproteine (vgl. Abschnitt 2.1.4), nach der genetischen Information der DNA [18]. Die DNA ist eine Doppelhelix aus Nukleotiden. Jedes Nukleotid besteht aus einem Zucker (Desoxyribose), einem Phosphatrest und einer Base. Die vier Basen *Adenin*, *Thymin*, *Guanin* & *Cytosin* bilden ein quaternäres Speichersystem [4, S. 366–399].

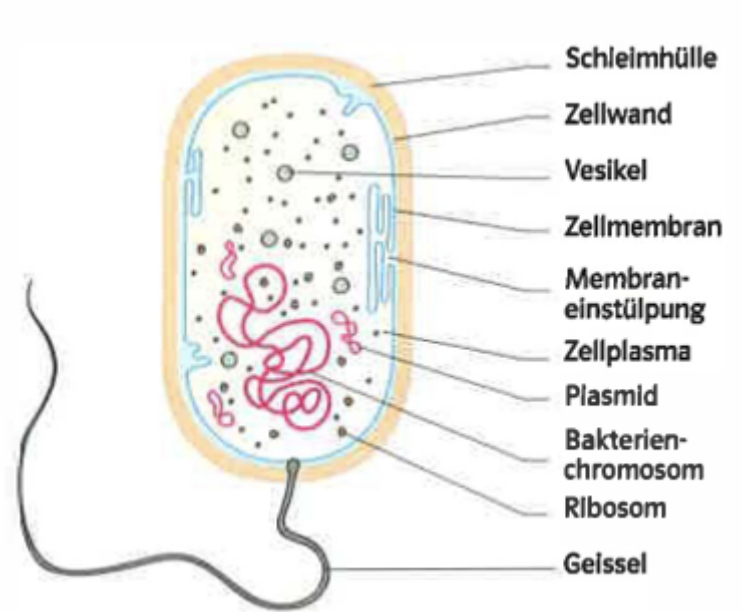

Abbildung 2.1: Schematische Darstellung eines Prokaryoten [4, S. 88–113]

Ausserdem besitzen manche Zellen weitere kleinere DNA-Stränge, sogenannte Plasmide, die frei im Zytoplasma (Zellplasma) schwimmen. Sie können zwischen verschiedenen Bakterien ausgetauscht werden. Oft befinden sich Resistenzgene zum Beispiel gegen Antibiotika auf Plasmiden. Da Plasmide sich nicht nur durch Fortpflanzung verbreiten, können sich schützende Gene viel schneller ausbreiten [18]. Die Übertragung erfolgt über sogenannte Plasmabrücken zwischen zwei Bakterien. Nur das gebende Bakterium muss eine Brücke aufbauen, das empfangende Bakterium nimmt das Plasmid passiv auf [17]. Dies wird horizontaler Gentransfer genannt, da Gene nicht nur von Eltern auf Nachkommen weitergegeben werden, sondern auch zwischen Individuen derselben Generation [9].

2.1.2 Fortpflanzung

Prokaryotische Zellen pflanzen sich vor allem mit Querteilung fort. Die Querteilung ist eine Form der ungeschlechtlichen Fortpflanzung, bei der sich das Bakterium senkrecht zur Längsachse in zwei Tochterzellen teilt. Nachdem das Bakterienchromosom geteilt wurde, bildet sich eine teilende Zellmembran. Beide Nachkommen besitzen nun ein beinahe identisches

Genom (vgl. 2.1.3). Unter günstigen Bedingungen kann sich das Bakterium *E. coli* innerhalb von 20 Minuten teilen [4, S. 88–113]. Die Zelle vergrössert ihr Volumen bis die kritische Grösse für die Zellteilung erreicht ist. Anschliessend wird die DNA repliziert, bevor die Zelle sich in der Zellteilung in zwei Tochterzellen aufteilt [15]. Diese Art von Gentransfer wird als vertikale bezeichnet, da die Gene von Eltern auf Nachkommen weitergegeben werden [9].

Die Bevölkerungszahl einer Bakterienkultur wächst exponentiell, solange genügend Nährstoffe und Platz vorhanden sind. Die exponentielle Wachstumsphase endet, wenn die Nährstoffe knapp werden oder Abfallprodukte die Umwelt belasten [4, S. 88–113].

2.1.3 Mutation

Veränderungen des genetischen Materials (Genom) nennt man Mutationen. Sie können spontan auftreten oder durch äussere Einflüsse wie Strahlung oder Chemikalien ausgelöst werden. Oft entstehen Mutationen während der DNA-Replikation vor der Zellteilung [4, S. 366–399]. Mutationen haben einen direkten Einfluss auf den Genotyp, also die genetische Information eines Organismus. Diese Veränderungen müssen aber nicht zwangsläufig Auswirkungen auf den Phänotyp, also das äussere Erscheinungsbild oder die beobachtbaren Merkmale, haben. Die genetische Variation, die durch neutrale Mutationen entsteht, kann bei sich verändernden Umweltbedingungen von Vorteil sein (vgl. Abschnitt 2.2.1) [14] [13, S. 50–61].

Die Mutationsrate gibt an, wie häufig Mutationen in einem bestimmten Zeitraum oder pro Generation auftreten. Sie variiert stark zwischen verschiedenen Organismen und kann durch Umweltfaktoren beeinflusst werden. Den Angaben von Lee u. a. [10] zufolge ist die Mutationsrate bei *E. coli* 2.2×10^{-10} pro Nukleotid pro Generation und 1.0×10^{-3} pro Genom pro Generation.

2.1.4 Chemotaxis und Nahrung

Chemotaxis bezeichnet eine durch chemische Reize ausgelöste Bewegung [6, S. 7]. Prokaryotische Zellen können chemische Stoffe in ihrer Umgebung mittels Rezeptoren wahrnehmen. Rezeptoren funktionieren nach dem Schlüssel-Schloss-Prinzip. Das heisst, jeder Rezeptor kann nur mit einem bestimmten Signalmolekül (Ligand) binden. Ihre Struktur ist modular aufgebaut und lässt sich in drei Bereiche unterteilen: Eine extrazelluläre Domäne, die für die Erkennung und Bindung der Liganden zuständig ist, die Transmembrandomäne, die den Rezeptor in der Membran verankert, sowie eine intrazelluläre Domäne, die für die Signalweiterleitung zuständig ist. Bindet ein Ligand, löst der Rezeptor auf der intrazellulären Seite eine Signalkaskade aus, deren Endprodukte *Second Messenger* (SM), auch Botenstoffe genannt, sind. Diese SM lösen in der Zelle verschiedene Reaktionen aus, unter anderem die Veränderung der Zellbewegung. Diese Modularität ermöglicht, dass Rezeptoren mit ähnlichen extrazellulären Bereichen dieselben Liganden erkennen können, während unterschiedliche intrazelluläre Domänen zu verschiedenen zellulären Antworten führen. [20] [8] [6, S. 113–125]

Prokaryoten können sich aktiv fortbewegen. Dafür besitzen sie mehrere Flagellen (Geisseln), die Rotationsbewegungen ausführen können. Die passive Bewegung folgt den Newtonschen Gesetzen. Gemäss Berg und Brown [2] bewegt sich der Einzeller *E. coli* nach dem *Run & Tumble* Prinzip. Drehen alle Flagellen in dieselbe Richtung, bewegt sich das Bakterium geradlinig und man spricht von einer *Run*-Bewegung. Weicht eine oder mehrere Flagellen davon ab, beginnt die Zelle mit der *Tumble*-Bewegung. Die Zelle beginnt, ihre Richtung scheinbar willkürlich zu ändern. Die Konzentrationsänderung des zuständigen SM ist direkt für die Häufigkeit der *Tumble*-Bewegung verantwortlich. Mithilfe von Chemotaxis kann das Bakterium so seine Richtung aktiv lenken. [6, S. 53–89]

Gewisse Einzeller können eigene Liganden synthetisieren und abgeben, um so mit anderen Zellen zu kommunizieren. So können kooperative Verhaltensmuster entstehen [22]. Prokaryoten können sich unterschiedlich ernähren. *E. coli* ernährt sich hauptsächlich von Zucker und Aminosäuren [21] [4, S. 88–113].

2.2 Populationsgenetik und Evolution

2.2.1 Evolutionstheorie

Die Evolution beschreibt die Veränderung der genetischen Zusammensetzung einer Population über Generationen hinweg. Sie ist kein zielgerichteter Prozess, sondern ein Nebeneffekt aus der Variation innerhalb einer Population und der Selektion durch die Umwelt. Evolution beschränkt sich nicht nur auf biologische Systeme, sondern kann auch in anderen Bereichen wie der Informatik angewendet werden, wie im Kapitel 1 gezeigt wurde [5, S. 13–24].

Folgende vier Kriterien müssen erfüllt sein, damit Evolution stattfinden kann:

- **Variation:** Innerhalb einer Population müssen Unterschiede (Variationen) im Phänotyp der Individuen vorhanden sein.
- **Vererbung:** Die Merkmale müssen von den Eltern auf die Nachkommen vererbbar sein.
- **Selektion:** Es muss einen Selektionsdruck geben, der bestimmt, welche Merkmale vorteilhaft sind. Individuen mit vorteilhaften Merkmalen haben eine höhere Wahrscheinlichkeit zu überleben und sich fortzupflanzen.
- **Zeit:** Evolutionäre Veränderungen benötigen Zeit, da sie über viele Generationen hinweg stattfinden.

[13, S. 1–9]

2.2.2 Populationsgenetik

Um die Evolution innerhalb einer Population zu verstehen, muss man die genetische Zusammensetzung der Population betrachten. Allele sind verschiedene Varianten eines Gens, die für unterschiedliche Merkmale kodieren. [13, S. 62–90]

Der Flaschenhalseffekt beschreibt eine drastische Reduktion der Populationsgrösse durch ein plötzliches Ereignis, wie eine Naturkatastrophe. Bei kleiner Variation innerhalb der Population kann dies sogar zum Aussterben der Population führen. Die genetische Vielfalt wird durch den Flaschenhalseffekt stark reduziert. Die Überlebenden bilden die Grundlage für die zukünftige Population, was zu veränderten dominanten Merkmalen führen kann [13, S. 260–286] [4, S. 450–469].

