Kollegium Spiritus Sanctus Brig

Maturaarbeit 2023/24

Computergestützte Nachbildung evolutionärer Anpassung

Entwicklung eines Computerprogramms

Von:

Carron David Réne Pierre, 5F

Eingereicht in den Fachbereichen Informatik & Biologie

Betreut durch:

Gasche Christoph
Perrig Marco

Brig, den 30.11.2023 Nummer 2023/76

Inhaltsverzeichnis

1 Einleitung und Zielsetzung

Die Evolutionsforschung ist ein sehr interessantes Teilgebiet der Biologie, das auch in Zukunft wichtig bleibt. Das bessere Verstehen von evolutionären Prozessen bringt uns einen tiefgehenden Einblick in das Entstehen und die Entwicklung von komplexen biologischen Systemen. Diese Prozesse sind grundlegend für die Artenvielfalt auf der Erde. Dank den massiven Fortschritten im Bereich der Computergeschwindigkeit im letzten Jahrhundert ist es möglich, diese Prozesse auch innerhalb eines Computerprogrammes zu beobachten. Dies bietet eine wertvolle neue Plattform zur Untersuchung von Evolutionsmechanismen.

In dieser Arbeit präsentiere ich ein Programm, welches dazu konzipiert wurde, evolutionäre Prozesse auf eine simple, einfach zu verstehende Weise darzustellen. So erschaffe ich ein Werkzeug, um komplexe, vielseitige Prozesse darstellen und analysieren zu können. Um möglichst viele verschiedene Szenarien darstellen zu können, ist der Simulationsaufbau anpassbar.

Das Programm ist mit der Processing Programmiersprache geschrieben. Processing verwendet Java als Basissprache und bietet eine vereinfachte Möglichkeit, visuelle und interaktive Anwendungen zu erstellen. Ich werde auch auf einen spannenden Aspekt der Programmierung eingehen: die optimierten Distanzberechnungen zwischen einzelnen Bestandteilen der Simulation.[1]

Ziel der Arbeit ist es also, ein Computerprogramm zu entwickeln, bei welchem durch einfache und veränderbare Regeln komplexe evolutionäre Prozesse beobachtbar sind. Das Programm zeigt diese Prozesse auf verständliche Weise auf. Es können viele Simulationsparameter konfiguriert werden. Eine hyperrealistische Simulation oder das Produzieren quantitativer Daten ist nicht Ziel dieser Arbeit.

In den nachfolgenden Abschnitten werde ich mein entwickeltes Computerprogramm ausführlich vorstellen und seine biologischen aber auch informatiktechnischen Aspekte beleuchten.[2]

2 Hauptteil

2.1 Überblick

2.1.1 Allgemeines

Das erstellte Programm simuliert die Entwicklung einer Bakterienkolonie in einer kontrollierten, begrenzten Umgebung, kann jedoch auch zur generelleren Simulation evolutionärer Prozesse verwendet werden. Nahrung in Form von stationären, sich vermehrenden Partikeln ist stets vorhanden. Um zu dieser Nahrung zu gelangen und deren Energie aufzunehmen, bewegen sich die Organismen mit einer konstanten Geschwindigkeit zu einem Partikel in ihrer Nähe hin. Hat ein Lebewesen genug Energie gesammelt, vermehrt sich dieses asexuell und ein neues, gleiches Lebewesen entsteht. Das Verhalten der Organismen verändert sich aufgrund verschiedener Attribute, welche wiederum durch die Aktivierung

[1] Kap. 2.3.2.
[2] Der Sourcecode befindet sich im Anhang 2.

diverser veränderbarer Gene beeinflusst werden. Wie stark ein Gen aktiviert ist, kann sich durch zufällig vorkommende Mutationen verändern.

2.1.2 Aufbau des Programms

Der Simulationsprozess lässt sich in drei Teile aufteilen: Simulationsaufbau, Simulation und Analyse der generierten Statistik. Vor dem Start des Programmes können verschiedene Parameter in JSON-Dateien konfiguriert werden, um das simulierte Umfeld sowie die sich darin befindenden Lebewesen nach Belieben zu verändern. Alle Dateien, welche das zu simulierende Ökosystem verändern, werden zusammen in einem Simulationsaufbau-Ordner gespeichert. Dies erlaubt es, mehrere Simulationsbedingungen vorzubereiten und diese dann einzeln zu simulieren. Während der laufenden Simulation sind Organismen als farbige Punkte verschiedener Grösse dargestellt. Diese suchen dann in ihrem zweidimensionalen Lebensraum nach Nahrungspartikeln. Die Simulation läuft in einem halbflüssigen Medium ab, worin sich die Organismen bewegen können, die Partikel jedoch stationär bleiben. Je nach Konfiguration werden zusätzliche Nahrungspartikel hinzugefügt. Durch die Aufnahme von Nahrungspartikeln gewinnen die Organismen an Energie, welche durch diverse Prozesse wieder verbraucht wird. Um einen besseren Einblick in die Entwicklung der Lebewesen während der Simulation erhalten zu können, können zu jedem Zeitpunkt während der Simulation diverse Graphen über die Veränderung von Lebensformen und deren Umfeld dargestellt werden. Die Navigation zwischen diesen verschiedenen Modi wird durch ein einfach aufgebautes Menu ermöglicht.

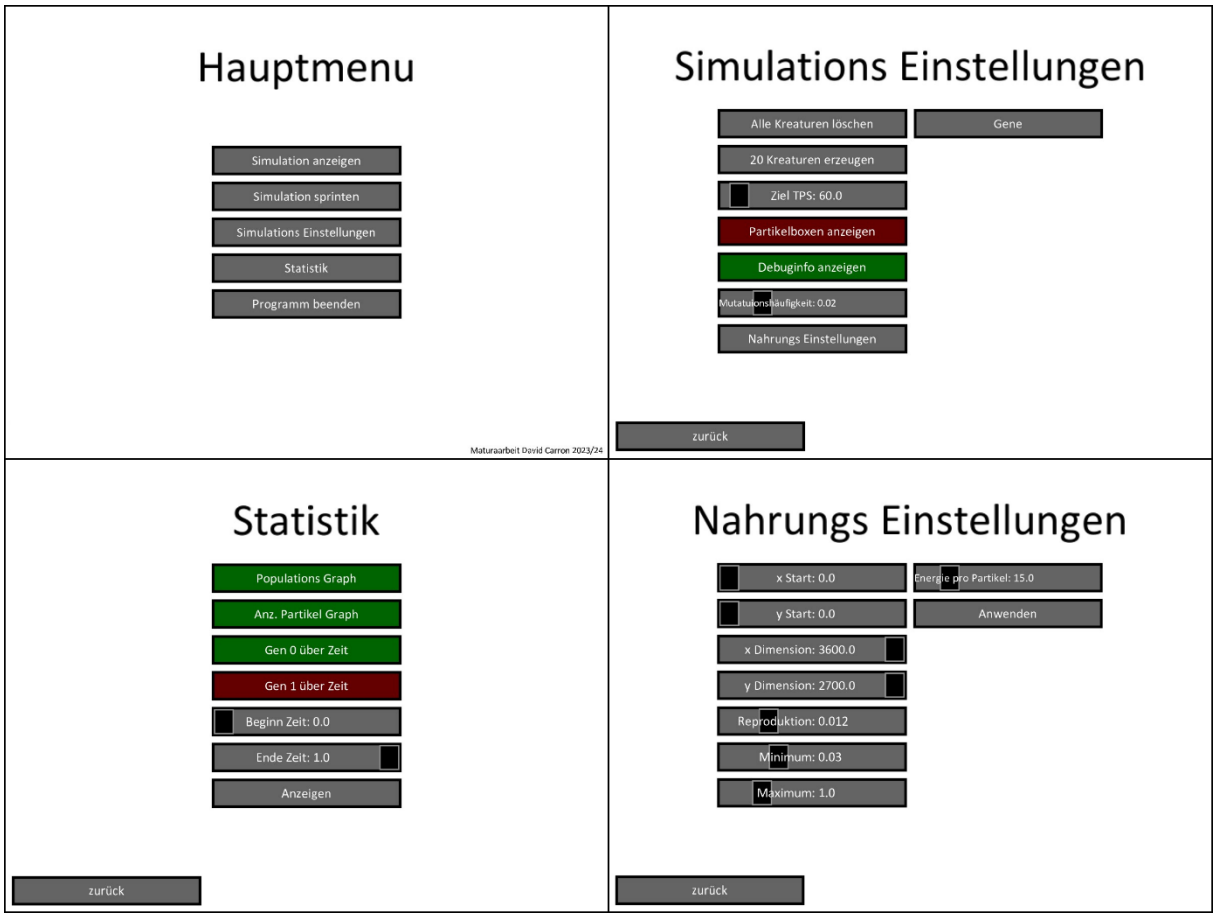


Abbildung 1: Menuführung

Legende: Über verschiedene, baumartig aufgebaute Schaltflächen lässt sich das Programm steuern.

