



FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Ein Simulationssystem für die Wirkung von Schutzmechanismen
von Pflanzen auf Fressfeinde – Visualisierter Wettbewerb
zwischen kooperierenden und feindlichen Teams

MASTERARBEIT

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
im Studiengang Bioinformatik

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik

eingereicht von Lucas Dietrich
geb. am 16.01.1997 in Kandel

1. Gutachter: PD Dr.-Ing. habil. Thomas Hinze
2. Gutachter: Benjamin Förster, M. Sc.

Jena, 7. März 2025

Zusammenfassung

Pflanzen existieren schon seit Millionen von Jahren und haben über die Jahre hinweg ausgeklügelte Schutzmechanismen gegen Fressfeinde entwickelt, um ihre begrenzte Energie optimal zu nutzen.

Zur besseren Einordnung werden ausgewählte Schutzmechanismen vorgestellt, die sich in ihrer Ausprägung unterscheiden.

Es wird ein Modell entwickelt, zur Abbildung eines Biotops mit verschiedenen Pflanzen und Fressfeinden. Wie in der Natur kommunizieren die Pflanzen über Signalstoffe, die über die Luft oder ein unterirdisches Wurzelnetzwerk verbreitet werden. Um sich gegen Fressfeinde zu wehren, besitzen die Pflanzen vertreibende oder tödliche Giftstoffe, die im Modell berücksichtigt werden.

Zur Anwendung des Modells wurde eine intuitive graphische Oberfläche entwickelt, mit der Nutzer eigene Biotope erzeugen und simulieren kann.

Eine umfassende Simulationsstudie zeigt, wie sich die verschiedenen Populationen im Verlauf der simulierten Zeit entwickeln. Dabei werden Entwicklungen wie das Wachstum der Pflanzen oder der Fressfeinde sichtbar. Die Ergebnisse der Simulation veranschaulichen die Veränderungen in der Populationsdynamik und wie sich diese im Zeitverlauf durch die Interaktionen zwischen den verschiedenen Akteuren im Modell entwickeln.

Abstract

Plants exist for millions of years and have developed complex defense mechanisms against enemies to optimally use their limited energy.

For better context, selected defense mechanisms are presented, which vary in their manifestation.

A model is developed to represent a biotope with various plants and enemies. As in nature, the plants communicate via signaling substances that are transmitted through the air or an underground root network. To defend against enemies, plants produce dispersing or toxic substances, which are considered in the model.

To apply the model, an intuitive graphical interface was developed, allowing users to create and simulate their own biotopes.

A comprehensive simulation study shows the different population's evolution over the course of the simulation time. Developments, such as the growth of plant and enemy populations, become visible. The simulation results illustrate changes in population dynamics and how they evolve over time through the interactions between the various actors in the model.

Inhaltsverzeichnis

Zusammenfassung	2
Abstract	3
Abkürzungsverzeichnis	8
Symbolverzeichnis	9
1 Einführung	14
1.1 Hintergrund	14
1.2 Zielsetzung	16
1.3 Aufbau dieser Arbeit	16
2 Biologischer Hintergrund	17
2.1 Das Szenario in der Natur	17
2.2 Pflanzliche Schutzmechanismen	18
2.2.1 Warnfärbungen bei Pflanzen	18
2.2.2 Mimikry als Schutzmechanismus	21
2.2.3 Kommunikation zwischen Pflanzen	23
2.2.4 Interaktion zwischen Pflanzen	24
2.2.5 Symbiose zwischen Pflanzen und Pilzen	25
2.2.6 Interaktionen zwischen Pflanzen und Insekten	27
2.2.7 Glucosinolate-basierte Abwehr	29
2.2.8 Schutzmechanismen gegen Pathogene	29
3 Formale Beschreibung des Modells	31
3.1 Das Grid als Biotop	31
3.2 Die Akteure auf dem Grid	31
3.2.1 Die pflanzlichen Spieler	32
3.2.2 Die feindlichen Spieler	32
3.3 Struktur und Belegung der Gridfelder	33
3.3.1 Das dynamische Grid	33

3.3.2	Das Hinzufügen von Elementen	34
3.3.3	Das Entfernen von Elementen	35
3.4	Die Bewegung von Fressfeinden über das Grid	36
3.4.1	Die Suche nach der nächsten Pflanze	36
3.4.2	Bewegung entlang des Pfades	39
3.5	Energiehaushalt der Pflanze	40
3.5.1	Initiale Energieeinheit einer Pflanze	40
3.5.2	Energiezuwachs einer Pflanze	41
3.5.3	Mindestenergie zum Überleben einer Pflanze	41
3.5.4	Energieprozentsatz als Lebensanzeige für die Pflanzen	42
3.6	Die Nachkommen der Akteure	42
3.6.1	Die pflanzlichen Nachkommen	42
3.6.2	Die Fresserfolge und Nachkommen der Fressfeinde	43
3.7	Von Pflanzen produzierte Substanzen	47
3.7.1	Die Interaktionsmatrix der Signalstoffe	47
3.7.2	Bedingungen zur Freisetzung von Signalstoffen	49
3.7.3	Mechanismen der Signalstoff-Ausbreitung	50
3.7.4	Die ungezielte Ausbreitung via Luft	51
3.7.5	Die gezielte Ausbreitung via Gridverbindung	52
3.7.6	Start und Beendigung der Signalstoffproduktion	54
3.7.7	Die Nachwirkzeit und der Wirkbereich eines Signalstoffs	55
3.7.8	Die Interaktionsmatrix der Giftstoffe	57
3.7.9	Der Start der Giftstoffproduktion	59
3.7.10	Die Wirkung der Giftstoffe	59
3.7.11	Der Einfluss auf die Fressfeinde durch Giftstoffe	60
3.7.12	Ende der Giftproduktion und Auswirkungen	62
3.7.13	Kosten für die Produktion einer Substanz	63
3.8	Gesamtenergiebilanz der Pflanze	64
3.9	Die Simulation und ihre Abbruchbedingungen	65

4 Implementierung des Modells	68
4.1 Die Pflanze	69
4.2 Die Feinde und ihre Fressfeindcluster	73
4.3 Die Substanzen	77
4.3.1 Die Klasse der Signalstoffe	78
4.3.2 Die Klasse der Giftstoffe	80
4.4 Die Gridverbindung zwischen zwei Pflanzen	82
4.5 Das Grid und die Simulation	83
4.5.1 Initialisierung des Grids	84
4.5.2 Hinzufügen und Entfernen von Klassen-Objekten	85
4.5.3 Visuelle Darstellung des Grids in der Konsole	86
4.5.4 Handhabung der Fressfeindcluster im Grid	87
4.5.5 Interaktionen zwischen Fressfeindclustern und Pflanzen	88
4.5.6 Substanzproduktion bei Interaktion	90
4.5.7 Substanzeffekte auf die Fressfeinde	92
4.5.8 Zurücksetzen der Substanzverfügbarkeit	93
4.5.9 Handhabung der Kommunikation zwischen Pflanzen	94
4.5.10 Verbindung zwischen der Grid- und der Simulations-Klasse . .	95
4.6 Der Dateimanager	97
4.6.1 Die Klasse zum Exportieren	98
4.6.2 Die Klasse zum Importieren	99
4.7 Die graphische Oberfläche und ihre Bedienung	100
4.7.1 Der Aufbau des Hauptfensters	100
4.7.2 Bedienungsanleitung zur Verwendung des Tools	102
5 Simulationsstudie	111
5.1 Konfigurierung des Netzwerks	111
5.2 Ergebnisse der Simulation	113
5.2.1 Entwicklung der Pflanzenpopulation über die Zeit	113
5.2.2 Entwicklung der Fressfeindpopulation über die Zeit	114

6 Diskussion und Schlussfolgerung	116
6.1 Diskussion der Ergebnisse	116
6.2 Schlussfolgerungen dieser Arbeit	120
Literaturverzeichnis	126
Abbildungsverzeichnis	127
Tabellenverzeichnis	128
Quelltextverzeichnis	129
Anlagen	130
Eigenständigkeitserklärung	136
Sonstiges	137

Abkürzungsverzeichnis

bool	Boolean
ET	Ethylen
ETI	Effektor-getriggerte Immunität (Effector-Triggered Immunity)
HAMP	Herbivor-assoziiertes molekulares Muster (Herbivore-Associated Molecular Pattern)
int	Integer
ISR	Induzierte systemische Resistenz (Induced Systemic Resistance)
JA	Jasmonat
MVC	Model-View-Controller
PTI	Muster-getriggerte Immunität (Pattern-Triggered Immunity)
ROS	Reaktive Sauerstoffspezies (Reactive Oxygen Species)
SA	Salicylsäure
SAR	Systemisch erworbene Resistenz (Systemic Acquired Resistance)
VAM	Versikuläre-arbuskuläre Mykorrhiza
VOC	Flüchtige organische Verbindung (Volatile Organic Compounds)
WSN	Drahtloses Sensornetzwerk (Wireless Sensor Network)

Symbolverzeichnis

(x_e, y_e)	Zielposition im kürzesten Pfad
(x_k, y_k)	k -te Position im kürzesten Pfad
(x_p, y_p)	Aktuelle Position im kürzesten Pfad
$\alpha(s_x)$	Funktion, die die Fressfeindarten bestimmt, die auf den Giftstoff reagieren
$\beta(s_x, C_i)$	Eliminationsrate des Giftstoffs
Δt_i	Zeitintervall, das vorgibt, wann ein Fressfeindcluster sich bewegt
$\delta(C_i, t)$	Fokus eines Fressfeindclusters zu einem Zeitpunkt
$\eta(C_i)$	Essgeschwindigkeit des Fressfeindclusters
Γ	Menge der Bewegungsrichtungen
γ	Spezifische Bewegungsrichtung aus Γ
\mathcal{A}	Emissionsmatrix, die zeigt, welche Pflanzenart welchen Signalstoff produzieren kann
\mathcal{B}	Empfängermatrix, die zeigt, welche Pflanzenart welchen Signalstoff empfangen kann
$\mathcal{E}_{i,j}(t)$	Energie der Pflanze $p_{i,j}$ zum Zeitpunkt t
$\mathcal{E}_{min}(e_h)$	Mindestanzahl an verzehrten Energieeinheiten der Fressfeindart e_h
$\mathcal{F}(p_{i,j}^*, t)$	Funktion, die überprüft, ob die Giftstoffproduktion noch nicht beendet ist
$\mathcal{J}(s_x)$	Interaktionsmenge eines Giftstoffs
\mathcal{M}	Funktion für die Bewegung des Fressfeindclusters
\mathcal{R}	Zufallsfunktion
$\mathcal{T}(s_k)$	Anzahl benötigter Zeitschritte, bis der Signalstoff verfügbar ist
$\mathcal{T}(s_x)$	Anzahl der Zeitschritte, die für die Giftstoffproduktion benötigt werden
$\mathcal{V}(p_{i,j}, p_{k,m})$	Symbiotische Verbindung zwischen $p_{i,j}$ und $p_{k,m}$
$\mathcal{W}(s_x, C_i)$	Wirkung des Giftstoffs auf das Fressfeindcluster

$\mathcal{Z}(C_i, p_{k,l}, t)$	Funktion, die beschreibt, ob die Pflanze nach der Beendigung der Giftstoffproduktion wieder als Ziel verfügbar ist
μ	Neue Position
Ω	Kürzester Pfad
$\phi(e_h, t)$	Anzahl der neu erzeugten Nachkommen zum Zeitpunkt t
Π	Vorgängerabbildung
Ψ	Distanzabbildung
ρ	Endpunkt eines kürzesten Pfades
σ	Startpunkt eines kürzesten Pfades zu einer Pflanze
$a_{i,k}$	Eintrag der Matrix \mathcal{A} für die Pflanzenart p_i und den Signalstoff s_k
$B_{i,j}$	Mindestenergie, die eine Pflanze benötigt, um zu überleben
$b_{i,k}$	Eintrag der Matrix \mathcal{B} für die Pflanzenart p_i und den Signalstoff s_k
C	Menge aller Fressfeindcluster
$c(s_i)$	Kosten für die Produktion einer Substanz
C_i, C_j, C_m	Fressfeindcluster der Fressfeindart e_h mit der Größe n
D	Platzhalter für die Fressfeindcluster, die sich tatsächlich auf einem Gridfeld befinden
$d_g((x, y), (x^*, y^*))$	Manhattan-Distanz zwischen zwei Positionen
d_E	Euklidische Distanz
d_{max}	Maximale Distanz
d_{min}	Minimale Distanz
E	Die Menge der Fressfeindarten
f_r	Funktion, die überprüft, ob ein Signalstoff ausgelöst wird oder nicht
f_s	Funktion, die die Menge der Signalstoffe auf die Menge der Ausbreitungsmöglichkeiten abbildet
G	Das Grid
$g(t)$	Maximale Anzahl an fortlaufenden Gridverbindungen zu einem Zeitpunkt