3 Methodik

3.1 Design

Um die Fragestellungen dieser Arbeit zu beantworten, wird eine Evolutionssimulation entwickelt. Zuerst wird die Simulation berechnet und kann dann vom Benutzer nachträglich analysiert und visualisiert werden. Der folgende Link führt zu einem Video der Simulation: `https://youtu.be/aYznFBBuo7w`. Der Quellcode der Simulation ist auf GitHub verfügbar: `https://github.com/Juli_3200/UniSim`.

In der Simulationsumgebung existieren Einzeller und Nahrung. Die Einzeller, fortan auch *Entitäten* genannt, sind die zentralen Akteure der Simulation. Das Ziel der Arbeit ist es, diese Einzeller möglichst realitätsnahe zu modellieren. Es gilt, einen Kompromiss zwischen der angestrebten biologischen Genauigkeit und einer begrenzten Komplexität der Simulation zu erzielen. Es müssen daher diverse Vereinfachungen vorgenommen werden. Die Unterschiede und Gemeinsamkeiten aller verwendeten Eigenschaften der Einzeller und Nahrung in der Simulation und der reellen Biologie werden in den folgenden Abschnitten erläutert.

Alle mit * markierten Werte können in den Simulationseinstellungen angepasst werden (vgl. Tabellen 3.2, 3.1, 3.4 und 3.3).

3.1.1 Aufbau der Simulation

Die Simulation findet in einer zweidimensionalen, rechteckigen Umgebung statt. Diese Umgebung ist von Wänden begrenzt, an denen Entitäten und Liganden reflektieren. Die Grösse* des Rechtecks ist standardmässig auf 100x100 Einheiten gesetzt. Die Entitäten und Liganden bewegen sich in dieser Umgebung entsprechend den newtonschen Gesetzen. Das Medium der Umgebung gleicht einem Fluid, um den Einzeller-Lebensraum Wasser zu simulieren [4, S. 88–113]. Auf die Entitäten wirken eine Richtungskraft* und ein Fluidwiderstand*. Die Richtungskraft der Simulation modelliert die Schwerkraft oder eine andere konstante Kraft, wie z.B. einen Strömungseinfluss. Diese Richtungskraft kann in alle Richtungen wirken. Der Fluidwiderstand ist dem Wasser- oder Luftwiderstand nachempfunden und wirkt der Bewegungsrichtung der Entität entgegen. Er sorgt dafür, dass Entitäten nicht unendlich beschleunigen können. Dieser Widerstand berechnet sich nach der Formel:

$$F_{Widerstand} = \frac{1}{2}\rho v^2 C A$$

wobei C der Widerstandskoeffizient*, v die Geschwindigkeit der Entität und ρ die Dichte des Mediums ist. In der Simulation beträgt ρ stets den Wert 1. A bezeichnet die Querschnittsfläche der Entität. Da die Simulation zweidimensional ist, reduziert sich die Querschnittsfläche auf eine Länge. Diese Querschnittslänge entspricht der Höhe der Entität. Da die Entitäten als Kreise modelliert sind, wird A als halber Kreisumfang berechnet.

3.1.2 Nahrung

In der Simulation werden Signalmoleküle (Liganden) und Nährstoffe zusammengenommen als Liganden bezeichnet. Liganden bewegen sich passiv durch die Simulationsumgebung. In der Simulation besitzen sie keine Masse und keine Ausdehnung, sie verhalten sich also ähnlich wie Photonen. Kollisionen zwischen Liganden werden in der Simulation nicht berücksichtigt. Kollisionen mit der Weltgrenze führen zu einer Reflexion des Liganden. Berührt ein Ligand

eine Entität, wird er von dieser aufgenommen (vgl. Abschnitt 3.1.4) oder an der Tangente der Berührungsstelle reflektiert.

Schadstoffe wie zum Beispiel Antibiotika oder andere Gifte werden in der Simulation ebenfalls als Liganden modelliert [4, S. 306–325]. Sie können in den Simulationseinstellungen deaktiviert werden. Schadstoffe haben eine toxische Wirkung auf die Entitäten, indem sie die Energie der Entität verringern, wenn sie aufgenommen werden. Anders als andere Liganden, müssen Schadstoffe nicht von einem Rezeptor erkannt werden, um aufgenommen zu werden (vgl. Abschnitt 3.1.4). Falls eine Entität mit einem Schadstoff kollidiert, wird dieser automatisch aufgenommen und die toxische Wirkung tritt ein. Die in Abschnitt 3.1.5 beschriebenen Plasmide können die toxische Wirkung von Liganden aufheben.

Die Anzahl* S_{max} Ligandentypen ist im Gegensatz zur Biologie limitiert, um die Bindungswahrscheinlichkeit zwischen Liganden und Rezeptoren zu erhöhen [4, S. 207]. Ist S_{max} zu hoch, besteht die Gefahr, dass kein Ligandentyp von einem Rezeptor erkannt wird, was zu einem Hungertod aller Entitäten führt. S_{max} sollte als Zweierpotenz gewählt werden, um die folgende Energiefunktion zu vereinfachen. Jeder Ligandentyp besitzt eine *Spezifikationszahl* S, welche die Proteinstruktur des Liganden widerspiegelt. Diese Spezifikationszahl wird verwendet, um die Bindung zwischen Liganden und Rezeptoren der Entitäten zu bestimmen (vgl. Abschnitt 3.1.4). Ausserdem hängt die Nährstoffmenge (Energie), die eine Entität durch die Aufnahme eines Liganden gewinnt, jedes Ligandentyps von der Spezifikationszahl ab. Die Energie eines Liganden wird mit der Funktion

$$el(S) = E_{min} + \frac{wt(S)}{\log_2(S_{max})} (E_{max} - E_{min})$$

berechnet, wobei $wt(S)$ das Hamming-Gewicht von S, daher die Anzahl der Einsen in seiner Binärdarstellung, ist. E^*_{max} und E^*_{min} sind einstellbare Parameter, die die minimale und maximale Energie eines Liganden festlegen. In der Biologie hängt die Energie eines Moleküls von dessen chemischer Struktur ab.

Wenn Schadstoffe in der Simulation aktiviert sind, besitzen alle Liganden mit einer ungeraden

Spezifikationszahl eine toxische Wirkung. Die Energiefunktion in diesem Fall ist definiert als:

$$el_{tox}(S) = (-1)^S \cdot \frac{el(S)}{2}$$

Liganden werden von sogenannten *Ligandenquellen* erzeugt. Ligandenquellen können vom Benutzer an beliebigen Positionen in der Simulationsumgebung platziert werden. Sie besitzen eine *Erzeugungsrate*, welche bestimmt, wie viele Liganden pro Zeiteinheit erzeugt werden. Ausserdem kann für jede Quelle die Spezifikationszahl des Ligandentyps festgelegt werden. Eine Entität kann Liganden synthetisieren, um diese an andere Entitäten zu senden. Sie benötigen dafür den Betrag der Energie des Liganden.

3.1.3 Physikalische Eigenschaften der Entitäten

Der Einzeller *E. coli* ist in seiner natürlichen Form zylindrisch [21]. Um eine einfachere Kollisionserkennung zu ermöglichen, wurden die Entitäten in der Simulation als kreisförmig modelliert. Die Kollisionen werden als elastische Kollisionen zwischen Kreisen behandelt. Energieverluste bei Kollisionen, wie sie in der Realität auftreten, wie z.B. durch Verformung der Zellmembran, werden in der Simulation nicht berücksichtigt. Eine Kollision ohne Energieverlust wird elastische Kollision genannt.

Die elastischen Kollisionen wurden nach den in Bereck [1] und Work [23] beschriebenen Gleichungen implementiert. In der folgenden Gleichung ist $\hat{\vec{v}}_1$ die Geschwindigkeit der ersten Entität nach der Kollision, $\vec{v}_1$ und $\vec{v}_2$ die Geschwindigkeitsvektoren der beiden Entitäten vor der Kollision, m_1 und m_2 deren Massen und C_1 und C_2 deren Positionen:

$$\hat{\vec{v}}_1 = \vec{v_1} - \frac{2m_2}{m_1 + m_2} \frac{(v_1 - v_2) \cdot (C_1 - C_2)}{\|C_1 - C_2\|^2} (C_1 - C_2)$$

Die Gleichung für die Geschwindigkeit $\hat{\vec{v}}_2$ der zweiten Entität nach der Kollision ist analog

dazu definiert.

Die innere Energie beziehungsweise Masse einer Entität ergibt sich nach der Beziehung $E_U = m = \pi r^2$, wobei r dem Radius der Entität entspricht. Sie entsprechen somit der Fläche des Kreises, der die Entität in der Simulation darstellt. Die newtonschen Gesetze gelten in der Simulation, weshalb Entitäten mit grösserer Masse (Energie) träger auf Kräfte reagieren als solche mit kleinerer Masse. Entitäten verbrauchen Energie für Bewegung, Stoffwechsel und die Ligandensynthese. Der Energieverbrauch für den Stoffwechsel ist proportional* zur Masse der Entität, während der Energieverbrauch für die Bewegung konstant* ist. Die Ligandensynthese zieht der Entität Energie in Abhängigkeit der Anzahl und Energie der erstellten Liganden ab. Die Standardwerte für den Bewegungsenergieverbrauch sind höher gesetzt als in der Realität, ausserdem unterscheidet sich der Energieverbrauch* zwischen den Bewegungsarten (vgl. Abschnitt 3.1.4) *Run & Tumble*. Dies ermöglicht eine stärkere Selektion nach verschiedenen Bewegungsstrategien.

Sobald die Energie einer Entität null erreicht, stirbt sie und wird aus der Simulation entfernt. Erreicht sie aber eine bestimmte Energiemenge*, teilt sie sich in zwei Tochterentitäten auf. Dabei wird die Energie der Mutterentität gleichmässig auf die beiden Tochterentitäten aufgeteilt.