2.1.3 Evolutionäre Faktoren

Damit evolutionäre Prozesse geschehen und beobachtet werden können, müssen folgende Bedingungen erfüllt sein:[3]

- Würden sich alle Individuen vermehren, würde die Population wachsen.
- Abgesehen von periodischen Fluktuationen bleibt die Populationsgrösse jedoch ungefähr gleich.
- Ressourcen wie Nahrung sind limitiert, bleiben aber im Laufe der Zeit relativ stabil.
- Individuen in einer Population unterscheiden sich voneinander.
- Diese Unterschiede sind, wenigstens teilweise, vererblich.
- Natürliche Selektion findet statt: Individuen, welche besser an ihr Umfeld angepasst sind, haben eine höhere Überlebens- und Reproduktionswahrscheinlichkeit als Individuen, welche schlechter an dieses Umfeld angepasst sind.

Alle diese Bedingungen sind in der Simulation erfüllt, solange die Startparameter sinnvoll festgelegt worden sind.

2.1.4 Aufbau des Organismus

Organismen sind in der Simulation eher einfach dargestellt. Jeder Organismus verfügt über einen internen Energiespeicher, welcher aufgefüllt oder entleert werden kann. Die Darstellungsgrösse eines Individuums widerspiegelt dessen Energielevel, hat jedoch keinen anderen Einfluss. Bei der Nahrungsaufnahme füllen sich die Energiereserven des Organismus, welche der Organismus durch seine Existenz und durch Bewegung wieder verbraucht. Bei einer Teilung des Organismus verteilt sich die im Organismus gespeicherte Energie auf die zwei neuen Organismen. Die Teilung erfolgt nur bei einem genügend hohen Energielevel. Auf diese Art wird der Stoffwechsel der Organismen modelliert.

2.1.5 Erklärung des Simulationsprozesses

Zu Beginn jedes Zeitschrittes der laufenden Simulation werden neue Nahrungspartikel erzeugt. Danach geschieht bei jedem Organismus Folgendes:

1. *Lebensdauer:* Das Alter wird erhöht. Dies ermöglicht einen erzwungenen Tod von Organismen, sobald sie ein vorgegebenes Alter erreicht haben. Dieses Feature ist jedoch auch ausschaltbar, falls- wie bei einer Bakterienkolonie- keine Altersstерblichkeit vorkommt.[4]

2. *Anvisiertes Ziel:* Es wird entschieden, in welche Richtung sich der Organismus fortbewegen soll: Aus den Nahrungspartikeln, welche am nächsten sind und sich innerhalb des Wahrnehmungsradius befinden, wird zufällig ein Zielpartikel ausgewählt, zu welchem der Organismus sich dann hinbewegt. Der Einfachheit halber und wegen der rechnergestützten Optimierung nimmt der Organismus ohne Verzögerung wahr, wo sich diese Partikel befinden.

[3] Vgl. Mayr (1982).
[4] Vgl Moseley (2013)

3. *Fortbewegung:* Diese Bewegung erfolgt mit der durch ein Attribut festgelegten Geschwindigkeit. Hierbei wird darauf geachtet, dass sich Organismen nicht ausserhalb des beschränkten Simulationsraumes bewegen können.

4. *Nahrungsaufnahme (Verdauungseffizienz)*: Nahrungspartikel, welche sich nahe genug bei einem Organismus befinden, werden von diesem aufgenommen. Die Energie, welche zuvor in diesem Partikel gespeichert war, wird dann vom aufnehmenden Organismus integriert, d. h. sein Energiewert erhöht sich. Die Effizienz dieses Verdauungsprozesses kann ebenfalls durch ein Attribut gesteuert werden. Falls der Energietank des Organismus beinahe voll ist, und darum nicht die ganze Energie auf einmal aufgenommen werden kann, wird ein Partikel mit der überschüssigen Energie zurückgelassen.

5. *Energieverbrauch:* Verwendete Energie wird vom Energietank des Organismus reduziert. Um zu existieren, verbraucht der Organismus Energie. Hiermit sollen energieaufwändige Prozesse wie Zellwand-Reparaturen oder Proteinsynthesen simuliert werden. Zusätzlich wird auch die zur Fortbewegung verwendete Energie abgezogen.

6. *Vermehrung des Organismus:* Falls ein Organismus nach diesen Schritten noch einen gewissen konfigurierbaren Energiespiegel hat, teilt sich dieser und kreiert eine exakte Kopie von sich selbst. Die Energie, welche sich in dem Energietank befindet, wird zwischen den beiden neuen Organismen aufgeteilt. Zu welchen Teilen diese Aufteilung erfolgt, ist konfigurierbar.

7. *Lebensende des Organismus:* Hat ein Organismus am Ende dieser Schritte keine Energie mehr, stirbt dieser und verschwindet. Diese Vereinfachung dient wiederum der rechnergestützten Optimierung.

8. *Mutationen und Attribute:* Zum Schluss jedes Zeitschrittes gibt es für jeden Organismus eine kleine Chance, dass seine Genaktivierungen mutieren und seine Attribute neu berechnet werden.

2.2 Details

2.2.1 Erklärung der Attribute

Jeder Organismus hat verschiedene Attribute. Attribute verändern direkt das Verhalten eines Organismus, indem sie beispielsweise seine Geschwindigkeit, den Wahrnehmungsradius oder die Verdauungseffizienz bestimmen. Sie sind die Folge der verschiedenen Gen-Aktivierungen des Organismus. Standardwerte für Attribute können vor dem Start des Programms bestimmt werden. Falls ein Gen ein Attribut beeinflusst, werden diese Standardwerte jedoch überschrieben.

Folgende Attribute existieren:

Name	Beschreibung	Bereich
energyToExist	Die Energie, welche jedem Organismus für jeden Zeitschritt abgezogen wird.	0- 100
digestionEfficiency	Wie effektiv die Verdauung von Partikeln ist.	0- 100
splitEnergyDistribution	Wie die Energie bei der Teilung aufgeteilt wird	0- 100
color	Beeinflusst nur das Aussehen des Organismus	0- 255
pickUpRange	In welchem Radius Partikel absorbiert werden.	0- ∞
visionRange	In welchem Radius Partikel wahrgenommen werden können.	0- ∞
visionCount	Eine Kreatur wählt ein Ziel-Partikel aus den nächstgelegenen Partikeln aus. "visionCount" bestimmt, aus wie vielen Partikeln ausgewählt wird.	1 - ∞
lifespan	Die Lebensdauer eines Organismus.	0- ∞
splitAtEnergy	Sobald dieses Energielevel erreicht ist, teilt sich der Organismus.	0- 100
movementEfficiency	Wie viel Energie pro Zeitschritt für die zurückgelegte Distanz verwendet wird.	0- ∞
speed	Die Geschwindigkeit, mit der sich ein Organismus fortbewegen kann.	0- ∞

Tabelle 1: Verwendete Attribute und deren Eigenschaften

2.2.2 Erklärung der Gene

Gene können bei allen Organismen verschieden stark aktiviert sein. Diese Gen-Aktivierung bestimmt die Ausprägung der mit diesem Gen gekoppelten Attribute. Wie viele Gene jeweils simuliert werden und welche Einflüsse deren Aktivierung auf verschiedene Attribute haben, kann vor dem Start des Programmes konfiguriert werden.