$G'_g(t, s_k)$	Menge aller Gridfelder, auf denen eine Pflanze steht, die den Signalstoff empfangen kann
$G_g(t)$	Menge aller Gridfelder innerhalb der Reichweite, die über eine Gridverbindung erreicht werden können
$G'_l(t, s_k)$	Untermenge aller Gridfelder, auf denen eine Pflanze steht, die den Signalstoff senden und empfangen kann
$G_l(t)$	Menge aller Gridfelder innerhalb des Signalstoffradius
I_{sig}	Interaktionsmatrix der Signalstoffe
$I_{tox}(s_x)$	Interaktionsmatrix des Giftstoffs s_x
L	Menge aller Gridfelder innerhalb des Radius
l	Funktion, die die Belegung eines Gridfeldes zu einem Zeitpunkt abbildet
$l((x^*, y^*), t^*)$	Position der auslösenden Pflanze zu einem Zeitpunkt
L^*	Untermenge aller Gridfelder innerhalb des Radius, die keine Pflanzen enthalten.
M	Menge der Ausbreitungsmöglichkeiten eines Signalstoffs
$n(t)$	Clustergröße zum Zeitpunkt t
$n(t + 1)$	Clustergröße zum Zeitpunkt $t + 1$
$n_c(t + 1)$	Anzahl der Individuen im Tochtercluster nach der Teilung zum Zeitpunkt $t + 1$
$N_{i,j}$	Anzahl der Nachkommen der Pflanze $p_{i,j}$
$O_k(t)$	Menge der Pflanzen, die den Signalstoff zu dem Zeitpunkt produzieren und freisetzen
P	Die Menge der Pflanzenarten
$p_{i,j}^*$	Eine giftige Pflanze
P_f	Menge aller nicht blockierten Pflanzen
$p_{i,j}$	Die j -te Pflanze der Pflanzenart p_i
P_{min}	Alle Pflanzen mit einem kürzesten Pfad
P_{s_x}	Menge der Pflanzen, die den Giftstoff s_x produzieren können
q	Warteschlange
Q_k	Menge der Auslösebedingungen für einen Signalstoff

$R(C_i, t)$	Die zum Zeitpunkt t verzehrte Energie von C_i
$r(t)$	Signalstoffradius zu einem Zeitpunkt
S	Menge der Substanzen
s_i	Eine Substanz
s_x	Giftstoff
S_{sig}	Untermenge von S , die die Signalstoffe beschreibt
S_{tox}	Untermenge der Substanzen, die die Giftstoffe beschreiben
T	Zeitmengen
t	Zeitpunkt
t_e	Zeitpunkt, an dem die Signalstoffproduktion eingestellt wird
t_f	Zeitpunkt, an dem die Giftstoffproduktion endet
t_g	Intervall, in dem ein Signalstoff über eine symbiotische Verbindung die nächste Pflanze erreicht
T_i	Zeitpunkt, an dem sich eine Pflanze reproduziert
T_k	Nachwirkzeit eines Signalstoffs s_k
t_k	Zeitpunkt, an dem die Produktion des Signalstoffs s_k startet
t_l	Zeitintervall, wann ein Signalstoffradius vergrößert wird
t_x	Zeitpunkt, an dem die Produktion des Giftstoffs ausgelöst wird
t_{end}	Konfigurierbarer Zeitpunkt, an dem die Simulation endet
T_{max}	Untermenge von T , die alle Zeitschritte bis zu einem bestimmten Zeitpunkt umfasst
$V_{i,j}$	Menge aller Pflanzen, die durch eine symbiotische Verbindung miteinander verbunden sind
$W(s_k, p_{i,j}, t)$	Signalstoff-Wirkbereich zu einem Zeitpunkt an einer Pflanze
$w(s_x, t)$	Wirkung des Giftstoffs zu einem Zeitpunkt
$W^*(s_k, t)$	Überlappende Wirkbereiche
Y	Spezifische Teilmenge von C , die sich tatsächlich auf einem Gridfeld befindet
$Y_{((x_k, y_k), t + \Delta t_i)}$	Tatsächliche Menge der Cluster auf dem Feld zum Zeitpunkt $t + \Delta t_i$

$Y_{((x_{k+1}, y_{k+1}), t + \Delta t_i + 1)}$	Tatsächliche Menge der Cluster auf dem Feld zum Zeitpunkt $t + \Delta t_i + 1$
Z	Funktion, die alle Abbruchbedingungen überprüft und die Simulation dann beendet
$Z_1(t_{end})$	Abbruchbedingung, wenn t_{end} erreicht wurde
$Z_2(p_i)$	Abbruchbedingung, wenn es keine Pflanzen der Art p_i mehr gibt
Z_3	Abbruchbedingung, wenn es keine Pflanzen mehr gibt
$Z_4(e_h)$	Abbruchbedingung, wenn es keine Fressfeinde der Art e_h mehr gibt
Z_5	Abbruchbedingung, wenn keine Fressfeindcluster auf dem Grid existieren
$Z_6(\mathcal{E}_{limit})$	Abbruchbedingung, wenn eine maximale Menge an pflanzlicher Energie existiert
$Z_7(n_{limit})$	Abbruchbedingung, wenn eine maximale Anzahl an Fressfeinden auf dem Grid existiert

1 Einführung

Diese Arbeit ist der Konzipierung und Entwicklung eines diskreten Event-Simulators gewidmet, welcher Dynamiken zwischen Pflanzen und Fressfeinden visualisiert.

Dazu werden eine Vielzahl von Faktoren und Eigenschaften hinzugezogen, wie die für jedes Individuum verfügbaren Ressourcen (in Form von Energie) und diversen Signalisations- und Abwehrmechanismen.

Um eine systematische Einführung in dieses Thema zu geben, gliedert sich die Einführung in drei Kapitel. Als erstes wird auf den Hintergrund eingegangen, warum die Simulation der Schutzmechanismen von Pflanzen auf Fressfeinde sinnvoll sein kann. Da das Thema der Arbeit sehr umfangreich ist, wird im zweiten Abschnitt kurz das eigentliche Ziel der Arbeit erläutert. Im dritten und letzten Abschnitt wird der Aufbau und die Struktur dieser Arbeit erläutert.

1.1 Hintergrund

Viele Mechanismen und Strukturen in der Natur haben sich über Millionen von Jahren so entwickelt, dass sie optimal für ihren jeweiligen Zweck genutzt werden können. Dieser fortlaufende Prozess wird umgangssprachlich als Evolution bezeichnet. Da Umweltbedingungen einem stetigen Wandel unterliegen und keinen festen Endpunkt besitzen, kann die Evolution als ein kontinuierlicher Prozess betrachtet werden [16].

In zahlreichen Bereichen lässt sich natürliche Evolution beobachten, bspw. in der Ökologie, wo Organismen versuchen, sich optimal an ihre Umwelt anzupassen, um ihre ökologische Nische bestmöglich zu nutzen [35]. Die ökologische Nische ist ein Konzept, das in der Ökologie die Gesamtheit aller abiotischen und biotischen Umweltfaktoren beschreibt, die ein Organismus zum Überleben benötigt [30].

Anpassungen erfolgen jedoch nicht ausschließlich durch natürliche Evolution, sondern auch durch gezielte Manipulation seitens der Menschen. So werden Organismen bspw. genetisch modifiziert [28], um bei Nutzpflanzen eine höhere Ertragsleistung zu erzielen [31].

In [14] werden die Parallelen zwischen drahtlosen Sensornetzen und (Mosaik-)Bio-

topen behandelt. Dabei werden vor allem Abwehr- und Kommunikationskonzepte von Pflanzen gegenüber Herbivoren behandelt, die in Anbetracht der Ressourcenbeschränkungen als Vorbild für zukünftige Sicherheitskonzepte in drahtlosen Sensornetzen (WSN) dienen können.

Weiterhin werden die Vorteile von evolutionär ausgeprägten ressourceneffizienten Abwehrmechanismen herausgearbeitet, welche Pflanzen im Wettkampf mit Herbivoren ausgeprägt haben. Dabei zeigt sich in der Natur, dass vor allem eine hohe Artenvielfalt in einem Biotop und die direkte oder indirekte Kooperation zwischen pflanzlichen Individuen zu einem effizienten Schutz des Biotops vor Herbivoren führt. Daher ist zu beobachten, dass eine geschickte Verteilung einer Vielzahl von Schutzmechanismen und deren Kooperation ein besonders effizientes auf unterschiedliche Bedrohungsszenarien abgestimmtes Schutzsystem hervorbringen kann.

Ein WSN ist ein Netzwerk aus zahlreichen kleinen Sensoren, die miteinander kommunizieren, Daten erfassen und diese zur weiteren Analyse in verschiedenen Anwendungsbereichen weiterleiten [18, 3, 7]. Im Kontext der referenzierten Arbeiten liegt der Fokus auf der Nutzung von WSN zur Umweltüberwachung. Dabei ermöglicht sie die Erfassung von physikalischen Parametern, wie bspw. der Temperatur, ohne dass eine dauerhafte physische Präsenz vor Ort erforderlich ist [3].

Damit ein WSN Daten auch in abgelegenen Gebieten erfassen und übertragen kann, benötigt jeder Sensor eine Energiequelle, die in der Regel durch eine Batterie bereitgestellt wird [18, 3, 15]. Da die verfügbare Energie durch die Batteriekapazität begrenzt ist, müssen die Sensoren effizient mit dieser Ressource wirtschaften. Eine ähnliche Einschränkung besteht in den Kommunikationsnetzwerken von Pflanzen, wodurch Parallelen zwischen diesen Netzwerken erkennbar werden.

Aufgrund der Parallelen zwischen den beiden Netzwerkarten und der Tatsache, dass Pflanzen über Millionen von Jahren verschiedene Strategien zur Abwehr von Fressfeinden entwickelt haben, erscheint eine Simulation der Pflanzenschutzmechanismen sinnvoll. Diese könnte Erkenntnisse zur Optimierung der Sicherheit in WSNs liefern. Der Grund für das Bestreben, diese Netzwerke besser zu verstehen und zu optimieren, liegt in der Relevanz ihrer Auswirkungen auf das Netzwerk selbst sowie auf die darin übertragenen Informationen. Trotz begrenzter Ressourcen und hoher

Lebenserwartungen strebt man danach, die im Netzwerk verfügbaren Informationen im Rahmen der Möglichkeiten bestmöglich zu schützen [3, 15].

Basierend auf [14], worin Sicherheitsmechanismen nach dem Vorbild des Pflanzenschutzmechanismus vorgestellt werden, könnten Maßnahmen für ein WSN bspw. in Form eines dauerhaften Schutzmechanismus, ähnlich den Dornen bei Pflanzen, implementiert werden. Alternativ könnte auch eine induzierte Abwehrstrategie zur Anwendung kommen, die nur im Fall eines Angriffs aktiviert wird. Diese Strategie würde dazu dienen, das WSN zu schützen und gleichzeitig die verfügbare Energie möglichst effizient zu nutzen.

1.2 Zielsetzung

Auf Grundlage des vorhergehenden Hintergrunds soll ein Modell für den Pflanzenschutzmechanismus entwickelt werden, das ausreichend komplex ist, um eine realistische Formalisierung eines Kommunikationsnetzwerks von Pflanzen zu ermöglichen. Dabei werden alle relevanten Komponenten dieses Netzwerks berücksichtigt, um sicherzustellen, dass die späteren Simulationen realistische Ergebnisse liefern.

Um das Modell bzw. die formale Beschreibung des Kommunikationsnetzwerks der Pflanzen zu simulieren, wird ein Tool mit einer graphischen Oberfläche entwickelt. Dieses Tool ermöglicht es, solche Netzwerke zu erstellen, zu simulieren und bietet die Möglichkeit, das Verhalten über die Zeit hinweg anhand von Plots darzustellen.

1.3 Aufbau dieser Arbeit

Um die Arbeit angesichts der verschiedenen, miteinander verknüpften Fachbereiche übersichtlich zu gestalten, beginnt sie nach der Einleitung mit einer biologischen Einordnung. In diesem Abschnitt wird der biologische Hintergrund des Modells und des Tools erläutert. Dabei werden verschiedene Schutzmechanismen von Pflanzen thematisiert, ebenso wie deren Wirkung auf Fressfeinde. Zudem wird beschrieben, welche Voraussetzungen erfüllt sein müssen, um den Schutzmechanismus auszulösen und wie Pflanzen untereinander kommunizieren, um sich ggf. vor Fressfeinden zu warnen.