3.1.4 Chemotaxis

Entitäten besitzen Rezeptoren auf ihrer Zellmembran, die Liganden in der Umgebung erkennen können. Anders als beim Einzeller *E. coli*, verteilen sich diese Rezeptoren nach einer in Abschnitt 3.1.5 beschriebenen Funktion über die gesamte Zellmembran der Entität. Diese Funktion wird bei π gespiegelt, um diese stetig über den gesamten Kreis zu machen. Um das Schlüssel-Schloss-Prinzip der Biologie zu modellieren, besitzt jeder Rezeptor eine Spezifikationszahl (vgl. Abschnitt 3.1.2). Wenn ein Ligand einen Rezeptor berührt, kann eine Bindung stattfinden, wenn die Spezifikationszahlen von Ligand und Rezeptor gleich sind. In der Simulation kann für jeden Ligandentypen nur ein Rezeptortyp existieren. Da Nahrung und Signalmoleküle in der Simulation als Liganden zusammengefasst sind, sind Rezeptoren

auch für die Nahrungsaufnahme verantwortlich. Bindet ein Ligand an einen Rezeptor, wird die Energie des Liganden der inneren Energie der Entität hinzugefügt. So lohnt es sich für eine Entität möglichst viele verschiedene Rezeptortypen zu besitzen, um eine grössere Vielfalt an Liganden aufnehmen zu können. Die Simulation limitiert die maximale Anzahl* verschiedener Rezeptortypen pro Entität.

Wenn ein Ligand mit einer Entität kollidiert, wird der Kollisionswinkel berechnet. Anhand dieses Winkels wird bestimmt, mit welchem Rezeptor der Ligand interagiert. Der Kollisionswinkel liegt im Intervall $[0, \pi]$, da die Rezeptoren symmetrisch auf der Zellmembran verteilt sind. Der Kollisionswinkel berechnet sich mit C_L als Kollisionspunkt, C_E als Zentrum und $\vec{v}_E$ als Geschwindigkeitsvektor der Entität:

$$\theta = \arccos\left(\frac{(C_L - C_E) \cdot \vec{v}_E}{\|C_L - C_E\| \cdot \|\vec{v}_E\|}\right)$$

Der Rezeptor, der am Winkel θ auf der Zellmembran liegt, wird als interagierender Rezeptor ausgewählt. In der Simulation wird die in Abschnitt 2.1.4 beschriebene Aktivierung eines Rezeptors vereinfacht dargestellt. Bindet ein Ligand an einen Rezeptor, schüttet der Rezeptor direkt einen *Second Messenger* (SM) in die Zelle aus. Dessen Art hängt vom Gen des Rezeptors ab. So können Rezeptoren des Typs A, die nur Liganden des Typs A erkennen, unterschiedliche Botenstoffe ausschütten. Dieses Prinzip ist dem modularen Aufbau prokaryotischer Rezeptoren nachempfunden [6, S. 113–125].

Die Zahl der SM ist konstant, da für jeden eine eigene Aktion implementiert werden muss. Sie kann jedoch durch eine Programmkonstante verändert werden, um das Programm skalierbar zu halten.

Der Botenstoff ist entweder negativ oder positiv, und beeinflusst so die Konzentration innerhalb der Zelle. Die Konzentration im Zellinneren wird für Bewegung, Ligandensynthese und Plasmidübertragung verwendet. Überschreitet die Konzentration des Botenstoffs A einen vom Genom festgelegten Schwellenwert, wechselt die Entität in den *Tumble*-Zustand. Sinkt die Kon-

zentration wieder unter den Schwellenwert, wechselt die Entität zurück in den *Run*-Zustand. Im *Run*-Zustand bewegt sich die Entität in eine feste Richtung mit konstanter Beschleunigung*. Im *Tumble*-Zustand ändert die Entität in jedem Update mit der Wahrscheinlichkeit* p_{Tumble} ihre Bewegungsrichtung zufällig mit einem Winkel im Intervall $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Anders als in der Biologie wird in der Simulation mit absoluten Konzentrationswerten gearbeitet und nicht mit Konzentrationsänderungen über die Zeit [2].

3.1.5 Genom der Entitäten

Jede Entität besitzt ein Genom, das aus mehreren Genen besteht. Die Gene liegen in der Simulation als Bitstrings und Zahlen vor und unterscheiden sich somit von der Doppelhelixstruktur der DNA in der Biologie 2.1.1. Die quaternäre DNA-Struktur wird durch ein Binärsystem ersetzt. In der Simulation sind die Gene direkt mit einem Phänotyp verbunden. In der Biologie wird heute nicht mehr davon ausgegangen, dass ein einzelnes Gen allein für einen bestimmten Phänotyp verantwortlich ist. Vielmehr entstehen die meisten Merkmale durch das komplexe Zusammenspiel mehrerer Gene [13, S. 50–62]. Beim Einzeller *E. coli* liegen die Gene in Form des Bakterienchromosoms und der Plasmide vor (vgl. Abschnitt 2.1.1). Das Genom der Entitäten wird in folgende Bestandteile unterteilt:

- **Schwellenwerte**: Jede Entität besitzt vier Gene, die für die Bewegung, die Ligandensynthese und den Austausch von Plasmiden zuständig sind. Das erste Gen legt den Schwellenwert für den SM A fest, der die Bewegung der Entität steuert. Das zweite Gen legt fest, ob die Ligandensynthese aktiviert ist, wenn der SM B einen bestimmten Schwellenwert überschreitet. Gen Nummer drei bestimmt den Schwellenwert für den SM C, der den Ligandentyp festlegt, den die Entität synthetisiert. Zuletzt legt das vierte Gen den Schwellenwert für den SM D fest, der die Bildung einer Plasmabrücke steuert.

- **Rezeptoren**: Jeder Rezeptortyp wird durch ein Gen kodiert. Das Gen bestimmt die Spezifikationszahl des Rezeptors, dessen Botenstoff und deren Wirkung (Positiv/Negativ). Ausserdem legt das Gen fest mit welcher Wahrscheinlichkeitsfunktion $P(x)$ die Rezep-

toren auf der Zellmembran verteilt sind. x ist dabei der Winkel auf der Zellmembran im Intervall $[0, \pi]$. Die Funktion ist definiert als:

$$f(x) = \begin{cases} 0, & ax^2 + bx + c < 0, \\ ax^2 + bx + c, & 0 \leq ax^2 + bx + c \leq 1, \\ 1, & ax^2 + bx + c > 1. \end{cases}$$

Dabei sind a, b und c Genparameter, die im Intervall $[-\frac{1}{6}, \frac{1}{3}]$ liegen. c ist im Intervall $[0, \frac{1}{2}]$. Die Funktion hat den Wertebereich $[-\frac{1}{3}, \frac{5}{3}]$, weshalb sie für negative Werte 0 und für Werte über 1 auf 1 gesetzt wird. Bei der Initialisierung einer Entität wird für jeden Rezeptortyp anhand der Funktion $P(x)$ bestimmt, ob an der Position x auf der Zellmembran ein Rezeptor des gleichen Typs vorhanden ist. Bei der Initialisierung der Entität werden die Rezeptoren einmalig über die Zellmembran verteilt. Die Anzahl* der Rezeptorgene pro Entität, sowie die gesamte Anzahl* Rezeptoren auf der Zellmembran sind limitiert.

- **Ligandensynthese**: Jeder Ligandentyp, den eine Entität synthetisieren kann, wird durch ein Gen kodiert. Die n^* Gene legen die Spezifikationszahl der synthetisierten Liganden fest. Wird ein Ligand synthetisiert, wird die Energie der Entität um die Energie des Liganden reduziert. Der Ligand wird in eine zufällige Richtung von der Entität aus mit der Geschwindigkeit der Entität abgefeuert. Der *Second Messenger* B entscheidet, ob die Entität Liganden synthetisiert. Der *Second Messenger* C legt fest, welchen Ligandentyp die Entität synthetisiert.

- **Plasmide**: Plasmide sind in der Simulation immer Resistenzgene gegen toxische Liganden. Sie werden durch eine Zahl kodiert, die angibt, gegen welche Ligandentypen die Entität resistent ist.

Bei der Geburt einer Entität werden die Gene der Eltern entnommen und mit einer Wahrscheinlichkeit* mutiert. Falls keine Eltern vorhanden sind (Initialisierung der Simulation), werden die Gene zufällig generiert. Die Schwellenwerte werden mit einer Normalverteilung

$X \sim \mathcal{N}(\mu, \sigma)$ generiert, wobei μ und σ in den Simulationseinstellungen angepasst werden können. Die Mutation erfolgt für fast alle Gene mit der Wahrscheinlichkeit* p_{Mut} pro Bit. Dies entspricht dem typischen *bit-flip* Verfahren der genetischen Algorithmen [5, S. 99–116]. Die Schwellenwert-Gene mutieren nicht mit dem *bit-flip* Verfahren, sondern werden durch Addition eines zufälligen, gerundeten Wertes aus einer Normalverteilung $X \sim \mathcal{N}(0, \sigma*_{Mutation})$ verändert. Der neue Wert wird anschliessend auf den zulässigen Bereich des erlaubten Schwellenwertintervalls* begrenzt.

3.2 Implementierung

Die Simulation `UniSim` wurde in den Programmiersprachen Rust und Cuda C++ implementiert. Die Hauptlogik wurde in Rust geschrieben. Rust ist eine moderne Programmiersprache, die sich durch hohe Performance und Speicher-Sicherheit auszeichnet. Cuda C++ wurde verwendet, um die Bewegungen und Kollisionen der Liganden auf der Grafikkarte (GPU) zu berechnen. Das Kernprogramm lässt sich in drei Teile einordnen:

- **Physik-Engine**: Berechnung der Bewegungen und Kollisionen der Liganden und Entitäten.
- **Biologischer Teil**: Umsetzung der biologischen Mechanismen wie Chemotaxis, Genom und Energiehaushalt.
- **Parallelisierung**: Nutzung der GPU zur Beschleunigung der Physik-Berechnungen.