Es können jeweils mehrere Gene gleichzeitig simuliert werden, auch wenn sie Attribute unterschiedlich beeinflussen. Über ein Gen kann also wie von einer mathematischen Funktion gedacht werden, wobei die individuelle Aktivierung des Gens das Argument ist und die verschiedenen Attribute mit dem Funktionswert vergleichbar sind. Lineare, inverslineare sowie quadratische Beziehungen sind möglich.

Welche Attribute auf welche Weise an die Aktivierung eines Genes gekoppelt werden, ist wie erwähnt konfigurierbar. Während die Gene gleichbleiben, ändert sich die individuelle Aktivierung der Gene durch Mutationen bei einzelnen Organismen im Laufe der Simulation. Diese Mutationen sind vergleichbar mit DNA-Mutationen, welche zum Beispiel durch Strahlungseinfluss erfolgen können. Um Umgebungen mit mehr oder weniger Strahlung darstellen zu können, kann die globale Mutationswahrscheinlichkeit vor aber auch während der Simulation verändert werden. Bei jedem Gen kann zusätzlich variiert werden, wie anfällig dieses auf Mutationen ist und wie stark sich die Gen-Aktivierung verändern kann. Dadurch folgen Veränderungen in den Attributen, was wiederum die Überlebens- und Reproduktionschance natürlich beeinflusst.

Die Aktivierung von Genen variiert von 0 bis 100 und es können ein oder mehrere Ausgangswerte definiert werden. Um sich davon ein besseres Bild machen zu können, was

eine stärkere Aktivierung eines Gens beeinflusst, können diese im Programm als graphische Zusammenhänge dargestellt werden.

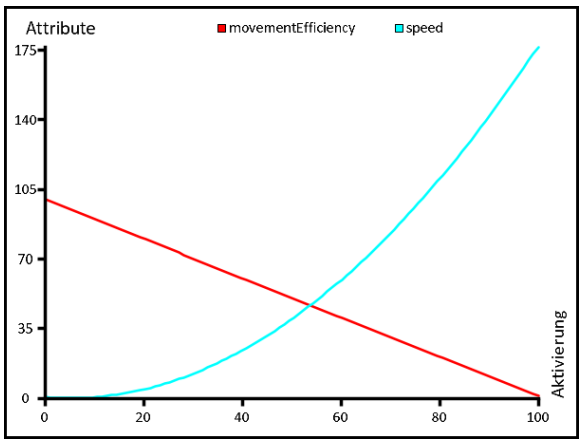


Abbildung 2: Beispiel eines Gens, welches die beiden Attribute "movementEfficiency" und "speed" verändert

2.2.3 Erklärung der Nahrungserzeugung

Neue Nahrungspartikel werden zu Beginn jedes Zeitschrittes erzeugt. Die Anzahl neuer Partikel wird wie folgt bestimmt:

*Anzahl Vorhandene Partikel * Reproduktionswert = Anzahl neue Partikel*

So wird die vereinfachte natürliche Reproduktion eines anderen Einzellers, beispielsweise einer Alge, simuliert. Ohne andere Einflussfaktoren wächst so die Partikelzahl exponentiell. Um zu verhindern, dass die Nahrungsquelle ganz ausstirbt, kann zusätzlich zum Reproduktionswert eine Mindestmenge an Partikeln festgelegt werden. Um eine ausufernde, exponentiell wachsende Partikelpopulation zu vermeiden, kann auch eine Maximalmenge festgelegt werden. Dies simuliert vereinfacht die Konkurrenz unter den Nahrungspartikeln.

Sowohl der Reproduktionswert als auch die Mindest- und Maximalmengen gelten nicht zwingend überall; es lassen sich mehrere Bereiche konfigurieren, in denen verschiedene Verhältnisse herrschen. Ein sonnigerer Teil kann in der Simulation beispielsweise mit einem höheren Reproduktionsfaktor modelliert werden. Somit kann beobachtet werden, wie sich Organismen situationsspezifisch genetisch an ihr Umfeld anpassen.

2.2.4 Erklärung der Statistiken

Zu jedem Zeitpunkt während der Simulation können verschiedene Statistiken in Form von Graphen generiert und eingesehen werden. Die Anzeige der Graphen lässt sich beeinflussen. Ähnliche Graphen werden in demselben Koordinatensystem überlagert, während Ausschnitte verschiedener Graphen nebeneinander angezeigt werden. Damit lassen sich Zusammenhänge erkennen.

Die Zeitspanne der Darstellung lässt sich verändern. Zur Verfügung stehen folgende Graphen:

- Anzahl Organismen im zeitlichen Verlauf
- Anzahl Partikel im zeitlichen Verlauf
- Entwicklung der verschiedenen Genaktivierungen im zeitlichen Verlauf

Letzterer zeigt, wie sich die Aktivierung verschiedener Gene über die Zeitdauer der Simulation verändert. Ebenfalls ablesbar ist die Verteilung von Gen-Aktivierungen und deren Durchschnitt.

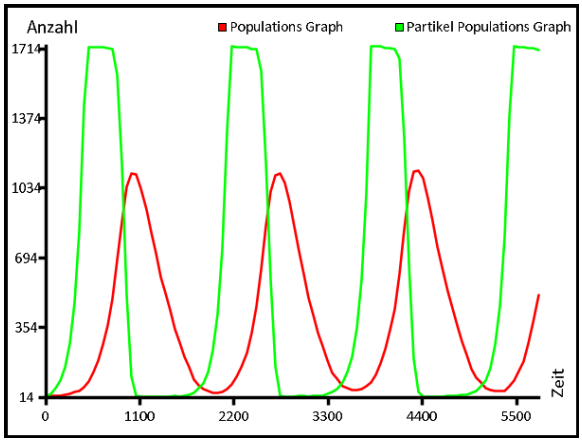


Abbildung 3: Beispiel für die Überlappung von Organismenmenge und Partikelmenge über der Zeit.[5]

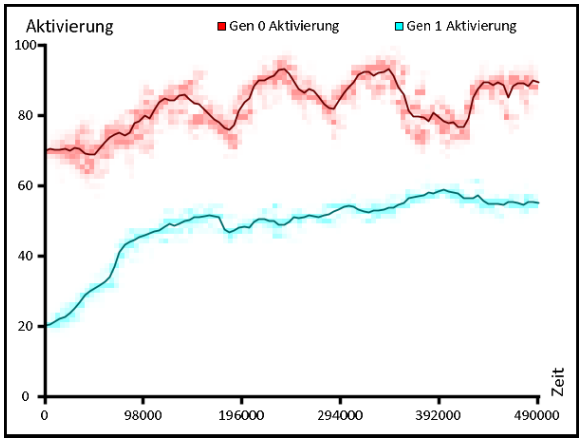


Abbildung 4: Beispiel für die Überlagerung von zwei verschiedenen Gen-Aktivierungen über der Zeit[6]