Im nächsten Kapitel zur formalen Beschreibung wird ein Modell für ein solches Netzwerk vorgeschlagen und präsentiert, das alle relevanten Faktoren berücksichtigt, um ein möglichst realistisches Ergebnis zu erzielen.

Das Modell dient als Grundlage für die Implementierung, die im Kapitel nach der formalen Beschreibung behandelt wird. Im Kapitel zur Implementierung wird nicht auf jede einzelne Zeile des Quelltextes eingegangen, da dies den Rahmen sprengen würde. Stattdessen wird auf die verschiedenen Klassen eingegangen, wobei die wichtigsten Funktionen erläutert werden und veranschaulicht wird, wie die Klassen miteinander verknüpft sind, um das Gesamtsystem zu realisieren. Aufbauend auf der Implementierung wird zudem eine Bedienungsanleitung für die graphische Oberfläche bereitgestellt.

Zur Demonstration der Funktionsweise des Tools und zur Präsentation möglicher Ergebnisse in Form von Plots wird eine Simulationsstudie durchgeführt. Bei dieser Simulation werden die Parameter des Netzwerks so gewählt, dass die Simulation über einen Zeitraum von mindestens 1000 Zeitschritten läuft und nicht aufgrund einer zu starken Pflanzenseite oder einer übermäßigen Fressfeindseite vorzeitig beendet wird. Diese Arbeit schließt mit einer Diskussion der vom Tool generierten Ergebnisse sowie einem Ausblick auf zukünftige Entwicklungen ab.

2 Biologischer Hintergrund

Um später ein formales Modell zu entwickeln, das präzise und fundierte Aussagen ermöglicht, ist es sinnvoll, die natürlichen Gegebenheiten zu betrachten. Dieses Kapitel bietet einen Überblick über die relevanten Umweltbedingungen sowie die Interaktionen zwischen kooperierenden und feindlichen Akteuren in der Natur. Im Anschluss werden verschiedene Strategien von Pflanzen im Detail betrachtet.

2.1 Das Szenario in der Natur

Ein Blick in die Natur verdeutlicht die beeindruckende Vielfalt der pflanzlichen Welt. Zahlreiche Pflanzenarten existieren in unterschiedlich hoher Dichte, abhängig von den jeweiligen Umweltbedingungen, auf unserer Erde.

Auf der anderen Seite stehen die Gegenspieler der Pflanzen, die sogenannten Herbivoren, umgangssprachlich als Pflanzenfresser bezeichnet. Diese Organismen sind auf Pflanzen als primäre Nahrungsquelle angewiesen, um in ihrem natürlichen Lebensraum zu überleben.

Zwischen diesen beiden Gruppen herrscht ein kontinuierlicher Überlebensdruck: Herbivoren fokussieren sich auf die Nahrungssuche und Fortpflanzung, um das Überleben ihrer Art zu sichern. Gleichzeitig stehen auch Pflanzen in einem ständigen Wettbewerb, um ihr eigenes Überleben und die Erhaltung ihrer Art zu gewährleisten.

Es wird deutlich, dass Pflanzen im Gegensatz zu ihren Fressfeinden nicht frei beweglich sind. Sie sind an den Standort gebunden, an dem sie als Samen niedergelassen wurden. In diesem Aspekt scheinen Pflanzen einen klaren Nachteil gegenüber ihren Fressfeinden zu haben und hätten theoretisch nur geringe Chancen, sich wirksam gegen sie zu behaupten.

Die Biologie, insbesondere die Evolution, zeigt jedoch, dass sich die verschiedenen Arten im Laufe von Millionen Jahren kontinuierlich an ihre Umwelt angepasst haben. Dabei wurden Strategien entwickelt, die dem einzelnen Individuum einen Überlebensvorteil bieten. In der Natur herrscht somit nicht nur ein ständiger Überlebensdruck, sondern auch ein sogenanntes „evolutionäres Wettrüsten“, bei dem es darum geht, gegenüber anderen Organismen einen Vorteil zu erlangen.

2.2 Pflanzliche Schutzmechanismen

Pflanzen haben verschiedene Strategien entwickelt, um sich gegen Fressfeinde zu verteidigen, andere Pflanzen zu warnen oder sich so anzupassen, dass sie für ihre Feinde nahezu unauffindbar bleiben. Dieses Kapitel bietet einen Überblick über unterschiedliche biologische Methoden, die diese Verteidigungsmechanismen unterstützen.

2.2.1 Warnfärbungen bei Pflanzen

In [21] bieten die Autoren einen umfassenden Überblick über pflanzliche Warnfärbungen und deren Bedeutung. Diese Mechanismen sind besonders wichtig, da Pflan-

zen vor Fressfeinden nicht durch Flucht entkommen können. Stattdessen nutzen sie Strategien wie Warnfärbungen, um potenzielle Angreifer effektiv abzuschrecken. Um ihre Überlebenschance zu maximieren und das Überleben ihrer Art zu sichern, haben Pflanzen über die Jahre hinweg eine Palette an Überlebensstrategien entwickelt. Eine davon ist die Nutzung von Warnfarben, auch bekannt als Aposematismus. Diese dienen dazu, potenziellen Fressfeinden zu signalisieren, dass sie giftig, ungenießbar oder gefährlich sind.

In der pflanzlichen Welt wird Aposematismus oft in Verbindung mit einer mechanischen Verteidigung verwendet. Pflanzen zeigen auffällige Farben, um Fressfeinden zu signalisieren, dass sie gefährliche oder ungenießbare Strukturen besitzen. Diese strukturellen Abwehrmechanismen können Dornen, Stacheln oder Stachelborsten sein. Dornen entstehen aus Ästen, Stacheln bilden sich aus Blättern und Stachelborsten entwickeln sich aus Rindengewebe. Die Warnfarben der Pflanze färben schließlich diese gefährlichen Strukturen ein, um die potenziellen Angreifer visuell abzuschrecken.

Ein anschauliches Beispiel für solche Pflanzen, die ihre Dornen bunt einfärben, sind Kakteen, deren Dornen sich in auffälligen Farben von der Pflanzenfarbe abheben.

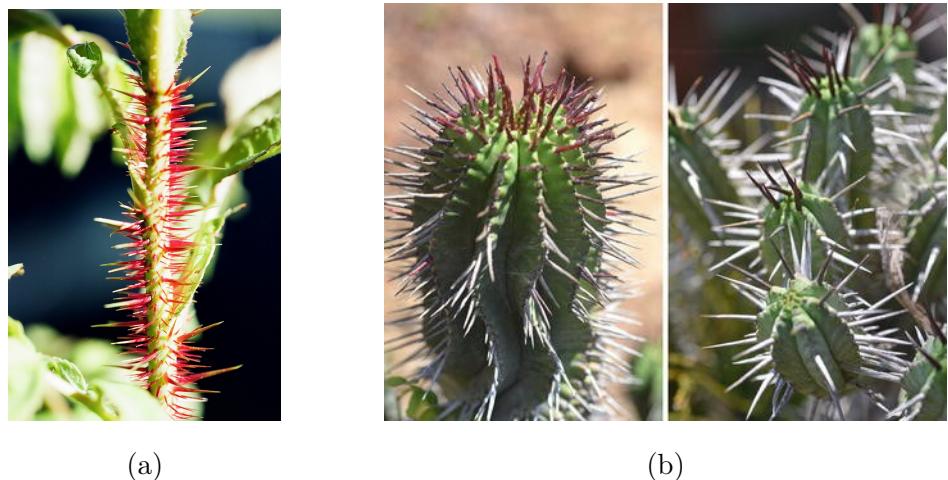


Abbildung 1: Beispiele wie Warnfarben aussehen können

Quellen: (a): [23], (b): [22]

Mechanische Schutzmechanismen wirken häufig nur kurzfristig im Gedächtnis von Fressfeinden und führen in der Regel nicht zu langfristigen Schäden. Aus diesem

Grund kombinieren Pflanzen mechanische Abwehrstrukturen oft mit zusätzlichen Verteidigungsstrategien.

Da nicht jede Pflanze über spitze oder verletzende Strukturen verfügt, entwickeln sie alternative Schutzmechanismen, die unter Umständen auch zur Übertragung von Krankheitserregern beitragen können. Solche alternativen Strukturen umfassen beispielsweise Silica-Nadeln und Raphiden.

Silica-Nadeln erfüllen neben ihrer Funktion als Verteidigungsmechanismus weitere Aufgaben, wie die strukturelle Stabilisierung der Pflanze sowie die Entgiftung von Schwermetallen.

Raphiden sind längliche Zellbestandteile mit nadelartigen Spitzen an beiden Enden. Sie verfügen über spezielle Kanäle und Widerhaken, die dazu dienen, das pflanzen-eigene Gift in das Gewebe von Fressfeinden zu übertragen, wenn diese Pflanzen mit Raphiden konsumieren.

Ähnlich wie Dornen und andere mechanische Abwehrstrukturen verursachen auch Raphiden mechanische Verletzungen, sobald sie mit dem Gewebe in Kontakt kommen. Diese Verletzungen können nicht nur direkte Schäden hervorrufen, sondern auch die Übertragung von Krankheitserregern oder Pathogenen ermöglichen. Dadurch wird der Fressfeind zusätzlich durch Vergiftungen geschädigt.

Mechanische Verteidigungsmechanismen in Kombination mit der Injektion von Bakterien können als zweistufige Abwehrstrategie betrachtet werden. Während Dornen und ähnliche Strukturen unmittelbare Verletzungen verursachen, tritt eine zeitverzögerte Schädigung durch Infektionen ein, die von den übertragenen Bakterien hervorgerufen werden. Die Verbindung dieser beiden Strategien kann metaphorisch als eine Form von „biologischer Kriegsführung“ beschrieben werden.

Da die Ausbildung auffälliger Warnfarben für die Pflanze eine ressourcenintensive Anpassung darstellt, erscheint es sinnvoll, die Entwicklung solcher Warnsignale dynamisch an die jeweiligen Umweltbedingungen anzupassen [21]. Zudem spielt der Entwicklungsstand der Pflanze eine wesentliche Rolle bei der Ausbildung von Schutzmechanismen, da unterschiedliche Stadien spezifische Abwehrstrategien erfordern können.

Junge Pflanzen verfügen in der Regel nicht über ein vollständiges Spektrum an

Schutzmechanismen und konzentrieren ihre verfügbaren Ressourcen daher auf einen primären Abwehrmechanismus [21]. Häufig besitzen sie zunächst nur Strukturen, die mechanische Verletzungen verursachen, ohne jedoch über chemische Verteidigungsstrategien zu verfügen. Aus diesem Grund bilden sie Dornen und ähnliche Strukturen in auffälligen Warnfarben aus, um potenzielle Angreifer abzuschrecken.

Mit zunehmendem Alter entwickeln Pflanzen zusätzliche Verteidigungsmechanismen, wie etwa chemische Abwehrstrategien. In diesem Entwicklungsstadium kann es vorkommen, dass die Warnfarben der Dornen und ähnlichen Strukturen an Intensität verlieren. Dadurch senken die Pflanzen die Kosten für die Aufrechterhaltung der Farbgebung und können ihre Ressourcen effizienter in andere Abwehrmechanismen investieren [21].

Ein Beispiel für Pflanzen, die ihre Ressourcen im Verlauf ihres Lebens gezielt einsetzen, sind Akazien. Diese entwickeln größere Dornen an den unteren Ästen, um einem Befall durch Fressfeinde entgegenzuwirken. Ebenso zeigen bestimmte Bäume, wie Zitrusbäume, in jungen Jahren Dornen und Stacheln, die sie später abwerfen, um die freiwerdenden Ressourcen für andere Zwecke nutzen zu können [21].

2.2.2 Mimikry als Schutzmechanismus

In [21] wurden neben den Warnfarben von Pflanzen auch andere Verteidigungsstrategien, wie verschiedene Formen der Mimikry, behandelt. Manche Pflanzen verfügen nicht über Dornen, Stacheln oder ähnliche Strukturen, die mechanischen Schaden bei Fressfeinden verursachen könnten. Allerdings ist das Vorhandensein solcher Strukturen nicht zwingend erforderlich. In einigen Fällen genügt es, wenn Pflanzen lediglich den Anschein erwecken, solche schädlichen Strukturen zu besitzen. Es sei an dieser Stelle darauf hingewiesen, dass einige pflanzliche Abwehrmechanismen auf Nachahmung beruhen. Dabei imitieren sie spitze und scharfe Strukturen sowie Warnsignale anderer Pflanzen, um Fressfeinde erfolgreich abzuschrecken.