Alle Analysen, Visualisierungen wurden mit Python umgesetzt, da hierfür eine grosse Auswahl an Bibliotheken zur Verfügung steht. Diese Skripte wurden teilweise von ChatGPT generiert und angepasst. Sie dienen ausschliesslich der Auswertung und Visualisierung und sind nicht Bestandteil der bewerteten Eigenleistung. Das Programm wurde möglichst modifizierbar gestaltet. Im Abschnitt 3.2.5 werden die wichtigsten Simulationsparameter beschrieben.

3.2.1 Grundstruktur und Simulationsupdate

Rust ist keine traditionell objektorientierte Programmiersprache, sondern verwendet Strukturen `Structs` und `Traits`, um Daten und Funktionen zu organisieren [16]. `Structs` gleichen C Strukturen, können jedoch auch Methoden besitzen. Das Programm enthält folgende zentrale Strukturen:

- **World**: `World` ist die Hauptstruktur der Simulation. Sie verwaltet alle anderen Strukturen und berechnet die Simulation. Ausserdem ist sie die Schnittstelle zum Benutzer.
- **Space**: `Space` verwaltet vor allem Kollisionen zwischen Entitäten.
- **Settings**: `Settings` enthält alle Simulationsparameter, die in Abschnitt 3.2.5 beschrieben werden.
- **Cudaworld**: `Cudaworld` verwaltet die GPU-Ressourcen und die Kommunikation zwischen Hauptprozessor (CPU) und GPU.
- **Entity**: `Entity` enthält biologische und physikalische Daten einer Entität. Sie implementiert Methoden zur Bewegung, Chemotaxis und Genom.
- **Ligand**: Wenn die GPU-Beschleunigung deaktiviert ist, werden die Liganden in dieser Struktur gespeichert und aktualisiert.
- **LigandSource**: `LigandSource` entspricht den in Abschnitt 3.1.2 beschriebenen Ligandenquellen.

Neben den beschriebenen Komponenten existieren zusätzliche Hilfsstrukturen, die hier nicht im Detail behandelt werden.

In der `update`-Methode der `World`-Struktur wird die Simulation n^* Mal pro Sekunde aktualisiert. Jedes Update, auch `State` genannt, durchläuft den folgenden Ablauf. Zu Beginn werden alle neuen Liganden der Simulation hinzugefügt. Anschliessend werden alle Entitäten

aktualisiert, indem ihre `update_physics`-Methode aufgerufen wird. In dieser Methode werden die Bewegungen und Beschleunigungen der Entitäten berechnet. Kollisionen der Entitäten werden mit der `Space` Struktur und der Methode `resolve_collision` berechnet. Es folgen die Positions-Updates der Liganden und schliesslich die Kollisionen der Liganden mit den Entitäten (3.2.2). Nachdem neue Liganden von den Entitäten verarbeitet wurden, werden in der `update_output` Methode die biologischen Mechanismen der Entitäten aktualisiert, wie z.B. Chemotaxis und Ligandensynthese (vgl. Abschnitt 3.2.3). Schlussendlich werden tote Entitäten entfernt und neue Entitäten hinzugefügt. Der State wird serialisiert und in einem Vektor zwischengespeichert. Nach m^* Updates wird der Vektor in eine Binärdatei übertragen.

Um die Welt zu speichern, wurde der Trait `Serialize` entwickelt. Der Trait ermöglicht die Serialisierung aller Weltdaten in eine Binärdatei. Dem Header der Datei folgt eine sogenannte *Jumper-Table*, die die Positionen der verschiedenen Datenblöcke in der Datei speichert. Die Datenblöcke enthalten für jeden State Informationen über die Entitäten und Metadaten der Simulation. So können States gezielt geladen werden, ohne die gesamte Datei einlesen zu müssen.

3.2.2 Physik Engine

Für die Berechnung der Bewegungen und Kollisionen der Liganden und Entitäten wurde eine eigene Physik Engine entwickelt. Eine Physik Engine umfasst alle physikalischen Berechnungen einer Simulation, wie z.B. Bewegungen, Kräfte und Kollisionen. In den Abschnitten 3.1.3 und 3.1.1 wurden die physikalischen Eigenschaften der Entitäten und der Simulationsumgebung bereits beschrieben.

In der Simulation wird die Kollisionserkennung in der `Space` Struktur verwaltet. Die naive Methode, bei der jede Entität mit jedem Liganden auf Kollision überprüft wird, hat eine Zeitkomplexität von $O(n^2)$, wobei n die Anzahl der Entitäten und Liganden ist. Um die Performance zu verbessern, wurde eine *grid-based collision detection* implementiert. Dabei werden alle Entitäten in ein Koordinatengitter der `Space` Struktur eingeteilt. Jede Gitterzelle

enthält eine Liste der Entitäten, die sich in dieser Zelle befinden. So müssen nur noch Entitäten in der gleichen oder benachbarten Gitterzellen auf Kollisionen geprüft werden. So reduziert sich die Zeitkomplexität der Kollisionsabfrage auf $O(kr^2)$, wobei k die durchschnittliche Anzahl der Entitäten in den benachbarten Gitterzellen ist und r den Suchradius im Gitter bezeichnet. Der Suchradius entspricht dem Durchmesser der grössten Entität in der Simulation.

Das Gitter enthält für jede Gitterzelle eine Liste der Indizes der Entitäten, die sich in dieser Zelle befinden. Mit der `check_position` Methode kann jede Position auf Kollisionen mit Entitäten und der Weltgrenze überprüft werden. Falls Entität A eine Kollision mit Entität B feststellt, gibt die Methode eine Kopie aller notwendigen Kenndaten von Entität B zurück. Entität A kann dann die Kollision mit den in Abschnitt 3.1.3 beschriebenen Formeln lösen. Entität A muss nur seinen eigenen Geschwindigkeitsvektor $\hat{\vec{v}}_A$ berechnen, da Entität B die Kollision ebenfalls erkannt hat und sie selbst verarbeitet.

3.2.3 Biologischer Teil

Alle Liganden, die mit einer Entität kollidieren, werden von der `receive_ligand` Methode der `Entity` Struktur aufgenommen. In dieser Methode wird der Kollisionswinkel berechnet und der interagierende Rezeptor ausgewählt (vgl. Abschnitt 3.1.4). Kommt es zu einer Bindung, so wirken die in Abschnitt 3.1.2 beschriebenen Mechanismen. Die Konzentration der SM wird in einem einstellbaren Intervall gehalten. Falls keine Bindung stattfindet, wird der Ligand an der Tangente der Berührungsstelle reflektiert. Falls die GPU-Beschleunigung (vgl. Abschnitt 3.2.4) aktiviert ist, wird die Bindung bereits auf der GPU geprüft (vgl. Abschnitt 3.2.4).

In der `update_output` Methode der `Entity` Struktur werden die Konzentrationen der Botenstoffe mit den zugehörigen Schwellenwerten verglichen. Jeder der in Abschnitt 3.1.5 beschriebenen Outputs (Bewegung, Ligandensynthese & Plasmidübertragung) wird entsprechend aktualisiert.

Bei einer Kollision zwischen zwei Entitäten wird geprüft, ob eine Plasmabrücke gebildet

werden kann. Ist dies der Fall, übergibt die aktive Entität das älteste Plasmid an die andere Entität. Falls die maximale Anzahl* Plasmide erreicht ist, wird das älteste Plasmid entfernt und durch das neue ersetzt.

3.2.4 Parallelisierung

Um die Performance der Simulation zu verbessern, werden alle Bewegungen und Kollisionen der Liganden auf der Grafikkarte berechnet. Kollisionen zwischen Entitäten werden auf der CPU berechnet, da ihre Anzahl vergleichsweise gering ist. Die GPU-Beschleunigung funktioniert nur mit einer kompatiblen NVIDIA-Grafikkarte, die Cuda unterstützt. Das Programm wurde so implementiert, dass es auch ohne GPU-Beschleunigung lauffähig ist. Die Parallelisierung beschränkt sich nicht nur auf die Grafikkarte, sondern nutzt auch die Mehrkernprozessoren moderner CPUs. Mit dem `rayon`-Crate werden Schleifen, ohne direkte Datenabhängigkeiten, parallelisiert. [19]

Die Grafikkarte besitzt tausende Kerne, die parallel arbeiten können. Besonders geeignet für diese Kerne sind *Floating point operations*, also Gleitkommaberechnungen, wie sie für eine Physik-Engine benötigt werden. Da sich jeder Ligand unabhängig von den anderen Liganden bewegen und sie nicht untereinander kollidieren, können die Bewegungen und Kollisionen der Liganden parallel auf der GPU berechnet werden. Insbesondere bei grossen unabhängigen Datenmengen, wie z.B. zehntausenden Liganden, wird die GPU-Beschleunigung sehr effizient. [7] [11]

Die Messungen in Abbildung 3.1 wurden mit einer NVIDIA GeForce RTX 3060 Grafikkarte und 16 AMD Ryzen 7 6800HS Prozessor durchgeführt. Auf der X-Achse ist die Anzahl der ausgeschütteten Liganden pro Sekunde in der Simulation aufgetragen. Die Y-Achse zeigt die durchschnittliche Zeit für 1000 Updates der Simulation in Millisekunden.