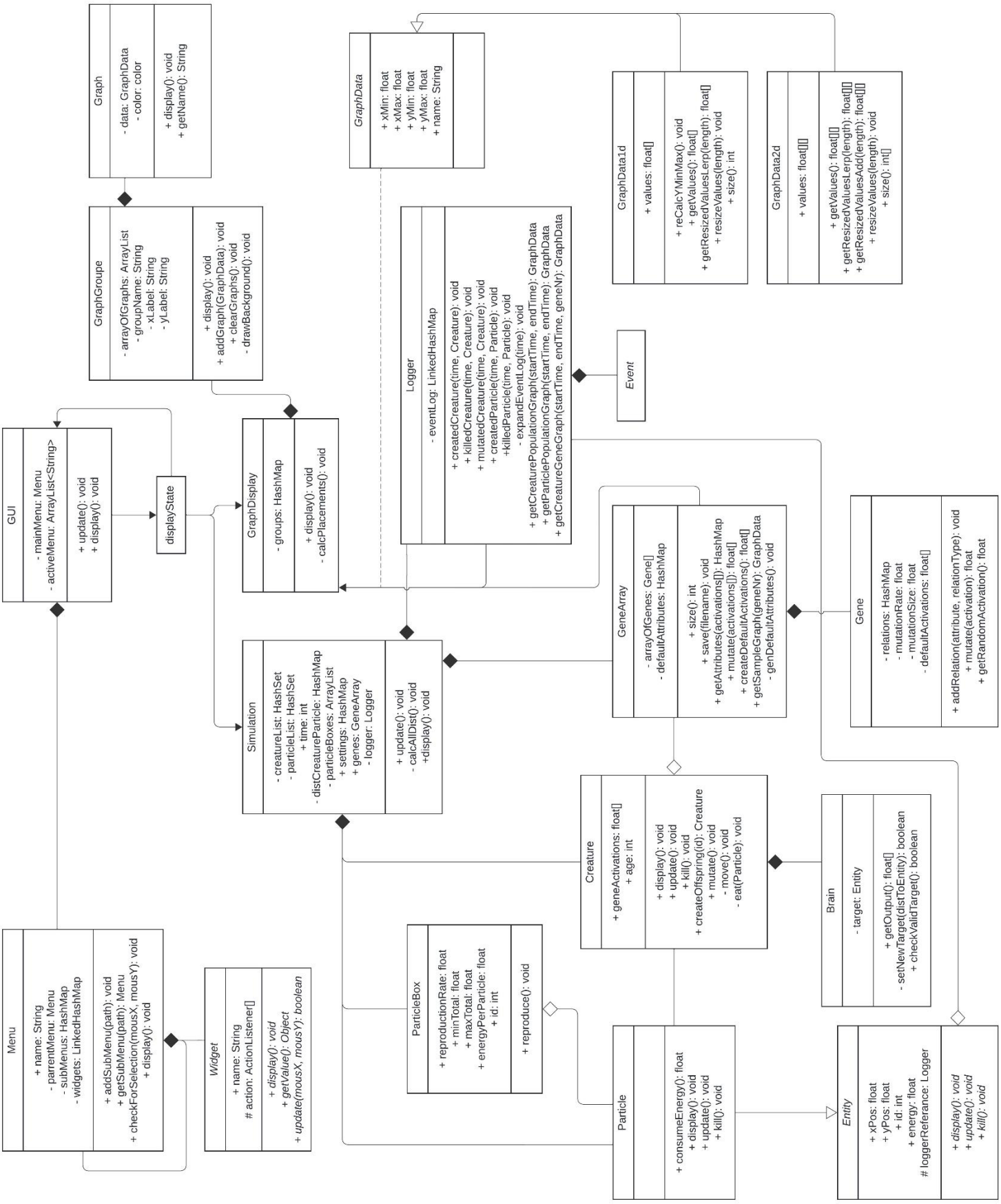
(Linie = durchschnittliche Aktivierung, farbige Bereiche = existierende Organismen mit der jeweiligen Aktivierung: je dunkler die Flächen, desto mehr Überlappungen).

2.3 Programmierung

2.3.1 Aufbau des Programms

Während die bisherigen Ausführungen jeweils auf biologische Aspekte Bezug nahmen, orientiert sich die folgende Darstellung vor allem an der informationstechnischen Umsetzung.

[5] Eigene Darstellung, generiert aus dem Programm.
[6] Eigene Darstellung, generiert aus dem Programm.


Graph
- data: GraphData
- color: color
+ display(): void
+ getName(): String
GraphGroupe
- arrayOfGraphs: ArrayList
- groupName: String
- xLabel: String
- yLabel: String
+ display(): void
+ addGraph(GraphData): void
+ clearGraphs(): void
- drawBackground(): void
GraphData
+ xMin: float
+ xMax: float
+ yMin: float
+ yMax: float
+ name: String
GraphData1d
+ values: float[]
+ reCalcYMinMax(): void
+ getValues(): float[]
+ getResizedValuesLerp(length): float[]
+ resizeValues(length): void
+ size(): int
GraphData2d
+ values: float[][]
+ getValues(): float[][]
+ getResizedValuesLerp(length): float[][]
+ getResizedValuesAdd(length): float[][]
+ resizeValues(length): void
+ size(): int[]
Logger
- eventLog: LinkedHashMap
+ createdCreature(time, Creature): void
+ killedCreature(time, Creature): void
+ mutatedCreature(time, Creature): void
+ createdParticle(time, Particle): void
+killedParticle(time, Particle): void
- expandEventLog(time): void
+ getCreaturePopulationGraph(startTime, endTime): GraphData
+ getParticlePopulationGraph(startTime, endTime): GraphData
+ getCreatureGeneGraph(startTime, endTime, geneNr): GraphData
Event
GUI
- mainMenu: Menu
- activeMenu: ArrayList<String>
+ update(): void
+ display(): void
displayState
GraphDisplay
- groups: HashMap
+ display(): void
- calcPlacements(): void
Simulation
- creatureList: HashSet
- particleList: HashSet
+ time: int
- distCreatureParticle: HashMap
- particleBoxes: ArrayList
+ settings: HashMap
+ genes: GeneArray
- logger: Logger
+ update(): void
- calcAllDist(): void
+display(): void
GeneArray
- arrayOfGenes: Gene[]
- defaultAttributes: HashMap
+ size(): int
+ save(filename): void
+ getAttributes(activations[]): HashMap
+ mutate(activations[]): float[]
+ createDefaultActivations(): float[]
+ getSampleGraph(geneNr): GraphData
- genDefaultAttributes(): void
Gene
- relations: HashMap
- mutationRate: float
- mutationSize: float
- defaultActivations: float[]
+ addRelation(attribute, relationType): void
+ mutate(activation): float
+ getRandomActivation(): float
Creature
+ geneActivations: float[]
+ age: int
+ display(): void
+ update(): void
+ kill(): void
+ createOffspring(id): Creature
+ mutate(): void
- move(): void
- eat(Particle): void
Brain
- target: Entity
+ getOutput(): float[]
- setNewTarget(distToEntity): boolean
+ checkValidTarget(): boolean
Menu
+ name: String
- parrentMenu: Menu
- subMenus: HashMap
- widgets: LinkedHashMap
+ addSubMenu(path): void
+ getSubMenu(path): Menu
+ checkForSelection(mousX, mousY): void
+ display(): void
Widget
+ name: String
# action: ActionListener[]
+ display(): void
+ getValue(): Object
+ update(mousX, mousY): boolean
ParticleBox
+ reproductionRate: float
+ minTotal: float
+ maxTotal: float
+ energyPerParticle: float
+ id: int
+ reproduce(): void
Particle
+ consumeEnergy(): float
+ display(): void
+ update(): void
+ kill(): void
Entity
+ xPos: float
+ yPos: float
+ id: int
+ energy: float
# loggerReferance: Logger
+ display(): void
+ update(): void
+ kill(): void

Legende:

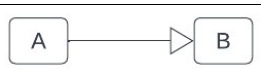
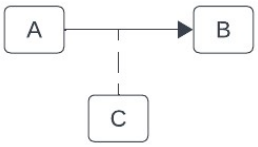
A extends B	
A has one or several instances of B	
B cannot exist without A	
B can exist without A	
Informationflow / A determines [something] of B	
An object of class C is passed from A to B	

Abbildung 5: Class Diagramm des Simulations-Programmes[7]

Klassenbeschreibung:[8]

Simulation	Das Haupt-Simulationsobjekt. Hierauf können die Methoden update() und display() aufgerufen werden.
Entity	Die abstrakte Klasse Entity funktioniert als superclass für alle sich in der Simulation befindenden Objekte.
Creature	Organismen sind intern als instances der Creature class dargestellt. Creature erweitert Entity.
Particle	Nahrungspartikel sind Instanzen der Particle class. Particle erweitert Entity.
GUI	Hier befinden sich Menus und Widgets für die Navigation durch das Programm.
Logger	Die Logger class ist zuständig für das Aufzeichnen von Ereignissen und kann diese als GraphData Objekte zum GraphDisplay weitergeben.
GraphData	Speichert Graphen und bietet Methoden an, um diese Daten umzuformatieren.
GraphDisplay	Zeigt GraphData Objekte in mehreren Gruppen an.
GeneArray	Speichert die verschiedenen Gene und bietet Methoden zur Interaktion mit diesen an.