Die Nachahmung anderer Pflanzen lässt sich in zwei Kategorien unterteilen. Zum einen gibt es die Müllersche Mimikry, bei der stacheltragende Pflanzen ihre Warnsignale gegenseitig imitieren und dadurch eine Art Verteidigungsgemeinschaft, auch als Verteidigungsgilde bezeichnet, bilden [21]. Zum anderen existiert die Bates'sche

Mimikry, die von nicht-stacheltragenden Pflanzen genutzt wird. Diese Pflanzen ahmen spitze Strukturen, wie Dornen, sowie deren Warnfarben nach, um potenzielle Fressfeinde erfolgreich abzuschrecken.

Ein weiteres Beispiel für Mimikry findet sich bei Pflanzen wie Agaven oder Palmen, die ihre eigenen Verteidigungsmechanismen nachahmen, um gefährlicher zu erscheinen, als sie tatsächlich sind. Einige Pflanzen sind in der Lage, den Eindruck zu erwecken, mit einer größeren Anzahl von Dornen bedeckt zu sein, indem sie Nachbildungen ihrer Dornen formen. Diese imitierenden Strukturen sind jedoch lediglich Oberflächenabdrücke und keine echten Dornen.

Eine weitere faszinierende Strategie von Pflanzen besteht darin, ihre Farben so anzupassen, dass kleine Fressfeinde wie Insekten, die sich von der Pflanze ernähren und sich auf ihr verstecken, leichter von ihren natürlichen Feinden, beispielsweise Vögeln, entdeckt werden können. Diese Anpassung folgt dem Prinzip „Der Feind meines Feindes ist mein Freund“.

Pflanzen nutzen Mimikry, um potenzielle Fressfeinde abzuschrecken. Sie imitieren beispielsweise Schmetterlingseier durch Eigelege-Mimikry, um andere Schmetterlinge davon abzuhalten, ihre Eier auf der Pflanze abzulegen. Durch das Nachahmen schwarzer Punkte ahmen sie Ameisen nach (Ameisen-Mimikry), um einen Ameisenbefall vorzutäuschen und potenzielle Fressfeinde abzuschrecken. Ebenso kopieren sie das Erscheinungsbild von Blattläusen (Blattlaus-Mimikry), simulieren giftig wirkende Raupen (Raupen-Mimikry) oder imitieren Fraßschäden (Fraßschäden-Mimikry), um die Pflanze für Fressfeinde unattraktiv erscheinen zu lassen.

Diese Strategien schützen Pflanzen sowohl vor Insekten als auch vor größeren Pflanzenfressern, indem sie ihre Angreifer täuschen oder abschrecken. Mimikry ermöglicht es den Pflanzen, potenzielle Fressfeinde davon zu überzeugen, dass sie bereits befallen sind, oder sie erscheinen aufgrund ihrer Warnfarben unverdaulich. In einigen Fällen kombinieren Pflanzen visuelle Mimikry mit zusätzlichen Signalen, wie spezifischen Duftstoffen, um ihre Verteidigung weiter zu verstärken.

2.2.3 Kommunikation zwischen Pflanzen

Pflanzen haben nicht nur gelernt, ihre Farben anzupassen oder durch Nachahmung zu agieren, sondern auch, spezifische Substanzen unter bestimmten Umständen freizusetzen. Eine solche Situation könnte die Beschädigung durch Fressfeinde der Pflanze darstellen. Hierbei greifen die Pflanzen wieder auf das Prinzip „Der Feind meines Feindes ist mein Freund“ zurück, indem sie flüchtige organische Verbindungen (VOCs) aussenden. Diese locken entweder die Feinde ihrer Feinde an [10] oder ermöglichen eine Kommunikation mit anderen Pflanzen derselben Gattung [13]. Das Ziel dieser Methode ist, die Schäden durch Fressfeinde zu minimieren, sodass die Pflanze möglicherweise ihre Überlebenschance erhöht. Darüber hinaus wird diese Strategie auch dazu verwendet, um unbeschädigte Nachbarpflanzen vor einem Angriff zu warnen. Die Nachbarn können sich dann entsprechend vorbereiten und sich durch Warnfarben oder andere Verteidigungsstrategien schützen beziehungsweise besser verstecken.

Zwei Beispiele für die Nutzung flüchtiger organischer Verbindungen sind Linalol, das von Maispflanzen freigesetzt wird, um Raupen abzuschrecken und das Belagern der Pflanze zu verhindern. Darüber hinaus können Tabakpflanzen eine flüchtige Verbindung ausstoßen, die als Duftstoff wirkt und Motten daran hindert, ihre Eier auf den Blättern abzulegen [13].

Diese Methode hat auch ihre Nachteile, da unter Pflanzen ein erheblicher Konkurrenzdruck hinsichtlich essenzieller Überlebensressourcen wie Licht besteht [32]. Wenn eine Pflanze flüchtige Stoffe freisetzt, besteht das Risiko, dass auch konkurrierende Pflanzen diese Signale wahrnehmen und sich, ähnlich wie die ursprünglich adressierten Empfänger, auf bevorstehende Angriffe vorbereiten [20]. Dies mindert die Effektivität dieser Strategie, da nicht nur Verbündete, sondern auch Konkurrenten gewarnt werden.

Ein weiterer Nachteil flüchtiger Signale besteht darin, dass diese über die Luft verbreitet und vom Wind an benachbarte Pflanzen weitergetragen werden. Die produzierende Pflanze hat keinen Einfluss darauf, welche Empfänger die Signale erreichen.

Pflanzen, die nicht in Windrichtung stehen, haben somit keine Möglichkeit, diese Signale wahrzunehmen und sich entsprechend auf einen bevorstehenden Angriff vorzubereiten [20].

Um diesem Problem entgegenzuwirken, haben Pflanzen eine alternative Kommunikationsmethode entwickelt, die auf wasserlöslichen und bodengebundenen Signalen basiert [20]. Diese Signale werden über den Boden an benachbarte Pflanzen übertragen, wodurch diese sich auf einen bevorstehenden Angriff durch Fressfeinde vorbereiten können [20]. Auf diese Weise können auch Pflanzen, die nicht in Windrichtung stehen, von der Signalproduktion profitieren und Schutzmaßnahmen ergreifen. Da die unterirdische Signalübertragung gezielt erfolgt, werden konkurrierenden Pflanzen diese Informationen vorenthalten. Dies verschafft den signalproduzierenden und verbundenen Pflanzen einen Vorteil, da die unvorbereiteten Konkurrenten von Fressfeinden angegriffen werden [20].

2.2.4 Interaktion zwischen Pflanzen

Wie bereits erwähnt, befinden sich Pflanzen auch untereinander in einem Wettbewerb um begrenzte Ressourcen. Dieser Konkurrenzkampf beschränkt sich nicht nur auf Licht, sondern umfasst ebenfalls Wasser und Nährstoffe [32].

Um in diesem Wettbewerb erfolgreich zu sein, haben viele Pflanzen spezielle Strategien entwickelt. Eine davon ist die Freisetzung chemischer Substanzen in die Luft oder in den Boden, die das Wachstum oder die Ansiedlung anderer Pflanzen negativ beeinflussen. Diese Form der chemischen Interaktion, bekannt als Allelopathie, dient dazu, den eigenen Standort zu verteidigen und die Verfügbarkeit von Ressourcen zu sichern [32].

Eine noch engere Wechselbeziehung zwischen Pflanzen zeigt sich im Parasitismus. Dabei entzieht eine Pflanze gezielt einer anderen Pflanze Ressourcen, um ihren eigenen Bedarf zu decken. Halbschmarotzer sind in der Lage, weiterhin Photosynthese zu betreiben und somit Energie für ihr Überleben zu erzeugen, können jedoch ihren Wasser- und Nährstoffbedarf nicht vollständig eigenständig decken [32, S. 641]. Sie entziehen diese lebenswichtigen Ressourcen ihrer Wirtspflanze über das Xylem,

das als Leitgewebe der Pflanze den Wassertransport gewährleistet [25, 32, S. 320]. Vollschorotzer hingegen sind vollständig auf ihre Wirtspflanzen angewiesen. Da sie keine eigene Photosynthese betreiben können, beziehen sie sowohl Wasser als auch organische Nährstoffe direkt von ihrem Wirt [32, S. 641].

2.2.5 Symbiose zwischen Pflanzen und Pilzen

Eine besondere Form des Ressourcenaustauschs ist die Symbiose. Im Gegensatz zum Parasitismus profitieren bei einer symbiotischen Beziehung beide Partner durch ihre Zusammenarbeit [26].

Symbiotische Beziehungen können nicht nur zwischen Pflanzen, sondern auch zwischen Pflanzen und Pilzen sowie zwischen Pflanzen und Bakterien entstehen [32, S. 617]. Eine der bekanntesten und ältesten Symbiosen ist die zwischen Pflanzen und Mykorrhizapilzen. In dieser Beziehung liefert die Pflanze Kohlenhydrate und Aminosäuren, während der Pilz mit seinem weitreichenden Hyphennetzwerk die Wasser- und Nährstoffversorgung der Pflanze verbessert [32].

Die Lebensgemeinschaft mit Mykorrhizapilzen zählt zu den Formen der intrazellulären Symbiosen. In dieser Beziehung dringt der Pilz in die Zellen der Pflanzenwurzel ein, ohne jedoch in das Zytoplasma zu gelangen [32]. Mykorrhiza tritt in mehreren Formen auf, die sich hinsichtlich ihrer Struktur und der beteiligten Organismen voneinander unterscheiden.

1. Vesikulär-arbuskuläre Mykorrhiza (VAM):

Die VAM stellt die am weitesten verbreitete Form der Mykorrhiza dar. In dieser Symbiose dringen die Hyphen des Mykorrhizapilzes in die Zellen der Wurzelrinde ein, jedoch bleiben sie durch Zellwände und Plasmamembranen von den Pflanzenzellen getrennt. Innerhalb dieser Zellen bildet der Pilz ein hochverzweigtes Hyphennetzwerk, das einen effizienten Austausch von Nährstoffen zwischen dem Pilz und der Wirtspflanze ermöglicht. Diese Symbiose ist besonders von Bedeutung in nährstoffarmen Böden, da sie die Aufnahme essenzieller Nährstoffe wie Phosphor erheblich verbessert [32, 5].

2. Endomykorrhiza:

Diese Form der intrazellulären Mykorrhiza findet sich hauptsächlich bei Orchide-

en. In den frühen Entwicklungsstadien sind Orchideenkeimlinge stark auf den symbiotischen Pilzpartner angewiesen, der ihnen essenzielle Kohlenhydrate bereitstellt [32]. Diese symbiotische Beziehung ist von großer Bedeutung für das Überleben und Wachstum der Keimlinge, da sie in dieser Phase noch nicht in der Lage sind, ausreichend eigene Nährstoffe zu synthetisieren [32].

Für die Ausbildung dieser Form der Mykorrhiza dringen die Hyphen des Pilzes in die Wurzeln der Orchidee ein, wodurch die Samenkeimung gefördert wird. Diese Interaktion ist entscheidend, um den Keimlingen eine stabile Nährstoffversorgung zu gewährleisten und somit das Überleben der Orchidee in ihrem natürlichen Lebensraum zu sichern [5].

3. Ektomykorrhiza:

Diese Form der Mykorrhiza ist bei zahlreichen Baumarten weit verbreitet, darunter Eichen, Buchen und Fichten [32]. Der symbiotische Pilz bildet ein dichtes Hyphengeflecht um die Wurzeln, dringt jedoch nicht in die Zellen der Wurzeln ein [5]. Neben der verbesserten Nährstoffaufnahme ermöglicht dieses Myzel eine Vernetzung zwischen verschiedenen Bäumen, wodurch ein komplexes Netzwerk entsteht, das als „Superorganismus“ bezeichnet wird. Dieses Netzwerk dient nicht nur der Nährstoffversorgung, sondern fördert auch den Austausch von Signalen zur Kommunikation zwischen den Bäumen [32].

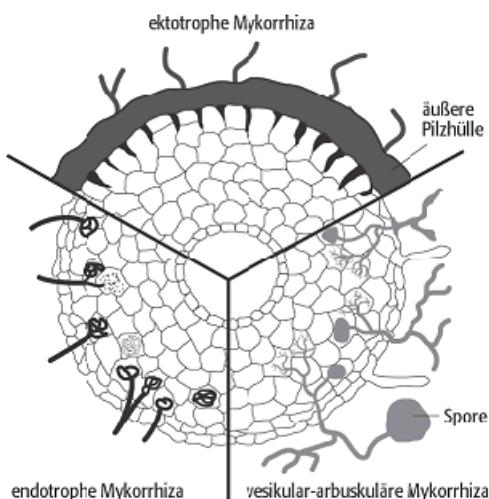


Abbildung 2: Die Formen der Mykorrhiza im Vergleich

Quelle: [17]

Das Mykorrhiza-Netzwerk dient nicht nur dem Austausch von Ressourcen zwischen Pflanze und Pilz, sondern auch als Kommunikationsnetzwerk, das Signale zwischen Pflanzen überträgt [4].