Wenn die GPU-Beschleunigung aktiviert ist, wird die Simulation in der `update_gpu`-Methode aktualisiert. Diese Methode gleicht der in Abschnitt 3.2.1 beschriebenen `update`-Methode. Der

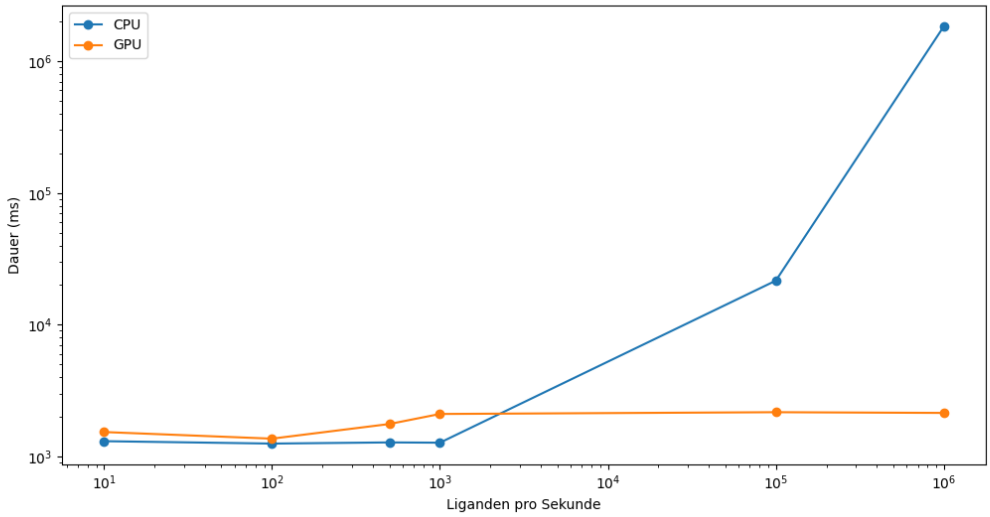

Abbildung 3.1: Performancevergleich der Simulation mit und ohne GPU-Beschleunigung

Unterschied besteht darin, dass die Bewegungen und Kollisionen der Liganden auf der GPU berechnet werden. Die Ligandendaten liegen in diesem Fall nur auf dem VRAM, also dem Arbeitsspeicher der Grafikkarte, vor. Nachdem alle Bewegungen und Kollisionen der Entitäten auf der CPU berechnet wurden, werden die aktualisierten Entitätsdaten mit den neu entstandenen Liganden an die GPU gesendet. Die Entitäten werden dann in ein Koordinatengitter eingeteilt. So kann eine *grid-based collision detection* durchgeführt werden (vgl. Abschnitt 3.2.2). Wird eine Kollision festgestellt, berechnet der GPU-Kernel den Kollisionswinkel und prüft, ob eine Bindung zwischen Ligand und dem dortigen Rezeptor stattfinden kann (vgl. Abschnitt 3.1.4). Wenn der Rezeptor den Liganden erkennt, wird der Ligand zurück an die CPU gesendet, wo er von der Entität aufgenommen wird. Falls keine Bindung stattfindet, wird er an der Kollisionstangente reflektiert.

Mit dem *Foreign Function Interface* (FFI) von Rust wird Cuda C++ in das Rust-Programm eingebunden. Die Struktur `Cudaworld` verwaltet alle Pointer zu den VRAM Daten und stellt Methoden zur Handhabung der GPU-Ressourcen bereit.

3.2.5 Parameter

Die Simulation kann mit verschiedenen Parametern angepasst werden, um unterschiedliche Szenarien zu testen. Sie können über eine Konfigurationsdatei im *JSON*-Format oder im Programmcode mit den `settings`-Makros festgelegt werden. Die Parameter werden hier in die Kategorien *Biologische Parameter* 3.1, *Simulationsparameter* 3.2, *Genom-Parameter* 3.3 und *Physikalische Parameter* 3.4 unterteilt.

Parameter können nur über das `edit_settings`-Makro geändert werden, um programmweite Konsistenz zu gewährleisten. Nicht alle Programmeigenschaften sind als Parameter ausgeführt. Die GPU-Beschleunigung stellt eine solche Ausnahme dar und kann lediglich durch entsprechende Codeänderungen (`world.cuda_initialize()`) ein- oder ausgeschaltet werden.

Parameter	**Beschreibung**	**Standardwert**	**Kommentar**
`concentration_range`	Maximale/minimale Konzentration der Second Messenger	(-32, 32)	-
`possible_ligands`	Anzahl verschiedener Liganden-typen in der Simulation	16	Muss eine 2er-Potenz sein
`ligands_per_entity`	Anzahl Liganden, die eine Entität synthetisieren kann	2	-
`receptor_types_per_entity`	Anzahl verschiedener Rezeptor-typen pro Entität	2	-
`receptors_per_entity`	Maximale Anzahl Rezeptoren pro Entität	10′000	-
`max_age`	Maximales Alter einer Entität in Sekunden	50	0 entspricht unendlich
`max_size`	Grösse bei der sich die Entität teilt	2.0	-
`max_plasmid_count`	Maximale Anzahl Plasmide pro Entität	5	-
`standard_plasmid_count`	Startanzahl Plasmide pro Entität	1	-
`entity_acceleration`	Beschleunigung der Entitäten	1.0	-
`idle_energy_cost`	Energieverbrauch pro Sekunde abhängig von der Fläche	1×10^{-3}	Energieverbrauch im Ruhezustand
`entity_run_energy_cost`	Energieverbrauch für Bewegung (*Run*) pro Sekunde	0.001	-
`entity_tumble_energy_cost`	Energieverbrauch für Bewegung (*Tumble*) pro Sekunde	0.005	-
`max_energy_ligand`	Maximale Energie eines Liganden	1.0	-
`min_energy_ligand`	Minimale Energie eines Liganden	0.1	-
`enable_entity_ligand_emission`	Gibt an, ob Entitäten Liganden synthetisieren können	true	-
`tumble_chance`	Wahrscheinlichkeit für Richtungswechsel im *Tumble*-Zustand pro Update	0.3333	-
`toxins_active`	Aktivierung von toxischen Liganden	true	-

Tabelle 3.1: Biologische Parameter

Parameter	**Beschreibung**	**Standardwert**	**Kommentar**
`default_population`	Anfangsanzahl der Entitäten	100	-
`dimensions`	Abmessungen der Simulationswelt (Breite, Höhe)	(100, 100)	-
`store_capacity`	Anzahl States, die zwischengespeichert werden	1024	Ist die Zahl zu gross, wird die Simulation nicht gespeichert
`fps`	Anzahl Updates(States) pro Sekunde	60	Bei hohen Geschwindigkeiten hoch anzusetzen
`cuda_slots_per_cell`	Anzahl Slots pro Zelle im CUDA-Gitter	10	Nicht manuell verändern

Tabelle 3.2: Simulationsparameter

Parameter	**Beschreibung**	**Standardwert**	**Kommentar**
`mutation_rate`	Wahrscheinlichkeit einer Mutation pro Bit	0.001	-
`std_dev_mutation`	Standardabweichung σ für Schwellenwert-Mutationen	1.5	μ ist 0
`mean_random`	Mittelwert μ für Zufallswerte	0.0	Gilt nur für Schwellenwerte
`std_dev_random`	Standardabweichung σ für Zufallswerte	10.0	Gilt nur für Schwellenwerte

Tabelle 3.3: Genom-Parameter

Parameter	**Beschreibung**	**Standardwert**	**Kommentar**
`spawn_size`	Startradius der Entitäten	1.0	Startenergie ist $r^2\pi$
`give_start_vel`	Startgeschwindigkeit der Entitäten	true	Wenn false, starten die Entitäten mit Geschwindigkeit 0
`velocity`	Standardgeschwindigkeit der Entitäten	3.0	-
`ligand_velocity`	Standardgeschwindigkeit der Liganden	2.0	-
`general_force`	Richtungskraft (x, y)	(0.0, 0.0)	-
`drag`	Fluidwiderstand der Umgebung	0.5	-

Tabelle 3.4: Physikalische Parameter

4 Ergebnisse

In diesem Kapitel werden die Resultate der durchgeführten Simulationen präsentiert. Die Parameter der verschiedenen Experimente sind für jede Simulation mit einem Link zur entsprechenden Konfigurationsdatei, den Rohdaten, sowie den generierten Grafiken und dem Programmcode unter folgendem Link verfügbar: `https://github.com/juli3200/UniSim/tree/main/experiments`

4.1 Einfluss der Mutationsrate

Es wurden drei Simulationen mit identischen Parametern ausschliesslich der Mutationsrate durchgeführt. Die Mutationsraten wurden auf 0, 0.01 und 0.1 gesetzt. In diesen Simulationen besitzt jede Entität nur ein Rezeptorgen. Es gab vier Ligandentypen und das maximale Alter war auf 20 Sekunden gesetzt. Zu Beginn der Simulation wurden eine Ligandenquelle mit dem Typ 1 platziert. Nach 2500 Simulationsschritten wurden alle existierenden Liganden entfernt und die Ligandenquelle mit einer von Typ 2 ersetzt.

Die folgenden Diagramme 4.1, 4.2 und 4.3 zeigen die Allelhäufigkeiten der Rezeptortypen für die drei verschiedenen Mutationsraten. Gemeint ist die extrazelluläre Domäne des Rezeptors, die den Liganden bindet (vgl. Abschnitt 3.1.4).

Bei allen drei Mutationsraten ist zu erkennen, dass das Allel für den Rezeptortyp 1 (blau) zu Beginn der Simulation dominiert. Nach dem Wechsel der Ligandenquelle auf Typ 2 (hellblau)