Tabelle 2: Klassenbeschreibung

[7] Erstellt mit https://lucid.app. Eigene Darstellung.

[8] Eine detailliertere Beschreibung der Klassen und deren Methoden findet sich in Form von Javadoc-Kommentaren im Sourcecode im Anhang.

2.3.2 Distanzberechnungen zwischen vielen Punkten (Optimierungen)

2.3.2.1 Optimierungsinhalt

Damit entschieden werden kann, ob sich ein Partikel innerhalb des Wahrnehmungsradius eines Organismus oder ob es sich sogar nahe genug bei ihm befindet, damit er es absorbieren kann, muss die Distanz zwischen dem Organismus und jedem der Partikel berechnet werden, und zwar für alle Organismen. Es handelt sich also um n^2 Operationen, welche alle die Berechnung einer Quadratwurzel benötigen. Die grosse Häufigkeit der Operationen und das Ziehen der Quadratwurzel machen den Prozess sehr rechenaufwändig und damit langsam. Dies ist ein häufig auftretendes Problem bei agentenbasierenden Simulationen.

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2.3.2.2 Wie kann der Rechenprozess verschnellert werden?

Weglassen der Wurzelfunktion:

Da das Berechnen von Wurzeln sehr resourcenaufwändig ist, bietet sich als erster Schritt zu einer Optimierung das Weglassen dieser Wurzelfunktion an. So erhalten wir bei der Distanzberechnung nicht mehr direkt die Distanz, sondern deren Quadrat.

$$D^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

Da jeweils nur verglichen wird, ob die Distanz zu einem Partikel in einem gewissen Rahmen liegt und die Wurzelfunktion streng monoton wachsend ist, können diese Vergleiche immer noch durchgeführt werden. Das Quadrat der Distanz muss dann jedoch mit dem Quadrat der Rahmenbedingung verglichen werden.

Alt: if(DistanzZuPartikel ≤ Wahrnehmungsradius) {…}

Neu: if(DistanzZuPartikelQuadrat ≤ Wahrnehmungsradius^2) {…}

Spatial hashing:[9]

Bei grösseren agentenbasierenden Simulationen wird vielfach eine Technik namens "spatial hashing" verwendet, damit Distanzen zu sehr weit entfernten, irrelevanten Agenten nicht berechnet werden müssen. Hierbei wird der Simulationsuntergrund mit den sich darauf befindenden Agenten (in diesem Falle Organismen und Partikel) in Quadrate eingeteilt. Da sich Agenten bewegen können, muss diese Einteilung einmal zu Beginn jedes Zeitschrittes gemacht werden.

[9] Vgl. Hastings et al (2005).

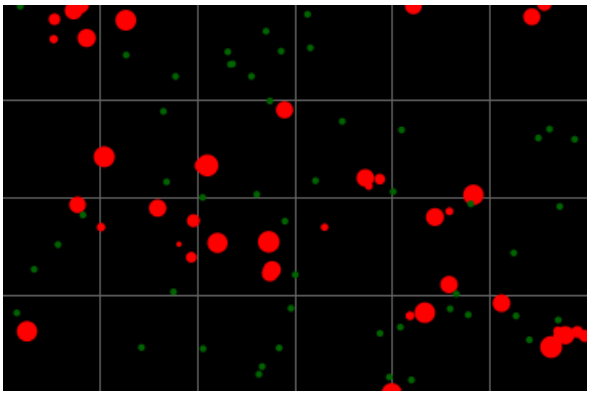


Abbildung 6: Organismen und Partikel auf einem unterteilten Simulationsuntergrund. [10]

Die Grösse dieser Quadrate wird so gewählt, dass eine Seite eines Quadrates dem Wahrnehmungsradius entspricht.

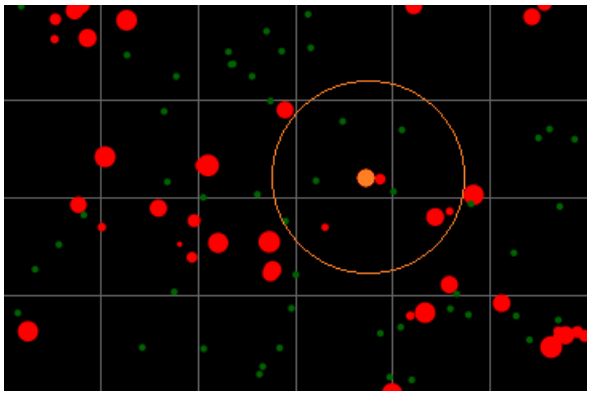


Abbildung 7: Die angepasste Grösse der Quadrate führt zu einer Reduktion der Berechnungen. [11]

Es ist zu beobachten, dass der Wahrnehmungsbereich niemals die neun Felder, welche den Agenten umgeben, verlässt. So muss nur die Distanz zu den sich in diesen Feldern befindenden Agenten berechnet werden, da sich Agenten in aussenliegenden Feldern nie im Wahrnehmungsbereich befinden können.

Erweitertes Spatial hashing

Eine weitere Optimierung lässt sich aus einer Verkleinerung der Quadrate und einem schrittweisen Vorgehen von innen nach aussen erzielen.

Im folgenden Abschnitt werde ich den Agenten, dessen wahrgenommene Agenten gerade bestimmt werden, als $Agent_M$ bezeichnen und alle anderen als $Agent_A$.

Bei einem grossen Wahrnehmungsradius werden auch nach der Implementierung von "spatial hashing" die Distanzen zu vielen $Agenten_A$ berechnet, welche sich nicht im Wahrnehmungsbereich befinden. Dies liegt daran, dass die Fläche der neun Quadrate $9/\pi$ mal grösser ist als die Fläche des Kreises in dem $Agenten_A$ wahrgenommen werden. Mehrere kleine Quadrate, welche zusammen die grobe Form eines Kreises bilden, wären optimaler als ein grosses Quadrat aus neun kleineren Quadraten.

[10] Eigene Darstellung, generiert mit dem Programm.
[11] Eigene Darstellung, generiert mit dem Programm (ausser Kreis).

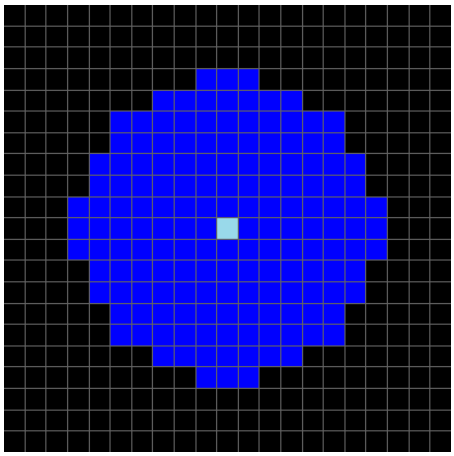


Abbildung 8: Optimierung aus einer Verkleinerung der Quadrate. [12]

Da ein $Agent_M$ nur eine gewisse Anzahl an am nächsten gelegenen $Agenten_A$ wahrnehmen kann, liegt ein weiterer Vorteil dieser Methode darin, dass die Suche nach validen $Agenten_A$ vom Zentrum aus gestartet und dann ausgeweitet werden kann, bis genügend $Agenten_A$ gefunden worden sind. Danach kann die Suche für diesen $Agenten_M$ abgebrochen und für den nächsten begonnen werden. Also wird die Fläche in Ringe aufgeteilt, welche immer auf dem Quadrat des aktuellen Agenten zentriert sind und immer grösser werden. Da keine Quadrate zweimal überprüft werden sollen, dürfen sich diese Ringe nicht überschneiden.