Es wurde nachgewiesen, dass Pflanzen, die von Fressfeinden befallen sind, ähnliche Reaktionen hervorrufen wie benachbarte Pflanzen, die über das Mykorrhiza-Netzwerk miteinander verbunden sind, obwohl sie selbst nicht von den Fressfeinden befallen wurden [4]. Dies deutet darauf hin, dass das Mykorrhiza-Netzwerk auch zur Kommunikation genutzt wird.

Bei den verwendeten Signalen kommen flüchtige Verbindungen wie Methylsalicylat zum Einsatz, das von befallenen Pflanzen über das Netzwerk an benachbarte Pflanzen gesendet wird. Methylsalicylat wirkt dabei abweisend auf Blattläuse und kann deren natürliche Feinde anlocken, um den Befall zu bekämpfen. Diese Strategie ermöglicht es benachbarten Pflanzen, vor einem bevorstehenden Angriff gewarnt zu werden, sodass sie sich entsprechend vorbereiten können [4, 20].

2.2.6 Interaktionen zwischen Pflanzen und Insekten

Um an dieser Stelle noch einmal genauer auf die Taktik „Der Feind meines Feindes ist mein Freund“ einzugehen: Diese Strategie wird angewendet, wenn es zu Insektangriffen kommt. Dabei werden die natürlichen Feinde der pflanzlichen Fressfeinde durch spezifische Lockstoffe angezogen, die bei einem Angriff freigesetzt werden [32, 1].

Die freigesetzten Lockstoffe bestehen aus flüchtigen Verbindungen, die nur von Insekten wahrgenommen werden und nicht von Pflanzen. Diese Verbindungen werden entweder durch mechanische Verletzung der Pflanze oder durch die Eiablage von Insekten induziert und ausgesendet [6, 10]. Solche Abwehrmechanismen gelten als direkte Abwehrstrategien, da sie spezifisch aktiviert werden, sobald ein Angriff stattfindet, um den Energieaufwand der Pflanze zu minimieren [10].

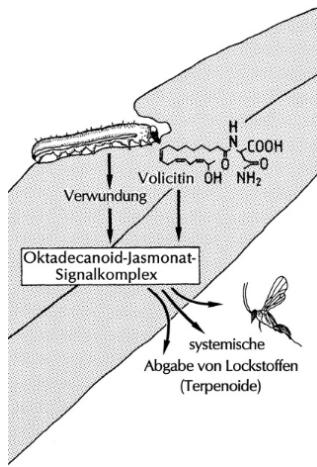


Abbildung 3: Schematischer Ablauf einer Pflanzen-Insekten-Symbiose

Quelle: [32]

Pflanzen erkennen einen Angriff durch spezifische molekulare Muster, bekannt als Herbivore-Associated Molecular Patterns (HAMPs), oder durch direkte mechanische Verletzungen ihrer Zellen, welche den Abwehrmechanismus der Pflanze aktivieren [6].

Insekten, die an einer Pflanze fressen, setzen ebenfalls Substanzen frei oder nutzen Mikroorganismen, um die pflanzliche Abwehrreaktion zu verzögern, um ihre Nahrungsaufnahme so lange wie möglich ungestört fortsetzen zu können [6]. Sobald die Pflanze einen Angriff erkennt, kann sie entsprechende Maßnahmen ergreifen. Dazu gehören unter anderem die Freisetzung von Lockstoffen sowie das Verstärken der Zellwände, um es den Insekten zu erschweren, die Pflanze zu verletzen [10].

Pflanzen sind in der Lage, die Art eines Angriffs zu erkennen und diese Information zu speichern, um ihre Abwehrreaktion zu beschleunigen, wenn derselbe Angriff erneut auftritt [6].

Zudem besteht die Möglichkeit, dass Pflanzen bestimmte Stoffe freisetzen, die nicht nur natürliche Feinde der Insekten anziehen, sondern auch solche, die von den Insekten während der Nahrungsaufnahme aufgenommen werden. Diese Stoffe können Enzyme enthalten, die die Darmstruktur der Insekten beeinflussen und den Verdauungsprozess beeinträchtigen. Sie können auch die Nährstoffqualität des aufgenommenen Pflanzenmaterials reduzieren, was zu einem Mangel an ausreichenden Nährstoffen für das Insekt führt [10].

Zusätzlich zu den Stoffen, die die Insekten direkt schädigen sollen, werden auch chemische Signale freigesetzt, die der Pflanze helfen, den Angriff zu erkennen und daraufhin die Abwehrreaktion gegen die Schädlinge zu initiieren [1].

Ein weiterer wesentlicher Aspekt solcher Mechanismen sind pflanzliche Hormone wie Jasmon-, Salicylsäure und Ethylen, die eine zentrale Rolle bei der Synthese und Freisetzung von Abwehrstoffen spielen [1, 33].

2.2.7 Glucosinolate-basierte Abwehr

Im Laufe der Evolution haben Pflanzen ein umfassendes Arsenal an Schutzmechanismen entwickelt, um sich gegen Fressfeinde zu verteidigen, darunter auch spezielle Stoffwechselprodukte [24].

Ein besonders bedeutendes Stoffwechselprodukt bei Pflanzen ist Glucosinolat, das erst bei Bedarf im Falle eines Angriffs produziert wird [24]. Bei einer Verletzung der Pflanze wird Glucosinolat durch das Enzym Myrosinase in giftige Isothiocyanate umgewandelt, die als Abwehrmechanismus gegen potenzielle Feinde dienen [24, 8]. Glucosinolat fungiert als Speicherprodukt, das bei Bedarf umgewandelt wird. Es wird in speziellen Zellen entlang des Phloems und in den Samen gespeichert [24]. Glucosinolat unterliegt einem strengen Transport- und Feedback-Mechanismus zur Regulierung seiner Konzentration [24], da eine erhöhte Glucosinolat-Konzentration generalistische Insekten abschrecken könnte, jedoch spezialisierte Insekten anziehen, die eine Resistenz gegen diese Stoffwechselprodukte entwickelt haben [8].

Die Produktion von Glucosinolat wird zudem durch die circadiane Uhr beeinflusst, eine biologische Uhr, die die Synthese an die tägliche, variable Präsenz von Insekten anpasst [24]. In anderen Worten: Zu Zeiten mit geringer Insektenpräsenz wird weniger Glucosinolat produziert, während in den Tageszeiten mit erhöhter Insektenaktivität auch mehr Glucosinolat synthetisiert wird.

2.2.8 Schutzmechanismen gegen Pathogene

Eine Pflanze kann nicht nur auf Angriffe durch Herbivoren, Insekten und andere Fraßfeinde reagieren, sondern auch auf Pathogene [33].

In [33] werden diese komplexen Abwehrmechanismen detailliert beschrieben. Die

Abwehrreaktionen gegen Pathogene werden dabei ausgelöst, sobald diese von so genannten Pathogenmustern (Pattern-Triggered Immunity, PTI) oder durch spezifische Effektoren (Effector-Triggered Immunity, ETI) erkannt werden. Die Reaktionen der Pflanze auf Pathogene können entweder lokale oder systemische Abwehrmechanismen aktivieren.

Bei der lokalen Signalisierung senden beschädigte Pflanzenzellen Signale aus, die eine Immunantwort direkt an der Verletzungsstelle aktivieren. Zu diesen Signalen gehören vor allem reaktive Sauerstoffspezies (ROS), die nicht nur die Abwehr vor Ort verstärken, sondern auch benachbarte Zellen alarmieren [33]. Darüber hinaus spielen Pflanzenhormone wie Salicylsäure (SA), Jasmonat (JA) und Ethylen (ET) eine entscheidende Rolle in der Koordination der Abwehrreaktion.

Salicylsäure fungiert als Signalgeber für benachbarte Zellen und bereitet diese auf einen Angriff vor, indem sie spezifische Abwehrgene aktiviert, die direkt gegen ein dringende Pathogene ausgerichtet sind [33]. Jasmonat spielt eine zentrale Rolle bei der Abwehr gegen schädigende Angreifer wie Pilze oder Insekten. Es interagiert häufig mit Ethylen, einem gasförmigen Hormon, das schnell zwischen den Zellen transportiert wird. Ethylen trägt dazu bei, dass die Abwehrmechanismen koordiniert ablaufen, insbesondere bei schnell auftretenden Bedrohungen [29].

Auf der Seite der systemischen Signalisierung spielen die systemisch erworbene Resistenz (SAR) und die induzierte systemische Resistenz (ISR) eine wichtige Rolle [33, 29].

SAR wird aktiviert, wenn Pathogene die Pflanze angreifen und infizieren. SAR schützt die Pflanze, indem es nicht nur lokale Abwehrmechanismen in der betroffenen Region induziert, sondern diese Signale auch über den gesamten Organismus verbreitet. SA ist von zentraler Bedeutung, da sie die Abwehrmechanismen aktiviert und die Pflanze somit auf einen Angriff vorbereitet [33].

Im Gegensatz dazu basiert ISR auf einer anderen Mechanismusstruktur und benötigt kein SA, sondern erfordert vielmehr die Hormone JA und ET, um Abwehrreaktionen in anderen Teilen der Pflanze zu initiieren [29].

3 Formale Beschreibung des Modells

Im vorangegangenen Kapitel wurden die pflanzlichen Schutzmechanismen aus biologischer Perspektive betrachtet. Es zeigte sich, dass dieses Forschungsfeld von hoher Komplexität und großer Tiefe geprägt ist. Dennoch erscheint es möglich, die grundlegenden Aspekte zu abstrahieren und ein Modell zu entwickeln, das einerseits hinreichend einfach ist, um analysiert werden zu können, andererseits jedoch die vielschichtige Realität in der Natur adäquat widerspiegelt. Im Folgenden werden daher die zentralen Komponenten formal definiert, die in Bezug auf Pflanzen, ihre Fressfeinde, die von Pflanzen produzierten Substanzen, die pflanzliche Kommunikation sowie die Umwelt von Bedeutung sind.

3.1 Das Grid als Biotop

Ein Grid bietet eine geeignete Grundlage zur Darstellung einer Umgebung, die vielfältige und dynamische Interaktionen zwischen verschiedenen Elementen ermöglicht. In diesem Kontext können sowohl Pflanzen als auch Fressfeinde auf spezifischen Positionen innerhalb des Grids platziert werden. Mathematisch wird das Grid als eine Menge von Positionen definiert:

$$G = \{(x, y) | x, y \in \mathbb{N}_{\leq 80}\} \quad (1)$$

Um eine hinreichend große Umgebung zu simulieren, wird das Grid so gestaltet, dass es bis zu 6400 Positionen umfasst. Jede dieser Positionen repräsentiert einen potenziellen Platz, der von Pflanzen oder Fressfeinden eingenommen werden kann. Diese Struktur ermöglicht die Modellierung komplexer Wechselwirkungen und die Darstellung vielfältiger Szenarien innerhalb des simulierten Netzwerks.

3.2 Die Akteure auf dem Grid

Auf dem Grid können verschiedene Arten von Akteuren platziert werden, die miteinander in Interaktion treten, sei es in Form von Konkurrenz oder Kooperation. In diesem Kapitel werden diese Akteure näher betrachtet und formalisiert.

3.2.1 Die pflanzlichen Spieler

Auf der einen Seite stehen die Pflanzen, deren primäres Ziel es ist, sich gegen Fressfeinde zu verteidigen und ihr Überleben zu sichern.

Diese Pflanzen können durch eine Menge beschrieben werden, die formal wie folgt definiert ist:

$$P = \{p_i | i \in \mathbb{N}_{\leq 16}\} \quad (2)$$

Diese Menge repräsentiert die verschiedenen Pflanzenarten, die innerhalb des Grids platziert werden können. Da von jeder Pflanzenart mehrere Exemplare auf dem Grid existieren können, ist es sinnvoll, zusätzlich die j -te Instanz einer spezifischen Pflanzenart zu definieren:

$$p_{i,j} \text{ ist die } j\text{-te Instanz der Pflanzenart } p_i \quad (3)$$

3.2.2 Die feindlichen Spieler

Neben den Pflanzen können auch Fressfeinde auf dem Grid positioniert werden. Analog zu den Pflanzen lassen sich die Fressfeinde durch eine Menge beschreiben, die die verschiedenen Arten wie folgt definiert:

$$E = \{e_i | i \in \mathbb{N}_0, 0 \leq i \leq 15\} \quad (4)$$

Diese Menge beschreibt die Arten von Fressfeinden, deren Anzahl auf maximal 15 begrenzt ist.