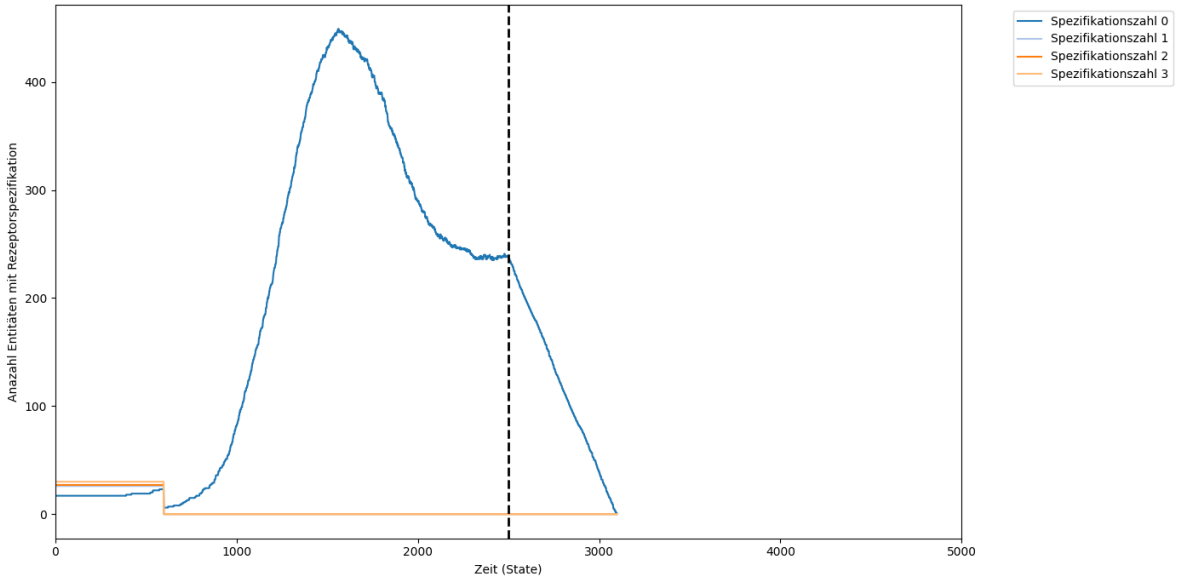

Abbildung 4.1: Allelhäufigkeiten mit Mutationsrate Null

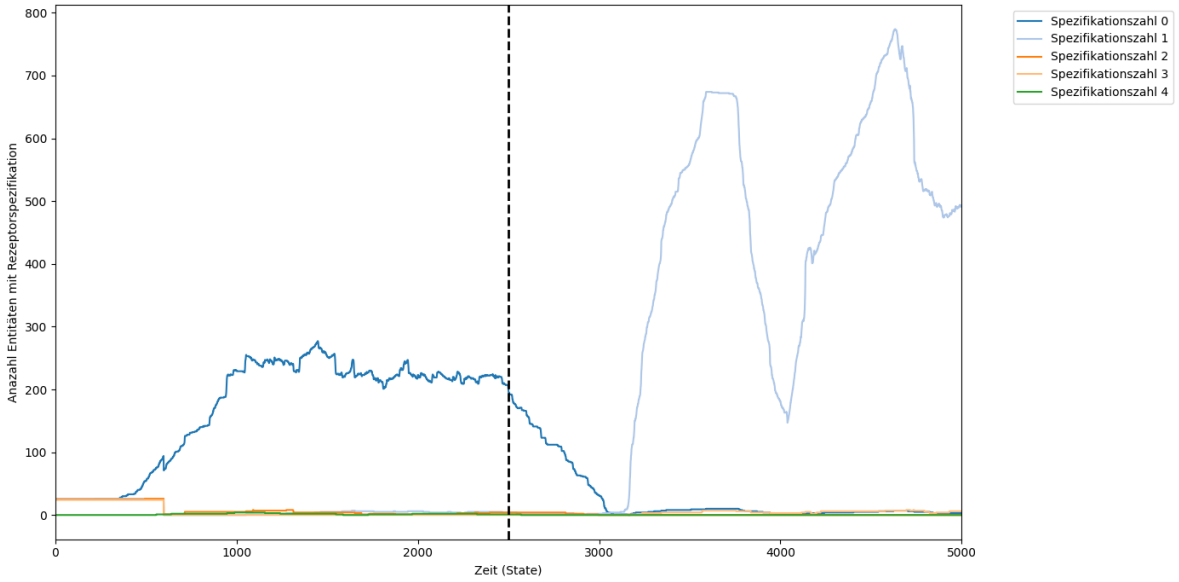

Abbildung 4.2: Allelhäufigkeiten mit Mutationsrate 0.01

steigt das Allel für den Rezeptortyp 2 an. Mit höherer Mutationsrate geschieht dies schneller. Interessant ist, dass bei Mutationsrate 0.1 das Allel für den Rezeptortyp 1 nach dem Wechsel der Ligandenquelle wieder ansteigt. Bei der Mutationsrate 0 sterben alle Entitäten aus.

Nach dem Wechsel der Ligandenquelle auf Typ 2, ist bei der Mutationsrate 0.01 und 0.1 ein

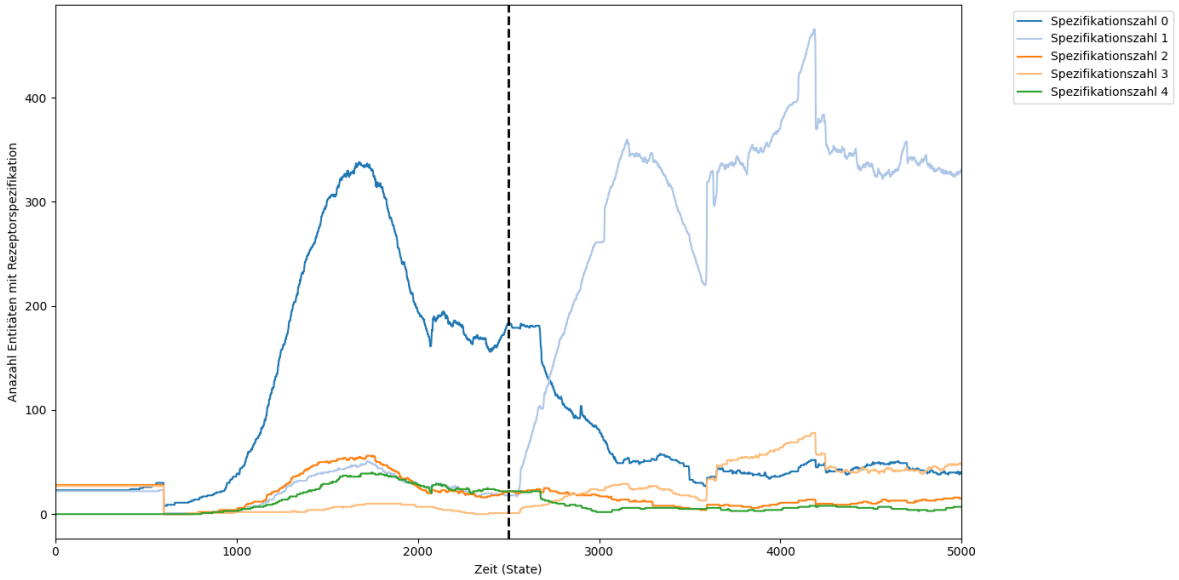

Abbildung 4.3: Allelhäufigkeiten mit Mutationsrate 0.1

kurzzeitiges Absinken der Gesamtpopulation zu beobachten, auf das ein starkes Wachstum folgt. In Simulation 4.2 stirbt die Population sogar beinahe aus, bevor sie sich wieder erholt. Die Bevölkerungszahl erreichen darauf höhere Werte als vor dem Wechsel der Ligandenquelle. Vor allem bei der Mutationsrate 0.01 erkennt man grosse Schwankungen in der Gesamtpopulation. Dieser Effekt kann man aber bei allen drei Mutationsraten durch die ganze Simulation beobachten.

4.2 Entwicklung kooperativer Verhaltensmuster

Um die Kooperation zwischen Einzellern zu untersuchen, wurden Simulationen mit Plasmidübertragung und toxischen Liganden durchgeführt. In diesen Simulationen konnten die Einzeller Resistenzgene gegen die toxischen Liganden an andere Entitäten übertragen. Eine Art, die Plasmide teilt, sie also an andere Einzeller weitergibt, wird als kooperativ bezeichnet.

Für das erste Experiment wurde die Mutationsrate auf null gesetzt und die toxischen Liganden wurden der Simulation bei $t = 0$ hinzugefügt. Die Mutationsrate wurde so gewählt, um die

Darstellung der folgenden Grafiken zu vereinfachen. Abbildung 4.4 zeigt die Verteilung der verschiedenen Arten über die Zeit. Eine Art ist definiert durch 90-prozentige Übereinstimmung im Genom. Abbildung 4.5 zeigt das Vorkommen der Plasmide in der Population über die Zeit.

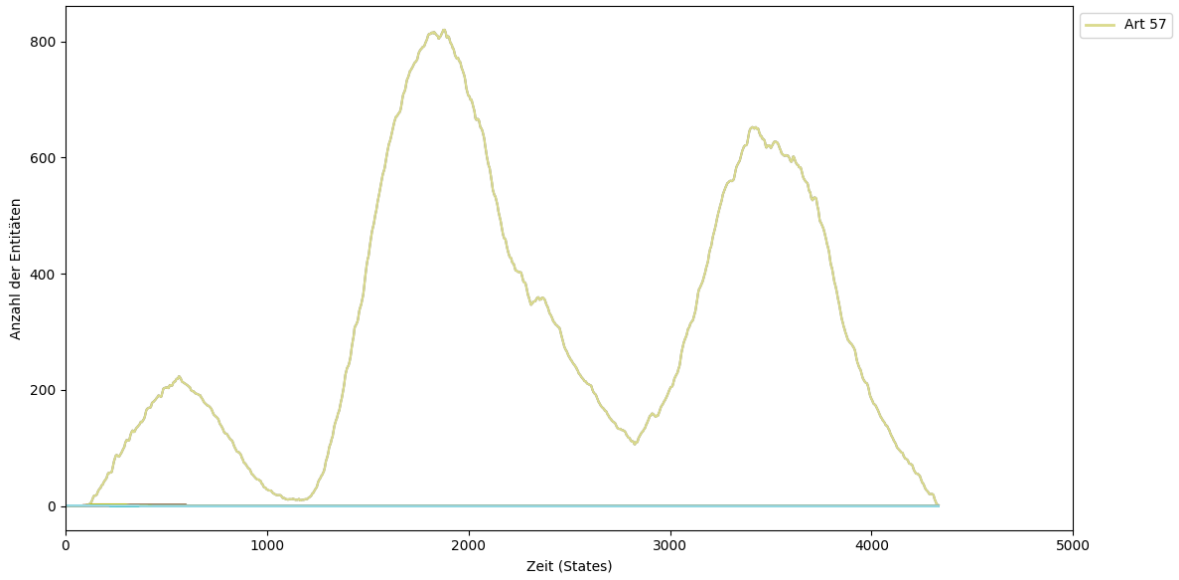

Abbildung 4.4: Artenvorkommen bei Mutationsrate 0 und toxischen Liganden ab $t = 0$