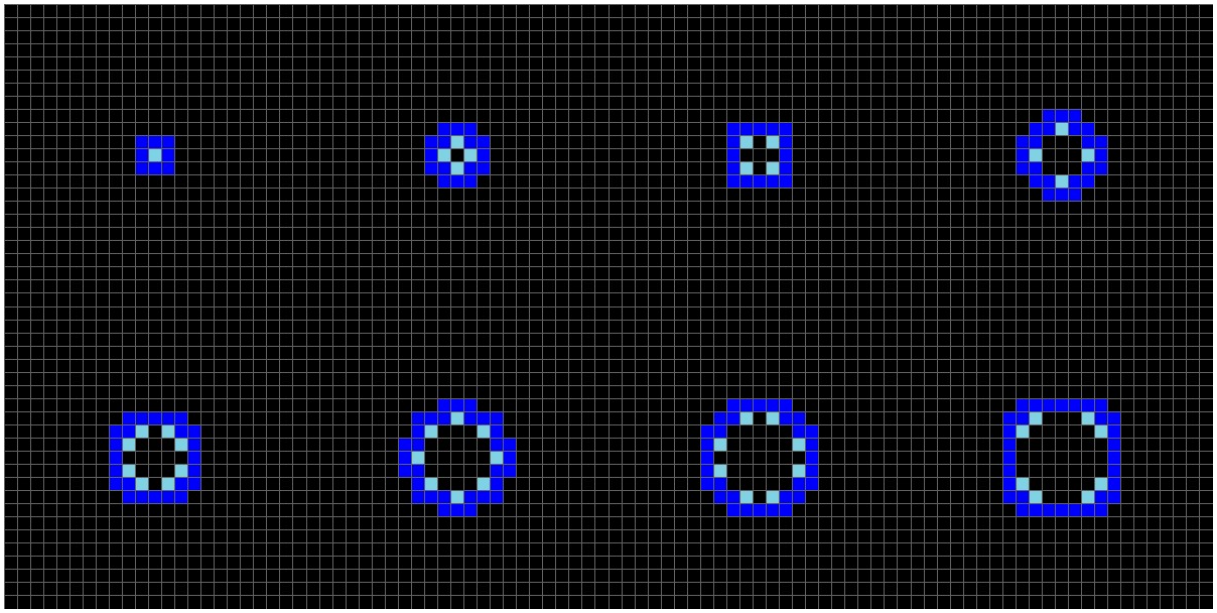


Abbildung 9: Optimierung aus einer Aufteilung der Fläche in Ringe.Dies erlaubt ein schrittweises Vorgehen von innen nach aussen: die ersten acht Ringe. [13]

Wie man sieht, überlappen sich die hellblauen Quadrate nie. Nach $Agenten_A$ untersucht werden immer zuerst die hellblauen Quadrate, und zwar in etappenweise sich vergrössernden

[12] Eigene Darstellung, auf der Basis des Programms und mit Hilfe einer separaten Programmierung für die Entwicklung der Ringe.
[13] Eigene Darstellung, auf der Basis des Programms und mit Hilfe einer separaten Programmierung für die Entwicklung der Ringe.

Ringen (vgl. Abb. xxx, in der Reihenfolge von links nach rechts, bis zum Wahrnehmungsradius). Finden sich auf diese Art genügend Agenten, wird der Prozess abgebrochen.

Es kann jedoch sein, dass sich innerhalb der dunkelblauen Quadrate ein $Agent_A$ befindet, der dem $Agenten_M$ näher liegt als ein bereits gefundener $Agent_A$ in einem hellblauen Quadrat (vgl. Abb. 10). Dieses Problem entsteht nur, falls das Muster noch nicht bis zum Wahrnehmungsradius ausgeweitet worden ist.

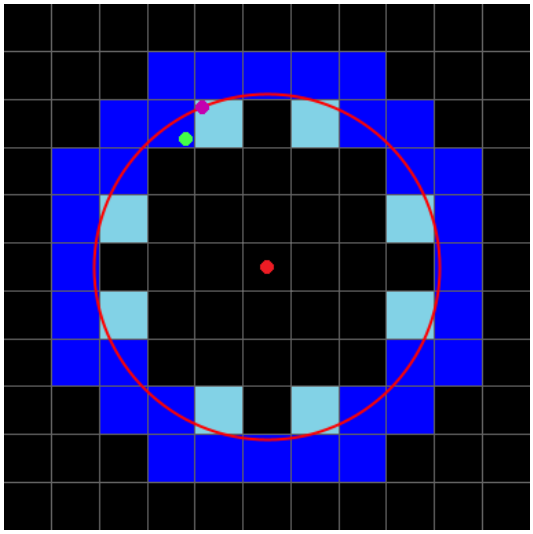


Abbildung 10: Problem bei der Suche nach den nächsten Agenten.

Legende: Zu sehen ist der sechste Ring um das Quadrat in dem sich der $Agent_M$ (rot) befindet. Auf dem roten Kreis in einem hellblauen Quadrat liegt ein $Agent_A$ (violett) welcher vom zentralen $Agenten_M$ weiter entfernt liegt als der grüne $Agent_A$.

Um dieses Problem zu lösen, werden zusätzlich die dunkelblauen Quadrate erforscht, welche zum letzten untersuchten Ring gehören.

Diese Methode erlaubt zusätzlich einen unbeschränkten Wahrnehmungsradius, da dieses Muster theoretisch unendlich weitergeführt werden kann. Der grösste Vorteil ist jedoch, dass die Quadratgrösse nun frei und dynamisch angepasst werden kann. Dies ermöglicht es bei einer tiefen Agentenkonzentration rasch viel Fläche abzudecken, während bei einer hohen Konzentration weniger unnötige Distanzen berechnet werden.

3 Schlussfolgerung

In der vorliegenden Arbeit wurde versucht, ein Programm zu entwickeln, das es dem Nutzer erlaubt, unterschiedliche evolutionäre Prozesse zu simulieren, indem folgende Parameter verändert werden können:

- Standardattribute: Anfangswert der Attribute zu Beginn der Simulation
- Grösse des Simulationsbereichs
- Häufigkeit von Mutationen
- Ort und Anzahl erzeugte Partikel
- Gene: ein Gen oder mehrere Gene gleichzeitig, sowie Anzahl Gene
- Eigenschaften der Gene: zu welchem Attribut welches Gen welche Beziehung hat

Dieses Ziel wurde erreicht.

Ein weiteres Ziel bestand darin, ein anwenderfreundliches Programm zu schreiben, das verständlich und einfach zu bedienen ist. Die Bedienung des Programmes ist intuitiv über Schaltflächen gelöst. Weniger benutzerfreundlich ist die Eingabe der oben erwähnten Simulationsparameter. Als Hilfestellung dient hierbei das Dokument README.txt[14], welches sich in demselben Ordner wie das Programm befindet. Die Datei erklärt die Nutzung des Programms Schritt um Schritt.

Das Ziel eines verständlichen Aufzeigens der Resultate wurde mittels Graphen gelöst, die sich über die Schaltfläche «Statistik» anzeigen lassen.

Es war nicht Ziel dieser Arbeit, die Simulationen zu interpretieren. Diese Arbeit überlasse ich gerne dem künftigen Nutzer.

4 Dank

Während dem Entwickeln des Programms konnte ich mir viel Wissen im Bereich der Biologie und der Informatik (Java-Programmiersprache) aneignen. Das war eine Herausforderung, die mir viel Spass gemacht hat. Somit danke ich insbesondere meinen beiden Betreuern, Herrn Christoph Gasche (Informatik) und Herrn Marco Perrig (Biologie), und meiner Familie, die mich in dieser Zeit unterstützt hat.

[14] Siehe Anhang 1.