Fressfeinde können in unterschiedlich großen Gruppen auftreten: Einige, wie etwa Ameisen, agieren in großen Schwärmen, während andere Arten bevorzugt einzeln auftreten. Zur Erfassung dieser unterschiedlichen Verhaltensmuster wird das Konzept des Fressfeindclusters C_i eingeführt. Dieses beschreibt sowohl eine Ansammlung von Fressfeinden als auch deren jeweilige Gruppengröße.

$$C_i = (e_h, n) \text{ mit } n \geq 1 \quad (5)$$

Ein solches Cluster C_i wird durch ein Tupel dargestellt, das die Fressfeindart $e_h \in E$ sowie die Gruppengröße $n \in \mathbb{N}$ umfasst. Dabei muss ein Cluster mindestens die Größe 1 aufweisen, um auf dem Grid platziert werden zu können.

Zur Veranschaulichung sei ein Beispiel mit 6 Clustern und 3 verschiedenen Fressfeindarten gegeben:

$$\begin{aligned} C_1 &= (e_1, 50) & C_2 &= (e_2, 37) & C_3 &= (e_1, 62) \\ C_4 &= (e_2, 8) & C_5 &= (e_1, 50) & C_6 &= (e_3, 1) \end{aligned}$$

3.3 Struktur und Belegung der Gridfelder

Da sich die zuvor definierten Akteure auf demselben Grid befinden und deren Handlungen das Grid im Verlauf der Zeit dynamisch verändern, wird in diesem Kapitel das Grid näher betrachtet und formalisiert. Dies umfasst sowohl die möglichen Belegungen eines Gridfelds als auch das Hinzufügen und Entfernen von Elementen.

3.3.1 Das dynamische Grid

Das Grid G sei wie zuvor definiert. Jede Position (x, y) wird durch ein Tupel beschrieben, das aus zwei Mengen besteht, um die gleichzeitige Speicherung mehrerer Elemente an einer Position zu ermöglichen. Die erste Menge repräsentiert die potenziell auf dem Feld platzierte Pflanze, während die zweite Menge die Fressfeinde enthält, die sich zu einem gegebenen Zeitpunkt an dieser Position befinden.

Da sowohl der Gesamtzustand des Grids als auch die Zustände einzelner Felder zeitlich variieren können, wird eine Menge von Zeitpunkten $T = \{t | t \in \mathbb{N}_0, t \leq m\}$ eingeführt. Diese erlaubt die Beschreibung des Zustands zu einem beliebigen Zeitpunkt $t \in T$. Daraus ergibt sich die Definition der Funktion l :

$$l : G \times T \rightarrow (\{p_{i,j} | i \in \mathbb{N}_{16}, j \in \mathbb{N}\} \cup \emptyset) \times (Y \in \mathcal{P}(C)) \quad (6)$$

Die Funktion beschreibt in Worten, dass auf einem Feld entweder eine Pflanze vorhanden ist oder keine Pflanze platziert wurde. Falls keine Pflanze vorhanden ist, wird die leere Menge \emptyset verwendet. Die zweite Menge beschreibt die Fressfeindcluster. Dabei steht C für die Menge aller möglichen Fressfeindcluster, während $\mathcal{P}(C)$ die Potenzmenge von C darstellt, also die Menge aller Teilmengen von C . Y bezeichnet eine spezifische Teilmenge von C , die die tatsächlich auf einem Feld befindlichen Fressfeindcluster repräsentiert.

Zur Veranschaulichung werden im Folgenden Beispiele für verschiedene mögliche Belegungen eines Feldes gegeben:

- Leeres Feld: $l((x, y), t) = (\emptyset, \emptyset)$
- Ein Feld mit Pflanze: $l((x, y), t) = (p_{i,j}, \emptyset)$
- Ein Feld mit einem oder mehreren Fressfeinden: $l((x, y), t) = (\emptyset, Y)$
- Ein Feld mit Pflanze und Fressfeinden: $l((x, y), t) = (p_{i,j}, Y)$

3.3.2 Das Hinzufügen von Elementen

Aufgrund der dynamischen Veränderungen innerhalb des Grids können verschiedene Situationen auftreten, in denen eine Pflanze oder ein Fressfeindcluster zu einem Feld hinzugefügt wird. Dabei lassen sich fünf unterschiedliche Fälle identifizieren, die eine Veränderung der aktuellen Belegung einer Position beschreiben.

1. Hinzufügen einer Pflanze auf ein leeres Feld.
 2. Hinzufügen eines Clusters auf ein leeres Feld.
 3. Hinzufügen eines Clusters auf ein Feld mit einer Pflanze.
 4. Hinzufügen eines Clusters auf ein Feld ohne Pflanze und mit anderen Clustern.
 5. Hinzufügen eines Clusters auf ein Feld mit Pflanze und anderen Clustern.
-

Im Folgenden sei C_k nicht zwingend das letzte Element der Menge aller auf dem Feld platzierten Fressfeindcluster, sondern vielmehr ein beliebiges Cluster mit $k \in \mathbb{N}_{\leq n}$, wobei $n = |\{Y \in \mathcal{P}(C)\}|$ die Kardinalität der Teilmenge Y aus der Potenzmenge von C ist.

$$l((x, y), t) = \begin{cases} (\emptyset, \emptyset) \cup (p_{i,j}, \emptyset) = (p_{i,j}, \emptyset) & : 1. \text{ Fall} \\ (\emptyset, \emptyset) \cup (\emptyset, \{C_k\}) = (\emptyset, \{C_k\}) & : 2. \text{ Fall} \\ (p_{i,j}, \emptyset) \cup (\emptyset, \{C_k\}) = (p_{i,j}, \{C_k\}) & : 3. \text{ Fall} \\ (\emptyset, \{C_1, \dots, C_{n-1}\}) \cup (\emptyset, \{C_k\}) \\ \quad = (\emptyset, \{C_1, \dots, C_{n-1}\} \cup \{C_k\}) & : 4. \text{ Fall} \\ \quad \quad \quad C_k \notin \{C_1, \dots, C_{n-1}\} \\ (p_{i,j}, \{C_1, \dots, C_{n-1}\}) \cup (\emptyset, \{C_k\}) \\ \quad = (p_{i,j}, \{C_1, \dots, C_{n-1}\} \cup \{C_k\}) & : 5. \text{ Fall} \\ \quad \quad \quad C_k \notin \{C_1, \dots, C_{n-1}\} \end{cases} \quad (7)$$

3.3.3 Das Entfernen von Elementen

Ähnlich wie Elemente zu einem Feld im Grid hinzugefügt werden, können auch Situationen entstehen, in denen Elemente von einem Feld entfernt werden.

1. Entfernen einer Pflanze von einem Feld ohne Cluster.
2. Entfernen des einzigen Clusters von einem Feld ohne Pflanze.
3. Entfernen des einzigen Clusters von einem Feld mit Pflanze.
4. Entfernen eines Clusters von einem Feld ohne Pflanze und anderen Clustern.
5. Entfernen eines Clusters von einem Feld mit Pflanze und anderen Clustern.

Diese Fälle lassen sich formal auf die gleiche Weise wie das Hinzufügen von Elementen zu einem Feld beschreiben. Dabei sei k wie bereits beim Hinzufügen von Elementen zu einem Feld definiert.

$$l((x, y), t) = \begin{cases} (p_{i,j}, \emptyset) \setminus (p_{i,j}, \emptyset) = (\emptyset, \emptyset) & : 1. \text{ Fall} \\ (\emptyset, \{C_k\}) \setminus (\emptyset, \{C_k\}) = (\emptyset, \emptyset) & : 2. \text{ Fall} \\ (p_{i,j}, \emptyset) \setminus (\emptyset, \{C_k\}) = (p_{i,j}, \emptyset) & : 3. \text{ Fall} \\ (\emptyset, \{C_1, \dots, C_n\}) \setminus (\emptyset, \{C_k\}) \\ = (\emptyset, \{C_1, \dots, C_n\} \setminus C_k) & : 4. \text{ Fall} \\ (p_{i,j}, \{C_1, \dots, C_n\}) \setminus (\emptyset, \{C_k\}) \\ = (p_{i,j}, \{C_1, \dots, C_n\} \setminus C_k) & : 5. \text{ Fall} \end{cases} \quad (8)$$

3.4 Die Bewegung von Fressfeinden über das Grid

Ein Fressfeindcluster soll sich nicht zufällig und ohne Struktur über das Grid bewegen, sondern eine annähernd natürliche Bewegung aufweisen, wie sie in der Natur beobachtet werden kann. Daher beschäftigt sich das folgende Kapitel mit der Formalisierung des Bewegungsmusters, einschließlich der Pfadfindung zur nächstgelegenen Pflanze sowie der darauf basierenden Fortbewegung, um diese zu erreichen.

3.4.1 Die Suche nach der nächsten Pflanze

Wird ein Fressfeindcluster auf ein Feld ohne Pflanzen positioniert, so sollte sich dieses Cluster in Richtung der nächstgelegenen Pflanze bewegen. Es gilt folgende Bedingung:

$$\exists x, \exists y, \exists t : l((x, y), t) = (\emptyset, D) \wedge C \in D \quad (9)$$

Dabei dient D als eine Art Platzhalter für die Fressfeindcluster, die sich zum Zeitpunkt t tatsächlich auf dem entsprechenden Gridfeld befinden.

Es ist wichtig, dass die potenzielle Produktion von Gift, die ein Fressfeindcluster vertreiben könnte, an dieser Stelle noch nicht berücksichtigt wird. Dieser Aspekt wird später analysiert, da unter diesen Bedingungen auch eine Pflanze auf demselben Feld existieren kann und der Fressfeind dennoch das Feld verlässt.

In diesem Kontext kann das Grid als ein Graph betrachtet werden, in dem jedes Feld einen Knoten darstellt und die Distanz benachbarter Felder den Wert 1 besitzen.

Aufgrund dieser Struktur eignet sich die Breitensuche besonders gut [11], um die nächstgelegene Pflanze zu identifizieren.

Die folgenden Komponenten werden für die Durchführung der Breitensuche verwendet:

- Ein Grid G
- Ein Startpunkt $\sigma = (x_s, y_s) \in G$ und ein Zielpunkt $\rho = (x_e, y_e) \in G$
- Vier mögliche Bewegungsrichtungen $\Gamma = \{(1, 0); (-1, 0); (0, 1); (0, -1)\}$, die eine Bewegung zu Nachbarknoten in den Richtungen rechts, links, oben und unten erlauben.

Ziel des Algorithmus¹ ist es, den kürzesten Pfad zu einer Pflanze zu finden. Zu Beginn wird eine Warteschlange q initialisiert, die zunächst nur den Startpunkt enthält. Eine Distanzabbildung Ψ dient dazu, den Abstand jedes Punktes im Grid zum Startpunkt zu speichern, wobei $\Psi(\sigma)$ zu Beginn auf 0 gesetzt wird. Zusätzlich wird eine Vorgängerabbildung Π eingeführt, die für jeden Punkt den vorherigen Punkt auf dem Pfad speichert; der Startpunkt wird dabei auf \emptyset gesetzt.

Um die Suche effizient zu gestalten, werden die möglichen Bewegungsrichtungen Γ dynamisch priorisiert. Dabei wird für jede potenzielle Bewegung die neue Position berechnet und die Bewegung basierend auf der euklidischen Distanz zum Zielpunkt ρ sortiert.

Die dynamische Priorisierung funktioniert folgendermaßen: Jede mögliche Bewegungsrichtung wird aus der Menge von Richtungen basierend auf der quadratischen euklidischen Distanz zum Zielpunkt sortiert. Die Priorisierungsfunktion berechnet für jede Richtung $\gamma \in \Gamma$ die Distanz zum Zielpunkt. Diese Distanz wird als quadratische Summe der Differenzen der x- und y-Koordinaten zwischen der aktuellen Position und der Zielposition definiert:

$$\text{dist}(\sigma, \rho, \gamma) = (x_p + \gamma_0 - x_e)^2 + (y_p + \gamma_1 - y_e)^2 \quad (10)$$

¹Pseudocode in Algorithmus 1, siehe Anhang

Dabei stehen (x_p, y_p) für die aktuelle Position und (x_e, y_e) für die Zielposition. Diese Berechnung gewährleistet, dass die Bewegungsrichtungen nach ihrer Nähe zum Zielpunkt sortiert werden, wodurch die Bewegung gezielt in Richtung des Ziels gelenkt wird.

Ohne eine solche Sortierung würden die Fressfeindcluster ein unnatürliches Verhalten zeigen: Die Bewegung würde zunächst vollständig in eine Richtung erfolgen (z.B. entlang der x-Achse), bevor die Bewegung in die andere Richtung (z.B. entlang der y-Achse) aufgenommen wird. Dieses blockartige Vorgehen würde dazu führen, dass die Suche in eine einzige Richtung voranschreitet, während andere Bewegungsrichtungen ignoriert werden.