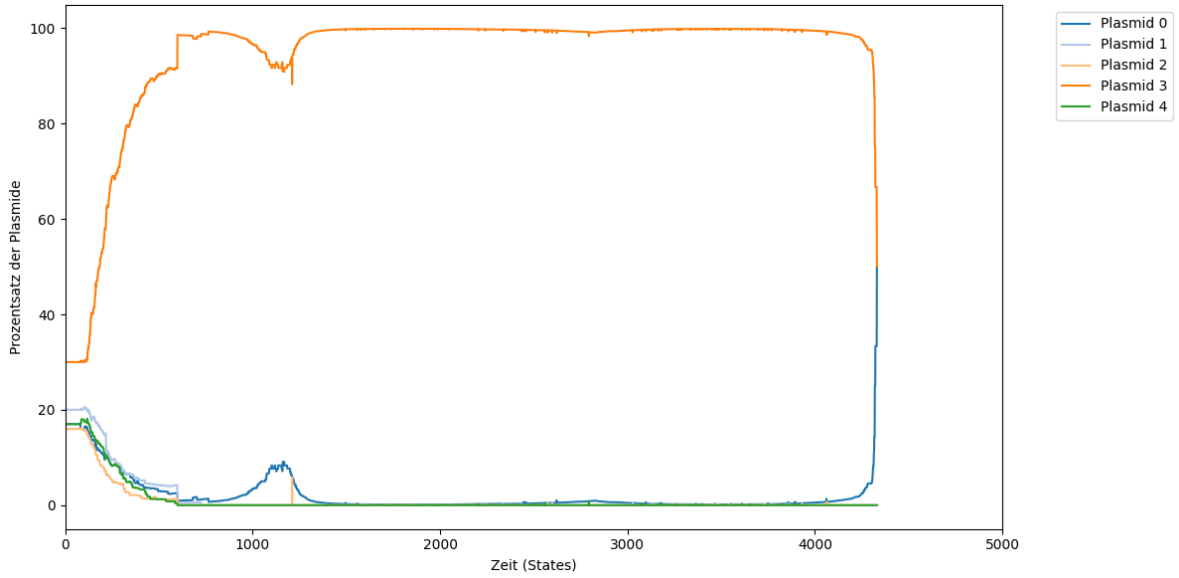

Abbildung 4.5: Plasmidvorkommen bei Mutationsrate 0 und toxischen Liganden ab $t = 0$

In Abbildung 4.4 ist zu erkennen, dass der Anstieg der Population der Art 57 stark mit dem Anstieg von Plasmid 1 in Abbildung 4.5 korreliert. Auch der kurzfristige Anstieg von Plasmid

0 bei $t \approx 1200$ korrespondiert mit dem beinahe vollständigen Aussterben von Art 57 zum gleichen Zeitpunkt. Es sind starke Schwankungen in der Gesamtpopulation zu beobachten, die noch extremer ausfallen als in den vorherigen Simulationen.

Es wurden noch weitere Simulationen mit ähnlichen Parametern durchgeführt. Geändert wurden die Mutationsrate, die Geschwindigkeit der Entitäten oder deren Energieverbrauch. In allen Simulationen hing die Plasmidverbreitung stark mit dem Überleben eines bestimmten Phänotyps zusammen.

5 Diskussion

Um die Fragestellungen aus Kapitel 1 zu beantworten, werden die Resultate aus Kapitel 4 im Folgenden diskutiert.

5.1 Interpretation der Ergebnisse

Die Fragestellung wird in zwei Teile gegliedert: Zum einen, ob evolutionäre Vorgänge simuliert werden können und zum anderen, welche Umstände kooperative Verhaltensmuster unter Entitäten fördern.

5.1.1 Evolutionäre Vorgänge in der Simulation

In den Simulationen mit Mutationsrate 0.1 und 0.01 kann man eine klare Veränderung des Phänotyps der Einzeller beobachten, die an der Anpassung der extrazellulären Domäne des Rezeptors an die veränderte Umwelt klar ersichtlich wird. So dominieren nach $t = 2500$ von den Simulationen 4.2 & 4.3 Entitäten mit Rezeptortyp 2, ein Phänotyp, der zuvor nicht (mehr) existierte. Besonders gut sieht man das bei Simulation 4.2. Da der Wiederanstieg der Population erst knapp vor deren Ausrottung geschieht, ist davon auszugehen, dass der Phänotyp für den Rezeptortyp 2 zuvor nicht (mehr) existierte und erst nach der Umweltveränderung durch Mutationen entstanden ist.

Die Umweltveränderung bei $t = 2500$ entspricht einem in Kapitel 2.2.2 beschriebenen Flaschenhalseffekt. Da die vier Merkmale der Evolution (vgl. Abschnitt 2.2.1) in den zwei erfolgreichen Simulationen mit Mutationsrate 0.01 und 0.1 vorhanden sind, war das Gelingen der Anpassung zu erwarten. Die Mutationsrate ist zentral für das Gelingen und die Geschwindigkeit dieser Anpassung. Bei einer Mutationsrate von 0 sterben alle Entitäten aus, da keine genetische Variation vorhanden ist, die eine Anpassung an die veränderte Umwelt ermöglicht. Ist die Mutationsrate zu hoch, kann die Anpassung erschwert werden, da vorteilhafte Mutationen durch weitere Mutationen wieder verloren gehen können [13, S. 50–61]. Dieser Effekt konnte mit der Mutationsrate von 0.1 noch nicht klar beobachtet werden.

Hier wurde nur ein Merkmal (die Variation) betrachtet, aber die anderen Merkmale sind ebenfalls vorhanden: Selektion geschieht durch die veränderte Umwelt, Vererbung ist durch die Replikation der Einzeller gegeben und Zeit ist durch die fortlaufende Simulation gegeben.

Höhere Populationszahlen nach $t = 2500$ sind auf die höhere Energie pro Ligand des Typs 2 zurückzuführen (vgl. Abschnitt 3.1.2). Die beobachteten starken Schwankungen in der Gesamtpopulation (vgl. Abbildungen 4.1, 4.2 und 4.3) sind dem Wachstum einer Bakterienpopulation ähnlich, die sich in einer begrenzten Umgebung mit endlichen Ressourcen befindet (vgl. Abschnitt 2.1.2). Dies weist auf ein bemerkenswert realitätsnahes Verhalten der Simulation hin.

5.1.2 Kooperation

In der Simulation 4.4 ist kein kooperatives Verhalten zu beobachten. Auch wenn sich ein Resistenzgen schnell durchsetzt, liegt dies nicht an horizontaler Genübertragung, sondern daran, dass die Entitäten mit dem Resistenzgen gegen den toxischen Liganden resistent sind und sich somit besser fortpflanzen können. Die starken Populationsschwankungen deuten auch in dieser Simulation auf ein realitätsnahes Verhalten hin, ähnlich wie in den vorherigen Simulationen. Toxische Liganden sind nicht verantwortlich für die stärkeren Schwankungen, da zu deren Zeitpunkt der Simulation bereits fast alle Entitäten das Resistenzgen besassen (vgl. Abbildung 4.5).

Die Verbreitung des Resistenzgens geschieht hauptsächlich durch die Replikation (vertikaler Gentransfer, Abschnitt 2.1.1) der Einzeller mit dem Resistenzgen. So sind (fast) alle Entitäten direkte Nachkommen von wenigen ursprünglichen Entitäten, die das Resistenzgen und die richtigen Rezeptoren für die nicht-toxischen Liganden besassen. Der horizontale Genübertransfer spielt in dieser Simulation eine untergeordnete Rolle. So bildet auch diese Simulation einen Flaschenhalseffekt ab und nicht die erwünschte Kooperation. So kam eine Kooperation zwischen den Einzellern nicht zustande.

In weiteren Simulationen mit veränderten Parametern wurde versucht, die horizontale Genübertragung zu fördern. Dies sollte durch eine vermehrte Häufigkeit von Kollisionen zwischen den Einzellern erreicht werden, da die Plasmidübertragung nur bei Kollisionen stattfinden kann. Alle Versuche blieben jedoch erfolglos.

Auch andere Mechanismen könnten kooperativ wirken. Zum Beispiel die Ligandensynthese könnte zu simpler Kommunikation zwischen den Einzellern führen. Diese Mechanismen wurden jedoch in dieser Arbeit nicht untersucht.

Kooperation ist in natürlichen biologischen Systemen weit verbreitet. Ob sie sich etabliert, hängt jedoch stark von Umweltbedingungen und Verhaltensmustern ab. Da diese Voraussetzungen in der vorliegenden Simulation nicht gegeben waren, setzte sich stattdessen eine kompetitive Entwicklung durch. Die Ergebnisse zeigen somit, dass Kooperation kein zwangsläufiges Resultat evolutionärer Prozesse ist, sondern nur unter spezifischen Bedingungen entsteht.

5.2 Ausblick & Fazit

Ziel dieser Arbeit war es zu untersuchen, ob sich biologische Evolution mit Hilfe einer computerbasierten Simulation modellieren lässt und unter welchen Bedingungen sich kooperative Verhaltensmuster bei Einzellern entwickeln können. Die Ergebnisse zeigen, dass evolutionäre Vorgänge in der Simulation erfolgreich modelliert werden konnten. Kooperation konnte jedoch unter den gezeigten Bedingungen nicht beobachtet werden.

Die Simulation erfüllt die vier Merkmale der Evolution (Variation, Selektion, Vererbung und Zeit) und konnte somit eine Anpassung der Einzeller an eine veränderte Umwelt erfolgreich modellieren. Es waren ausserdem realitätsnahe Verhaltensmuster zu beobachten, wie z.B. starke Schwankungen in der Gesamtpopulation. Kooperation zwischen den Einzellern konnte in den durchgeführten Simulationen nicht beobachtet werden. Die horizontale Genübertragung war zu schwach vertreten im Vergleich zur Selektion durch die Umwelt (vertikaler Gentransfer). Vielleicht könnten andere Parameter oder Umweltveränderungen kooperatives Verhalten fördern, was in zukünftigen Arbeiten untersucht werden könnte.