5 Literaturverzeichnis

Hastings et al. (2005): Hastings, Erin J.; Mesit, Jaruwan; Guha, Ratan K.: Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing, publiziert auf: https://www.researchgate.net/publication/228958917_Optimization_of_large-scale_real-time_simulations_by_spatial_hashing, 2005, (zuletzt besucht: 26.11.23)

Mayr (1982): Mayr, Ernst: The Growth of Biological Thought, Diversity, Evolution, and Inheritance, 12. Aufl., Cambride, Massachusetts, London 1982, zit. n. https://en.wikipedia.org/wiki/On_the_Origin_of_Species, (zuletzt besucht: 26.11.23)

Moseley (2013): James B. Moseley: Cellular Aging: Symmetry Evades Senescence, 2013 Elsevier Ltd. Published by Elsevier Inc. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4276399/ (zuletzt besucht: 26.11.23)

Oracle (2023): Oracle JDK 21 Documentation, on https://docs.oracle.com/en/java/javase/21/ Austin 2023, (zuletzt besucht: 26.11.23)

6 Abbildungs- und Tabellenverzeichnis

7 Authentizitätserklärung

Ich bezeuge mit meiner Unterschrift, dass ich diese Arbeit selbstständig verfasst und erlaubte Hilfen von Drittpersonen korrekt veranschaulicht habe.

Informationen, die ich wörtlich oder sinngemäss von anderen Publikationen oder Quellen, unter anderem aus dem Internet oder mithilfe künstlicher Intelligenz (KI) verfasst habe, sind klar, wiederauffindbar zitiert und machen nur einen geringen Anteil der Arbeit aus. Die in meiner Arbeit benützten Hilfsmittel entsprechen der Wahrheit, sind vollständig aufgelistet und auf diese Weise noch nicht veröffentlicht worden. Mir ist bewusst, dass ich unter Umständen zu den von mir gebrauchten Hilfsmitteln Stellung nehmen muss.

Ort und Datum: 30.11.2023

Unterschrift des Schülers: _____

8 Anhang 1

README:

Anleitung zur Verwendung:

Vor dem Start des Programms:

1. Wählen Sie in config/settings.json den gewünschten Simulationsaufbau aus.

Falls kein Ordner, den Sie als Simulationsaufbau verwenden möchten, existiert, können Sie einen neuen erstellen, indem Sie den Ordner "copyPasteVorlage" kopieren und umbenennen.

Der Namen dieses Ordners ist dann in config/settings.json bei "simulationSetup" einzutragen.

Anschliessend können Sie die darin enthaltenen JSON-Dateien nach Wunsch verändern. Beachten Sie, dass nicht alle Dokumente so, wie Sie in der "copyPasteVorlage" vorhanden sind, ausführbar sind.

Zum Editieren der JSON-Dateien kann theorethisch fast jeder Texteditor verwendet werden. Es empfiehlt sich jedoch ein Programm wie Visual-situdio-code.

2. Konfigurieren Sie die simulationSettings.

Hierfür öffnen Sie die JSON-Datei simulationSettings.json, die sich im zuvor erstellten bzw. ausgewählten Ordner befinden, und verändern Sie folgende Werte nach Belieben:

"stratingSize": (float) Um wieviel der Simulationsuntergrund grösser ist als der Bildschirm. Bei einem Wert von zwei ist dieser doppelt so gross.

"mutationProbability": (float 0- 100) Die chance dass für jede Kreatur an einem Zeitschritt eine Mutation statfindet.

"creatureStartingEnergy": (float 0- 100) Das Energielevel, das jede Kreatur zu Beginn der Simulation hat.

"displayParticleBoxes": (boolean) Ob Partikelboxen angezeiget werden sollen. Mehr zu Partikelboxen unter "particleBoxes".

"showDebugInfo": (boolean) Zeigt während dem die Simulation läuft zusätzlich Infos über diese an.

"particleBoxes": (JSONArray) In diesem JSONArray befinden sich ein oder mehrere JSONObjects, welche alle eine andere Partikelbox beschreiben.

Mehr Informationen zu JSONArrays / JSONObjects finden Sie zuunterst.

Partikelboxen haben folgende Parameter:

"xStart": (float 0- 1) Die x-Koordinate des Startpunktes relativ zur Simulationsbreite.

"yStart": (float 0- 1) Die y-Koordinate des Startpunktes relativ zur Simulationshöhe.

"xDimFraction": (float 0- 1) Die Breite der Box relativ zur Simulationsbreite.

"yDimFraction": (float 0- 1) Die Höhe der Box relativ zur Simulationshöhe.

"reproductionRate": (float) Die Anzal der Partikel, welche jeden Zeitschritt neu erschaffen werden, entspricht der Anzahl Partikel, welche zu dieser Partikelbox gehören, multipliziert mit diesem Faktor.

Hierbei werden unvollständige Partikel auf den nächsten Zeitschritt übertragen.

"minPerArea": (float * 10000) Minimum Anzahl Partikel, welche pro Fläche immer existieren. "reproductionRate" wird ignoriert bis dieses Minimum erreicht ist.

"maxPerArea": (float * 10000) Maximum Anzahl Partikel, welche pro Fläche existieren können. "reproductionRate" wird ignoriert sobald dieses Maximum erreicht ist.

"energyPerParticle": (float 0- 100) Energie, welche jedes Partikel enthält.

3. Konfigurieren Sie die defaultAttributes.

Hierfür öffnen Sie die JSON-Datei defaultAttributes.json und verändern Sie folgende Attribute nach Belieben: (es handelt sich hierbei immer um floats)

defaultAttributes werden überschrieben, falls diese von einem Gen beinflusst werden.

"energyToExist": (0- 100) Die Energie, welche jeder Kreatur für jeden Zeitschritt abgezogen wird.

"digestionEfficiency": (0- 100) Wie effizient die Verdauung von Partikel ist: 0-> keine Energei wird aufgenommen, 1-> die ganze verfügbare Energie wird aufgenommen.

"splitEnergyDistribution": (0- 100) Wie die Energie bei der Teilung (Vermehrung) aufgeteilt wird: 0-> die Eltern-Kreatur gibt alle Energie an die neue Kreatur ab, 1-> Die Eltern-Kreatur behält ihre vollständige Energie.

"color": (0- 255) Die Hue der Kreatur. Dieses Attribut hat nur visuelle Auswirkungen.

"pickUpRange": In welchem Radius Partikel aufgesammelt bzw. gegessen werden.

"visionRange": In welchem Radius Partikel gesehen werden können. Kann auf-1 gesetzt werden für unendliche visionRange

"visionCount": Eine Kreatur wählt ein Ziel-Partikel aus den am nächsten liegenden Partikeln aus. "visionCount" bestimmt, aus wie vielen Partikeln ausgewählt wird.

"lifespan": Die Anzahl Zeitschritte, welche eine Kreatur maximal überleben kann. Für unlimitierte lifespan setzen Sie diesen Wert auf-1.

"splitAtEnergy": (0- 100) Sobald dieses Energielevel erreicht ist, teilt sich eine Kreatur.

"movementEfficiency": Wie viel Energie jeden Zeitschritt für Bewegung verwendet wird. Verbrauchte Energie = zurückgelegte Distanz / movementEfficiency. Kann auf-1 gesetzt werden für unendliche Effizienz.

"speed": Die Geschwindigkeit, mit der sich eine Kreatur in der Simulation fortbewegen kann. Beachten Sie, dass die "speed" niemals "pickUpRange" überragen sollte, da ansonsten Partikel übersprungen werden können.

4. Konfigurieren Sie die Gene.

Die Gene befinden sich in der JSON-Datei genes.json. Bei dieser Datei handelt es sich um ein JSONArray, in der mehrere Gene in Form von JSONObjects gespechert sein können.

Ein einzelnes Gen ist folgendermassen aufgebaut:

"defaultActivation": (float 0- 100) Die Gen-Aktivierung, welche die Kreaturen zu Beginn der Simulation haben werden.

Alternativ zu einer Zahl kann auch ein Array angegeben werden, aus dem Gen-Aktivierungen zufällig ausgewählt werden. Z.B [20, 30, 45]-> Kreaturen bekommen zu Beginn zufällig eine dieser Aktivierungen zugeteilt.

"mutationRate": (float 0- 100) Bei diesen Wert handelt es sich um die prozentuale Wahrscheinlichkeit, dass die Aktivierung dieses Genes verändert wird, falls eine Kreatur zur Mutation ausgewählt wird.