Die dynamische Priorisierung hingegen gewährleistet, dass die Suche stets auf das Ziel ausgerichtet bleibt, indem sie der kürzesten euklidischen Distanz folgt. Dadurch entsteht ein natürliches und zugleich effizientes Bewegungsmuster.

Der Hauptteil des Algorithmus besteht aus einer Schleife, die so lange ausgeführt wird, bis die Warteschlange leer ist. In jeder Iteration wird der vorderste Punkt aus der Warteschlange entfernt und untersucht. Der Punkt wird durch die Distanzabbildung Ψ repräsentiert, die den minimalen Abstand des jeweiligen Punkts vom Startpunkt speichert. Wenn dieser Punkt mit dem Zielpunkt ρ identisch ist, wird die Schleife beendet, da der kürzeste Pfad gefunden wurde. Andernfalls werden die benachbarten Punkte überprüft.

Die Bewegungsrichtungen werden gemäß der zuvor berechneten Priorisierung geordnet, und für jede Richtung wird die neue Position μ bestimmt. Befindet sich diese Position innerhalb der Grenzen des Grids und wurde sie noch nicht besucht, so wird die Distanz von μ zum Startpunkt in der Distanzabbildung $\Psi(\mu)$ gespeichert. Gleichzeitig wird der Vorgänger $\Pi(\mu)$ auf den aktuellen Punkt gesetzt, und μ wird der Warteschlange hinzugefügt.

Durch diesen systematischen Prozess werden alle erreichbaren Punkte durchsucht, bis entweder das Ziel erreicht ist oder keine weiteren Punkte mehr untersucht werden können. Der Algorithmus liefert entweder den kürzesten Pfad durch Rückverfolgung der Vorgängerabbildung oder gibt eine leere Lösung zurück, falls kein Pfad existiert.

3.4.2 Bewegung entlang des Pfades

Existiert mindestens eine Pflanze auf dem Grid, wird diese mithilfe des beschriebenen Algorithmus gefunden, der den kürzesten Pfad $\Omega = [\sigma, \dots, \rho]$ liefert. Das Fressfeindcluster C_j soll sich entlang dieses Pfades vom Startpunkt σ bis zum Endpunkt ρ bewegen. Dazu wird der gefundene Pfad iterativ durchlaufen, wobei für jede Position $(x_k, y_k) \in \Omega$ überprüft wird, ob $(x_k, y_k) \neq \rho$ ist.

Ein Cluster muss sich nicht zwingend in jedem Zeitschritt um ein Feld weiterbewegen; es kann auch aus Fressfeinden bestehen, die mehrere Zeitschritte benötigen, um das nächste Feld zu erreichen.

An dieser Stelle wird eine Funktion \mathcal{M} eingeführt, die die Bewegung eines Fressfeindclusters C_j beschreibt. Das Cluster C_j bewegt sich in Richtung der Zielposition, indem es nach jedem Zeitintervall $\Delta t_i \geq u$ mit $u \in \mathbb{N}_{\geq 1}$ Zeitschritten zum nächsten Feld im kürzesten Pfad weiterzieht. Das Cluster verbleibt jeweils Δt_i Zeitschritte auf einer Position, bevor es zum nächsten Feld im Pfad übergeht. Der Prozess wird gestoppt, wenn das Cluster den Zielpunkt ρ mit der angestrebten Pflanze erreicht. Zusammenfassend wird das Cluster C_j zum Zeitpunkt $t + \Delta t_i$ aus der Menge $Y_{((x_k, y_k), t + \Delta t_i)}$ entfernt und zur Menge $Y_{((x_{k+1}, y_{k+1}), t + \Delta t_i + 1)}$ zum Zeitpunkt $t + \Delta t_i + 1$ hinzugefügt. Daraus ergeben sich folgende drei Fälle, die in der Funktion \mathcal{M} berücksichtigt werden:

-
1. Entfernen eines Clusters C_j von Feld (x_k, y_k) zum Zeitpunkt $t + \Delta t_i$.
 2. Hinzufügen eines Clusters C_j auf ein Feld (x_{k+1}, y_{k+1}) ohne Pflanze zum Zeitpunkt $t + \Delta t_i + 1$.
 3. Hinzufügen eines Clusters C_j auf ein Feld (x_{k+1}, y_{k+1}) mit Pflanze zum Zeitpunkt $t + \Delta t_i + 1$.
-

Die formal definierte Funktion \mathcal{M} erhält als Eingabe zwei benachbarte Felder $l_1 = l((x_k, y_k), t)$ und $l_2 = l((x_{k+1}, y_{k+1}), t)$ sowie ein konfigurierbares Δt_i , das die Anzahl der Zeitschritte angibt, die ein Cluster auf einem Feld verweilt. Die Funktion ist dann definiert als:

$$\mathcal{M}(l_1, l_2, t + \Delta t_i) = \begin{cases} l((x_k, y_k), t + \Delta t_i) = (\emptyset, Y_1 \setminus \{C_m\}) & : 1. \text{ Fall} \\ l((x_{k+1}, y_{k+1}), t + \Delta t_i + 1) = (\emptyset, Y_2 \cup \{C_m\}) & : 2. \text{ Fall} \\ l((x_{k+1}, y_{k+1}), t + \Delta t_i + 1) = (p_{i,j}, Y_2 \cup \{C_m\}) & : 3. \text{ Fall} \end{cases} \quad (11)$$

wobei $Y_1 = Y_{((x_k, y_k), t + \Delta t_i)}$ und $Y_2 = Y_{((x_{k+1}, y_{k+1}), t + \Delta t_i + 1)}$ dargestellt werden.

Es kann jedoch auch mehrere Pflanzen geben, die alle den kürzesten Pfad zum Fressfeindcluster haben, sodass dieses ein Ziel auswählen kann. Daher werden nun alle Pflanzen, die diesen kürzesten Pfad teilen, in der Menge P_{min} gesammelt. Die Menge P_{min} ist wie folgt definiert:

$$P_{min} = \{p_{i,j} \mid i \in \mathbb{N}_{\leq 16}, d(\sigma, \rho) = d_{min}\} \quad (12)$$

In Worten stellt P_{min} die Menge der Pflanzen dar, die alle die gleiche minimale Distanz d_{min} zum Fressfeindcluster aufweisen.

Anschließend wird mithilfe einer Zufallsfunktion $\mathcal{R}(P_{min})$ eine zufällige Pflanze aus der Menge ausgewählt, die dann als Ziel für das Fressfeindcluster dient.

3.5 Energiehaushalt der Pflanze

Eine Pflanze verfügt stets über einen bestimmten Energiehaushalt, der ihr zum Überleben zur Verfügung steht und dabei sowohl steigen als auch sinken kann. Dieser Aspekt wird im folgenden Kapitel betrachtet und für das Modell formalisiert.

3.5.1 Initiale Energieeinheit einer Pflanze

Jede Pflanze $p_{i,j}$ auf dem Grid besitzt zum Zeitpunkt $t = 0$ eine anfängliche Energieeinheit $\mathcal{E}_{i,j}(0)$:

$$\mathcal{E}_{i,j}(0) \in \mathbb{R}^+ \quad \text{für jede Pflanze } p_{i,j} \quad (13)$$

Darüber hinaus besitzen einzelne Pflanzen $p_{i,j}$ der Pflanzenart p_i unterschiedliche anfängliche Energiemengen. Dies dient dazu, eine gewisse Dynamik im Grid zu erzeugen und sowohl energiereichere als auch energiearme Pflanzen zu repräsentieren.

3.5.2 Energiezuwachs einer Pflanze

Überlebt eine Pflanze $p_{i,j}$, so erhöht sich ihre Energie pro Zeitschritt $t \geq 1$ um den Faktor $\frac{r_{i,j}}{100}$ der anfänglichen Energiemenge $\mathcal{E}_{i,j}(0)$. Dabei bezeichnet $r_{i,j}$ die Anzahl der Energieeinheiten, um die die Energie steigt. Die Energie der Pflanze $\mathcal{E}_{i,j}(t+1)$ ergibt sich somit wie folgt:

$$\mathcal{E}_{i,j}(t+1) = \mathcal{E}_{i,j}(0) \cdot \left(1 + \frac{r_{i,j}}{100} \cdot t\right) \quad (14)$$

Die Formel gibt die Energieeinheiten an, die einer Pflanze zum Zeitpunkt $t+1$ zur Verfügung steht, um auf dem Grid weiter überleben zu können. Es gibt weitere Einflussfaktoren, die den Energilevel jeder Pflanze im Zeitverlauf mit bestimmen.

3.5.3 Mindestenergie zum Überleben einer Pflanze

Um überleben zu können, benötigt eine Pflanze $p_{i,j}$ eine Mindestanzahl an Energieeinheiten. Hierzu wird eine konfigurierbare Schranke $B_{i,j}$ eingeführt, die angibt, welche Energiemenge die Pflanze mindestens benötigt, um zu überleben.

$$B_{i,j} \in \mathbb{R}^+ \quad \text{für die Pflanze } p_{i,j} \quad (15)$$

Dieser Schwellwert darf nicht unterschritten werden. Die verfügbaren Energieeinheiten müssen mindestens $B_{i,j}$ betragen, damit die betrachtete Pflanze überleben kann:

$$\mathcal{E}_{i,j}(t) \geq B_{i,j} \quad \text{benötigt die Pflanze } p_{i,j} \text{ zum Überleben} \quad (16)$$

Wird der Schwellwert $B_{i,j}$ unterschritten, verfügt die Pflanze nicht mehr über ausreichend Energie, um zu überleben, und stirbt.

$$\mathcal{E}_{i,j}(t) < B_{i,j} \quad \text{Pflanze } p_{i,j} \text{ stirbt} \quad (17)$$

Infolge dieses Ereignisses verschwindet die betroffene Pflanze vom Grid. Dadurch wird einer der definierten Fälle zum Entfernen von Elementen von einem Feld angewendet.

3.5.4 Energieprozentsatz als Lebensanzeige für die Pflanzen

Während der Simulation wird auf jedem Gridfeld, das von einer Pflanze $p_{i,j}$ belegt ist, eine Zahl angezeigt. Diese Zahl repräsentiert den prozentualen Anteil der aktuell verbleibenden Energie der Pflanze im Vergleich zu ihrer anfänglichen Energiemenge $\mathcal{E}_{i,j}(0)$. Der angezeigte Wert kann dabei sowohl über als auch unter 100% liegen.

$$\text{Energieprozentsatz} = \frac{\mathcal{E}_{i,j}(t)}{\mathcal{E}_{i,j}(0)} \cdot 100 \quad (18)$$

3.6 Die Nachkommen der Akteure

In diesem Kapitel wird die Erzeugung von neuen Pflanzen sowie die Generierung von Individuen innerhalb eines Fressfeindclusters beschrieben. Dabei ist zu beachten, dass die Erzeugung neuer Pflanzen auf einem festgelegten Intervall basiert, welches überprüft, ob eine ausreichende Anzahl an Simulationsschritten vergangen ist.

Im Gegensatz dazu hängt die Erzeugung neuer Individuen innerhalb eines Fressfeindclusters von einem sogenannten Fresserfolg ab. Mit anderen Worten: Die Generierung eines neuen Individuums ist direkt von der Menge der von einem Cluster konsumierten pflanzlichen Energie abhängig.

3.6.1 Die pflanzlichen Nachkommen

Für jede Pflanze $p_{i,j}$ ist ein Vermehrungszeitpunkt $T_i \in \mathbb{N}$ festgelegt. Dieser gibt an, nach wie vielen Simulationsschritten die Pflanze Nachkommen erzeugt. Nach T_i Schritten setzt sie neue Nachkommen frei.

Die Anzahl der Nachkommen, die eine Pflanze hervorbringt, wird für jede Pflanze $p_{i,j}$ durch $N_{i,j} \in \mathbb{N}_0$ definiert. Ist $N_{i,j} = 0$, werden keine Samen ausgesendet, und es entstehen keine Nachkommen.

Die Nachkommen einer Pflanze gehören stets derselben Art wie die Mutterpflanze $p_{i,j}$ an und werden der Pflanzenart p_i zugeordnet.

Ähnlich wie manuell platzierte Pflanzen auf dem Grid, erhalten auch die von Pflanzen produzierten Nachkommen eine vorgegebene anfängliche Energiemenge $\mathcal{E}_{i,j}(0) \in \mathbb{R}^+$. Das bedeutet, dass jedem Nachkommen bei seiner Entstehung die Energie $\mathcal{E}_{i,j}(0)$ zugewiesen wird.