Die entwickelte Simulation unterliegt mehreren Limitationen. Alle Vereinfachungen und Annahmen, die in Kapitel 3.1 beschrieben wurden, könnten das Verhalten der Simulation beeinflussen. Die Ergebnisse sind stark von den gewählten Parametern abhängig, die in dieser Arbeit nur in einem kleinen Rahmen variiert wurden. Nachträglich hätten Liganden und toxische Substanzen getrennt voneinander eingeführt werden sollen. So könnte die Energie pro Ligand und die Toxizität unabhängig voneinander variiert werden. Auch die Energieformel mit dem Hamming-Gewichtungsfaktor sollte überarbeitet werden (vgl. Abschnitt 3.1.2), da sie zu einer unnötigen Komplexität führt. Besser wäre es, die Energie pro Ligand linear zu gestalten.

Das Programm wurde modifizierbar und erweiterbar gestaltet, sodass weitere Experimente mit anderen Zielsetzungen und zusätzlichen Fähigkeiten der Einzeller durchgeführt werden können. Anknüpfungspunkte für zukünftige Arbeiten sind unter anderem die Entwicklung gerichteter Chemotaxis oder die Überlebensstrategien bei komplizierten Umweltveränderungen. Auch die Untersuchung von Kooperation unter anderen Bedingungen und auch anderen Kooperationsformen, wie z.B. die Fähigkeit einander zu helfen, bietet Potenzial für weitere Forschungen.

Glossar

Escherichia coli Prokaryotisches Bakterium

Allel Variante eines Gens, die unterschiedliche Merkmalsausprägungen kodieren kann

Chemotaxis Bewegung als Reaktion auf chemische Reize

CPU Hauptprozessor

DNA Desoxyribonukleinsäure; Träger der genetischen Information in einer Doppelhelix aus Nukleotiden

Elastische Kollision Kollisionsmodell ohne Energieverlust

Entität Simuliertes Einzeller-Objekt; zentraler Akteur der Simulation

Evolution Veränderung der genetischen Zusammensetzung einer Population über Generationen

FFI Foreign Function Interface

Flaschenhalseffekt Starke, plötzliche Reduktion der Populationsgröße mit Verlust genetischer Vielfalt

Genom Gesamtheit der genetischen Information

Genotyp Genetische Information eines Organismus

GPU Grafikkarte

Horizontaler Gentransfer Übertragung von genetischer Information zwischen Individuen derselben Generation (z. B. über Plasmide)

Ligand In der Simulation Sammelbegriff für Signalmoleküle und Nährstoffe (und optional auch Schadstoffe); kann von Rezeptoren gebunden/aufgenommen werden

Ligandensynthese Fähigkeit einer Entität, Liganden selbst zu erzeugen und auszusenden; kostet Energie und kann (potenziell) Kommunikation ermöglichen

Mutation Veränderung des genetischen Materials

Physik-Engine Programmteil für Bewegungen, Kräfte und Kollisionen von Entitäten/Liganden

Phänotyp Beobachtbare Merkmale/Eigenschaften

Plasmid Kleines, zusätzliches DNA-Molekül; kann unabhängig von Zellteilung verbreitet werden

Prokaryot Organismus ohne Zellkern (z. B. Bakterien)

Rezeptor Proteine auf der Zelloberfläche, die spezifische Liganden binden können

Run & Tumble Bewegungsprinzip: Run = geradlinige Bewegung; Tumble = zufällige Richtungsänderungen

Second Messenger Intrazelluläres Signal, das nach Rezeptorbindung entsteht

Selektion Selektionsdruck durch Umweltbedingungen, der den Fortpflanzungserfolg unterschiedlicher Merkmale beeinflusst

State Ein einzelner Simulationszustand (Update-Schritt)

Variation Unterschiede zwischen Individuen einer Population

Vererbung Weitergabe von Merkmalen/Genen an Nachkommen

Vertikaler Gentransfer Weitergabe genetischer Information von Eltern auf Nachkommen (z. B. bei Zellteilung)

VRAM Arbeitsspeicher der GPU

Literaturverzeichnis

[1] Chad Bereck. *2-Dimensional Elastic Collisions without Trigonometry*. URL: https://www.vobarian.com/collisions/2dcollisions2.pdf.

[2] Howard C. Berg und Douglas A. Brown. „Chemotaxis in Escherichia Coli Analysed by Three-dimensional Tracking“. In: *Nature* 239.5374 (Okt. 1972), S. 500–504. ISSN: 1476-4687. DOI: 10.1038/239500a0. URL: https://www.nature.com/articles/239500a0 (besucht am 29. 11. 2025).

[3] Charles Darwin. *On the Origin of Species*. 2nd ed. Oxford, 1859.

[4] Eckebrecht, Detlef. *Natura 9-12: Grundlagen der Biologie für Schweizer Maturitätsschulen*. 1. Auflage, Ausgabe für die Schweiz. Baar: Klett und Balmer Verlag, 2018. ISBN: 978-3-264-84038-4.

[5] A. E. Eiben und J. E. Smith. „Evolutionary Computing: The Origins“. In: *Introduction to Evolutionary Computing*. Hrsg. von A.E. Eiben und J.E. Smith. Berlin, Heidelberg: Springer, 2015, S. 13–14. ISBN: 978-3-662-44874-8. DOI: 10.1007/978-3-662-44874-8_2.

[6] Michael Eisenbach. *Chemotaxis*. London: Imperial college press, 2004. ISBN: 978-1-86094-413-0.

[7] *Floating-Point Operation | Computer Science KB*. URL: https://faq.computersciencewiki.org/index.php/home/article/floating-point-operation (besucht am 03. 01. 2026).

[8] *G Protein-Coupled Receptors and Their Role as Drug Targets*. The Scientist. URL: https://www.the-scientist.com/g-protein-coupled-receptors-and-their-role-as-drug-targets-72951 (besucht am 12. 12. 2025).

[9] *Gene Transfer: Types, Mechanisms, and Methods*. URL: https://www.abcam.com/en-us/knowledge-center/dna-and-rna/gene-transfer-types-mechanisms-and-methods.

[10] Heewook Lee u. a. „Rate and Molecular Spectrum of Spontaneous Mutations in the Bacterium Escherichia Coli as Determined by Whole-Genome Sequencing“. In: *Proceedings of the National Academy of Sciences of the United States of America* 109.41 (9. Okt. 2012), E2774–2783. ISSN: 1091-6490. DOI: 10.1073/pnas.1210309109. PMID: 22991466.

[11] Gilad Maayan. *GPU Parallel Computing: Techniques, Challenges, and Best Practices*. Atlantic.Net. 7. Jan. 2025. URL: https://www.atlantic.net/gpu-server-hosting/gpu-parallel-computing-techniques-challenges-and-best-practices/ (besucht am 03. 01. 2026).

[12] David Masci. „Darvin and His Theory of Evolution“. In: (2. Apr. 2009). URL: https://www.pewresearch.org/wp-content/uploads/sites/7/2009/02/Darwin-and-his-theory-of-evolution-pdf.pdf (besucht am 12. 10. 2025).

[13] Lawrence E. Mettler, Thomas G. Gregg und Schaffer Henry E. *Population Genetics and Evolution*. 2nd ed. Englewood Cliffs (N.J.): Prentice hall, 1988. ISBN: 978-0-13-685678-8.

[14] *Mutation*. URL: https://www.u-helmich.de/bio/lexikon/M/mutation.html (besucht am 29. 11. 2025).

[15] Parker Nina, Lister Phillip und Schneegurt Mark. „Microbial Growth“. In: *Microbiology*, S. 323–351. ISBN: 978-1-947172-23-4.

[16] *Object Oriented Programming Features - The Rust Programming Language*. URL: https://doc.rust-lang.org/book/ch18-00-oop.html (besucht am 24. 12. 2025).

[17] *Plasmid • Aufbau, Konjugation und Einteilung*. Studyflix. URL: https://studyflix.de/biologie/plasmid-2128 (besucht am 29. 12. 2025).

[18] *Prokaryoten Prokaryotische Zelle, Unterschiede zu Eukaryoten*. Studyflix. URL: https://studyflix.de/biologie/prokaryoten-2120 (besucht am 28. 11. 2025).

[19] *Rayon - Rust*. URL: https://docs.rs/rayon/latest/rayon/ (besucht am 04. 01. 2026).

[20] *Rezeptoren: Klassifikation & Überblick | Lecturio*. 10. Jan. 2022. URL: https://www.lecturio.de/artikel/medizin/rezeptoren/ (besucht am 26. 11. 2025).

[21] Dr Rolf Steinmüller. „Aktuell | Ernährungslehre & Praxis“. In: *Ernährungs Umschau* (Apr. 2010).

[22] Mazal Varon und David Gutnick. „Chemotaxis as a means of Cell-Cell Communication in Bacteria“. In: *Chemotaxis in Escherichia coli analysed by Three-dimensional Tracking*. London, S. 216–245. ISBN: 978-1-86094-413-0.

[23] Pezzza's Work, director. *Writing a Physics Engine from Scratch - Collision Detection Optimization*. URL: https://www.youtube.com/watch?v=9IULfQH7E90.

Verwendung von Künstlicher Intelligenz

Bei der Erstellung von *UniSim* wurde GitHub Copilot verwendet, um Code-Snippets vorzuschlagen und die Entwicklung zu beschleunigen. Die Vorschläge wurden jeweils kritisch geprüft und bei Bedarf angepasst. **Keine** nicht angegebenen Code-Abschnitte wurden direkt übernommen und die übergeordneten Strukturen und Algorithmen wurden vollständig selbstständig entwickelt.

Python-Scripts im `scripts\` Ordner wurden von ChatGPT und GitHub Copilot erstellt oder angepasst. Sie gehören nicht zum eigentlichen Projekt, sondern dienen der Datenanalyse und Visualisierung der Simulationsergebnisse.

Es folgt ein Link zu einer Sammlung der verwendeten ChatGPT Prompts:

`https://chatgpt.com/g/g-p-695543e1bc548191b120fac8bf0cf457-ma/project`

Abbildungsverzeichnis

Tabellenverzeichnis

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich meine Maturitätsarbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen und KI-/LLM-Tools angefertigt habe. Die Mitwirkung von anderen Personen hat sich auf Beratung und Korrekturlesen beschränkt. Alle verwendeten Unterlagen und Gewährspersonen sind vollständig aufgeführt.

7. Januar 2026 Greifensee

_____

Unterschrift