"mutationSize": (float 0- 100) Bestimmt die maximale Grösse einer stattfindenden Mutation. Z.B. bei einer mutationSize von 4 und einer vormaligen Gen-Aktivierung von 39 wird der neue Wert minimal 35 und maximal 43 sein.

"relations" Hierbei handelt es sich um ein JSONArray, in welchem sich JSONObjekte befinden, welche bestimmen, welche Effekte die Gen-Aktivierung auf verschiedene Attribute hat. Sie sind folgendermassen aufgebaut:

"atribute": Bestimmt, welches Attribut verändert wird. Wählen Sie eines aus defaultAttributes.json aus.

"type": Entscheidet über die Art der Beziehung. Zur Verfügung stehen:

"linear": Attribut = Gen-Aktivierung * "multiply" + "offset"

"inverseLinear": Attribut = "multiply" / Gen-Aktivierung + "offset"

"quadratic": Attribut = (Gen-Aktivierung- "moveX")^2 * "multiply" + "offset"

"parameters": Ein JSONObject, in dem die zuvor erklärten Parameter ("multiply", "offset, "moveX") angegeben werden können. Bei deren Konfigurierung kann ein Tool wie Desmos helfen.

Alternativ können die Effekte von Genen auch im Programm unter Hauptmenu/Simulations Einstellungen/Gene dargestellt werden.

Beispiel für ein valides Gen, welches speed linear und die color quadratisch verändert:

```
{
  "defaultActivation": 30,
  "mutationRate": 40,
  "mutationSize": 2,
  "relations": [
    {
      "attribute": "speed",
```

```
      "type": "linear",
      "parameters": {
        "offset": 0,
        "multiply": 1
      }
    },
    {
      "attribute": "color",
      "type": "quadratic",
      "parameters": {
        "offset": 0,
        "multiply": 2.55,
        "moveX":-2
      }
    }
  ]
}
```


5. Starten Sie das Programm.

Falls das Programm während längerer Zeit nicht reagiert, ist vermutlich ein Fehler im Simulations-Aufbau.

Navigation im Menu der Programmes:

Bei erfolgreichem Start des Programmes ist das HauptMenu zu sehen. Dieses kann jederzeit durch das Drücken der ESC-Taste erreicht werden.

HauptMenu:

Simulation anzeigen: Zeigt die Simulationsfläche mit der in den Simulationseinstellungen vorgegebenen TPS (Simulationsgeschwindigkeit).

Simulation sprinten: Lässt die Simulation so schnell wie möglich laufen. Zeitschritte passieren eventuell nicht mehr immer mit dem selben Abstand.

Simulations Einstellungen:

Alle Kreaturen löschen: Selbsterklärend.

20 Kreaturen erzeugen: Erzeugt 20 Kreaturen mit einer zufälligen Position und den standard-Gen-Aktivierungen.

Ziel TPS: Reguliert die TPS beim "Simulation Anzeigen" Modus.

Partikelboxen anzeigen: Bei Aktivierung werden die Ränder von allen Partikelboxen angezeigt.

Debuginfo anzeigen: Bei Aktivierung werden während der laufenden Simulation zusätzliche Infos angezeigt:

FPS: Wieviele Male in der Sekunde die Simulation angezeigt wird.

TPS: Wieviele Male pro Sekunde ein Zeitschritt passiert.

Kreaturen: Die Anzahl lebendiger Kreaturen.

Partikel: Die Anzahl Partikel.

Zyklen: Die Anzahl Zeitschritte, welche seit Beginn der Simulatiuon simuliert wurden.

Distanz Berechnungen: Die Anzahl Distanz-Berechnungen, welche jeden Zeitschritt vorgenommen werden.

PartikelBox X

Erste Zahl: Minimum Partikel in dieser Partikelbox.

Zweite Zahl: Effektive Anzahl Partikel in dieser Partikelbox.

Dritte Zahl: Maximum Partikel in dieser Partikelbox.

Die zu sehenden grauen Quadrate spielen eine wichtige Rolle bei der Berechnung von Distanzen zwischen Partikeln und Kreaturen.

Falls eine limitierte "visionRange" eingestellt ist, wird diese als gleichfarbiger Kreis um eine Kreatur angezeigt.

Mutationshäufigkeit: Siehe "Vor dem Start" Nr. 2 "mutationProbability".

Nahrungs Einstellungen: Siehe "Vor dem Start" Nr. 2 "particleBoxes". Vergessen Sie nicht, nach der Konfiguration auf Anwenden zu klicken.

Gene: Zeigt Effekt von aktiven Genen auf Attribute an.

Statistik: Gewünschte Grafiken auswählen und auf Anzeigen klicken. Falls die Simulation schon lange läuft, kann die Erstellung dieser Graphen etwas Zeit benötigen.

Beginn und Endzeit können in den entsprechenden Sliders konfiguriert werden.

Achten Sie darauf, dass Graphen möglicherweise nicht immer bei (0, 0) beginnen.

Programm benenden: Selbsterklärend.

Während der Simulation:

Nachdem auf "Simulation anzeigen" oder "Simulation sprinten" geklickt worden ist, ist Folgendes zu sehen:

Kreaturen: Diese werden als farbige Kreise dargestellt. Die Farbe hängt hierbei vom Attribut "color" ab und die Grösse von dem aktuellen Energielevel der Kreatur.

Partikel: Partikel werden als dunkelgrüne, meist kleinere Punkte dargestellt. Auch ihre Grösse hängt von deren Energielevel ab.

Wie immer kann auch hier die ESC-Taste gedrückt werden, um zurück zum Hauptmenu zu gelangen. Die Simulation wird hierbei pausiert.

Um die Simulation zu pausieren, drücken Sie die Lehrtaste. Bei erneutem Drücken der Lehrtaste wird die Simulation fortgesetzt.

Information zu JSONObjects und JSONArrays:

JSON-Dateien können sowohl JSONObject als auch JSONArrays und verschachtelte Komninationen davon enthalten.

JSONObjects:

Ein JSONObject ist eine Datenstruktur, bei der Schlüssel-Wert-Paare gespeichert werden können. Hierbei müssen Schlüssel einzigartig sein.

Ein JSONObject ist wie folgt aufgebaut:

{ //Eine geschweifte Klammer zu Beginn des JSONObjects.

"Schlüssel1": Wert1, //Der Schlüssel steht in Anführungszeichen und besteht aus Buchstben (Text). Danach folgt ein Doppelpunkt und dann ein Wert.

"Schlüssel3": Wert3, //Zwischen zwei Schlüssel-Wert-Paaren wird ein Komma gesetzt. Die Reihenfolge dieser Paare spielt keine Rolle.

"Schlüssel2": Wert2 //Obwohl Formatierung optional ist, macht es das JSONObject übersichtlicher.

} //Eine geschweifte Klammer schliesst das JSONObject wieder.

JSONArrays:

Ein JSONArrayist eine Liste von Werten werden.

Ein JSONArray ist wie folgt aufgebaut:

[//Eine eckige Klammer zu Beginn des JSONArrays

Wert1, //Werte werden in korekter Reigenfolge und ohne jegliche Schlüssel aufgelistet.

Wert2, //Zwischen zwei Werten wird ein Komma gesetzt.

Wert3 //Obwohl Formatierung obtional ist, macht es den JSONArray übersichtlicher.

] //Eine eckige Klammer schliest den JSONArrays wieder.

Werte können nicht nur Zahlen oder Booleans (true / false) sein, sondern auch andere JSONObjects oder JSONArrays.

Dies macht eine komplexe Hirarchie möglich.

Obwohl JSON-Dateien mit den meisten Text-Editoren geschrieben und verändert werden können, empfiehlt sich ein Programm wie Visual-studio-code.

Beispiel einer komplexeren JSON-Datei:

```
[
  {
    "Schlüssel1.1": true,
    "Schlüssel1.2": 17.83
  },
  {
    "Schlüssel2.1": [
      10,
      11,
      false
    ],
    "Schlüssel2.2": 0
  },
  true,
  true,
  -28
]
```