Um sicherzustellen, dass nicht jede Pflanze ihre Nachkommen überall auf dem Grid platzieren kann, wird für jede Pflanze ein spezifischer Minimum- und Maximum-Radius festgelegt, innerhalb dessen Nachkommen um die Mutterpflanze positioniert werden können. Zur Bestimmung der Gridfelder $(x^*, y^*) \in G$, die sich in einem bestimmten Abstand von der Mutterpflanze auf dem Feld (x, y) befinden, wird die euklidische Distanz d_E herangezogen. Es gilt: $d_{min} \leq d_E \leq d_{max}$.

$$d_E = \sqrt{(x^* - x)^2 + (y^* - y)^2} \quad (19)$$

Die Menge der möglichen Felder, auf denen Nachkommen platziert werden können, wird formal wie folgt definiert:

$$L = \{(x^*, y^*) \in G \mid \exists x, \exists y, \exists t \text{ für } l((x, y), t) \text{ mit } d_{min} \leq d_E \leq d_{max}\} \quad (20)$$

Die Menge L umfasst alle Gridfelder, die sich innerhalb des festgelegten Radius befinden. Da jedoch nicht garantiert werden kann, dass diese Felder frei von Pflanzen sind, ist eine zusätzliche Filterung erforderlich, um jene Felder zu identifizieren, die keine Pflanze enthalten. Die gefilterte Menge L^* wird wie folgt definiert:

$$L^* = \{(x^*, y^*) \in L \mid l((x^*, y^*), t) = (\emptyset, Y)\} \quad (21)$$

wobei Y die tatsächlich auf dem Feld befindlichen Fressfeindcluster bezeichnet.

Die Menge L^* stellt eine Untermenge von L dar und umfasst alle Felder innerhalb des Minimum-Maximum-Radius, die nicht von einer Pflanze belegt sind.

Da eine Pflanze keinen Einfluss darauf hat, wo ihre Nachkommen platziert werden, wird für jeden Nachkommen zufällig eine Position aus der Menge L^* ausgewählt. Ist jedoch die Menge L^* leer, weil alle Positionen innerhalb des festgelegten Radius belegt sind, kann keine geeignete Position für den Nachkommen gefunden werden. In diesem Fall wird der Nachkomme entweder nicht erzeugt oder ist aufgrund des Konkurrenzdrucks nicht in der Lage, sich durchzusetzen.

3.6.2 Die Fresserfolge und Nachkommen der Fressfeinde

Bei Fressfeindclustern wird die Reproduktion von Individuen nicht durch ein festes Intervall, sondern durch den sogenannten Fresserfolg bestimmt. Mit anderen Worten:

Die Größe des Clusters wächst in Abhängigkeit von der Menge der verzehrten Energieeinheiten der Pflanze. Dabei ist es entscheidend, dass die verschiedenen Cluster nicht alle mit der gleichen Geschwindigkeit wachsen. Stattdessen soll eine Individualität in den Wachstumsraten berücksichtigt werden, um die natürliche Variabilität, wie sie in der realen Umwelt beobachtet wird, nachzubilden.

Die Erzeugung eines Nachkommen soll flexibel konfigurierbar gestaltet werden, sodass festgelegt werden kann, wie viele Energieeinheiten notwendig sind, um ein neues Individuum zu erzeugen, wodurch die Größe des Fressfeindclusters entsprechend anwächst.

Dazu sei e_h ein Individuum der Fressfeindart h , $p_{m,n}$ eine Pflanze der Art m , $\mathcal{E}_{min}(e_h) \in \mathbb{N}$ die Mindestanzahl an Energieeinheiten, die erforderlich ist, um ein neues Individuum zu generieren. Ferner sei $R(C_i, t) \in \mathbb{N}$ die bis zum Zeitpunkt t von dem Cluster C_i verzehrte Gesamtenergie, $\eta(C_i) \in \mathbb{N}$ die Essgeschwindigkeit des Clusters C_i , welche beschreibt, wie viele Energieeinheiten pro Zeitschritt konsumiert werden können, und $\mathcal{E}_{m,n}(t)$ das Energieniveau der Pflanze $p_{m,n}$ zum Zeitpunkt t . Zum Zeitpunkt t nimmt das Cluster C_i entsprechend der Essgeschwindigkeit $\eta(C_i)$ die Energie einer Pflanze auf, die sich im selben Gridfeld befindet. Dies führt zu einer Erhöhung der aufgenommenen Energie $R(C_i, t+1)$ für den nächsten Zeitschritt. Formal lässt sich diese Situation durch die folgende Gleichung darstellen:

$$R(C_i, t+1) = R(C_i, t) + \min\{\eta(C_i), \mathcal{E}_{m,n}(t)\} \quad (22)$$

In Worten ausgedrückt wird zu der aufgenommenen Energie zum Zeitpunkt t eine entsprechende Menge an Energieeinheiten hinzugefügt, die sich nach der Essgeschwindigkeit des Clusters richtet. Allerdings kann es sein, dass eine Pflanze zu diesem Zeitpunkt weniger Energie enthält, weswegen eine Minimumsabfrage sinnvoll ist. Das bedeutet, dass, falls die Pflanze weniger Energie besitzt, als das Cluster pro Zeitschritt verzehrt, lediglich die tatsächlich vorhandene Energieeinheiten aufgenommen werden. Auf diese Weise wird gewährleistet, dass ein Cluster nur wirklich verfügbare Energieeinheiten aufnimmt und nicht mehr Energie als der Pflanze zur Verfügung steht.

Der Verzehr von Energie kann als Transfer von Energie der Pflanze auf das Cluster

interpretiert werden. Daher muss die entnommene Energie des Clusters auch von der verfügbaren Energie der Pflanze abgezogen werden. Auf diese Weise lässt sich die Verringerung der Energie der Pflanze für den nächsten Zeitpunkt berechnen durch:

$$- \min\{\eta(C_i), \mathcal{E}_{m,n}(t)\} \quad (23)$$

Auch in diesem Fall wird das Minimum aus der Essgeschwindigkeit des Clusters und der verfügbaren Energie der Pflanze herangezogen, um die verbleibende Energie der Pflanze zum Zeitpunkt $t + 1$ zu bestimmen.

Im Laufe der Zeit sammelt ein Fressfeindcluster Energie in $R(C_i, t)$. Sobald die Mindestenergie $\mathcal{E}_{min}(e_h)$ erreicht wird, die für die Erzeugung eines Nachkommens erforderlich ist, wird mindestens ein neues Individuum generiert, wodurch die Größe des jeweiligen Clusters zunimmt. Die Erhöhung der Clustergröße kann durch die folgende Formel beschrieben werden, wobei die Anzahl der erzeugten Nachkommen $\phi(e_h, t) \in \mathbb{N}$ berechnet wird:

$$\phi(e_h, t) = \left\lfloor \frac{R(C_i, t)}{\mathcal{E}_{min}(e_h)} \right\rfloor \quad (24)$$

Für die Erzeugung neuer Nachkommen wird die aufgenommene Energie $R(C_i, t)$ verwendet, welche entsprechend abgezogen werden muss, um eine doppelte Nutzung der Energie zu vermeiden. Der Abzug der verwendeten Energie von der aufgenommenen Energie lässt sich durch die folgende Formel ausdrücken:

$$R(C_i, t + 1) = R(C_i, t) - \phi(e_h, t) \cdot \mathcal{E}_{min}(e_h) \quad (25)$$

In Worten bedeutet dies, dass die aufgenommene Energie um einen Betrag reduziert wird, der der Energie entspricht, die zur Erzeugung neuer Individuen verwendet wurde. Dabei können die Parameter, wie die Essgeschwindigkeit und die benötigte Energie für einen Nachkommen, so gewählt werden, dass ein Cluster pro Zeitschritt mehr Energie aufnimmt, als es für die Erzeugung eines einzelnen Nachkommens benötigt. Dadurch wird es möglich, in einem Zeitschritt mehrere Nachkommen zu erzeugen. Aus diesem Grund wird die Anzahl der erzeugten Nachkommen $\phi(e_h, t)$ mit der für einen Nachkommen erforderlichen Energie $\mathcal{E}_{min}(e_h)$ multipliziert und der Gesamtwert von der aufgenommenen Energie subtrahiert.

Im letzten Schritt wird die tatsächliche Anzahl der Individuen im entsprechenden Cluster um die Anzahl $\phi(e_h, t)$ der neu erzeugten Individuen erhöht. Die Clustergröße n zum Zeitpunkt $t + 1$ wird entsprechend um $\phi(e_h, t)$ erweitert und lässt sich mit der folgenden Formel beschreiben:

$$n(t + 1) = n(t) + \phi(e_h, t) \quad (26)$$

Um diesen ganzen Sachverhalt an dieser Stelle noch bildhafter zu verdeutlichen, folgt ein Beispiel mit konkreten Werten.

Sei $\mathcal{E}_{1,1}(t) = 120$ die aktuelle Energie der Pflanze $p_{1,1}$, und e_1 die Fressfeindart im Cluster C_1 mit einer Größe von $n(t) = 5$. Weiterhin sei die für ein neues Individuum benötigten Energie $\mathcal{E}_{min}(e_1) = 50$ und die Essgeschwindigkeit $\eta(C_1) = 80$ gegeben.

Schritt 1: Energieverzehr durch das Cluster C_1 :

$$R(C_1, t + 1) = R(C_1, t) + \min\{80, 120\} = 0 + 80 = 80 \quad (27)$$

Die verbleibende Energie der Pflanze beträgt dann $\mathcal{E}_{1,1}(t + 1) = 120 - 80 = 40$

Schritt 2: Erzeugen neuer Individuen:

$$\phi(e_1, t) = \left\lfloor \frac{R(C_1, t)}{\mathcal{E}_{min}(e_1)} \right\rfloor = \left\lfloor \frac{80}{50} \right\rfloor = 1 \quad (28)$$

Nun muss die verwendete Energie, wie besprochen, von der verzehrten Energie subtrahiert werden. Dies ergibt: $R(C_1, t + 1) = R(C_1, t) - \phi(e_1, t) \cdot \mathcal{E}_{min}(e_1) = 80 - 1 \cdot 50 = 30$.

Schritt 3: Aktualisieren der Clustergröße:

$$n(t + 1) = n(t) + \phi(e_1, t) = 5 + 1 = 6 \quad (29)$$

An dieser Stelle wurde das Fressfeindcluster durch den Verzehr pflanzlicher Energie entsprechend der konfigurierten Parameter vergrößert, indem neue Individuen erzeugt wurden.

Ein Fressfeindcluster kann eine bestimmte Größe erreichen, indem es pflanzliche Energie konsumiert. Sei die Clustergröße zum Zeitpunkt $t = 0$ mit $n(0)$ definiert. Sobald $n(t)$ den Wert $2 \cdot n(0)$ erreicht, hat sich die Clustergröße verdoppelt. In diesem Fall spaltet sich ein neues Cluster von dem ursprünglichen Fressfeindcluster ab.

Dabei wird ein Teil der Individuen aus dem ursprünglichen Cluster entnommen und bildet ein separates Cluster derselben Fressfeindart. Formal lässt sich die Größe des neuen Tochterclusters zum Zeitpunkt $t + 1$ wie folgt beschreiben:

$$n_c(t + 1) = \left\lfloor \frac{n(t)}{2} \right\rfloor \quad (30)$$

Dabei steht $n_c(t + 1)$ für die Anzahl der Individuen des neuen Tochterclusters zum Zeitpunkt $t + 1$.

Das ursprüngliche Cluster bleibt bestehen, reduziert sich aber um die Individuen, die in das neue Cluster übergehen. Seine Größe wird durch die folgende Formel bestimmt:

$$n(t + 1) = n(t) - n_c(t + 1) \quad (31)$$

wobei $n(t + 1)$ die neue Anzahl der Individuen des ursprünglichen Clusters ist.

3.7 Von Pflanzen produzierte Substanzen

Eine Pflanze kann potenziell zwei verschiedene Substanzen produzieren, um sich zu verteidigen oder andere Pflanzen vor einem Angriff durch Fressfeinde zu warnen. Zu diesem Zweck wird im Folgenden eine Menge von Substanzen eingeführt:

$$S = \{s_i | i \in \mathbb{N}_0, 0 \leq i \leq 15\} \quad (32)$$

In der späteren Simulation können bis zu 15 verschiedene Substanzen definiert werden. Jede Substanz wird durch ein Tupel bestehend aus einem Namen und einem Typ beschrieben:

$$s_i = (name_i, typ_i) \quad (33)$$

Der Name einer Substanz s_i kann individuell festgelegt werden. Jede Substanz kann entweder als Signalstoff oder als Giftstoff klassifiziert werden, wobei gilt $typ_i \in \{\text{Signal}, \text{Gift}\}$.

3.7.1 Die Interaktionsmatrix der Signalstoffe

Eine Interaktionsmatrix I_{sig} wird verwendet, um die Wechselwirkungen zwischen Pflanzen und Signalstoffen zu beschreiben. Zur Definition dieser Matrix ist es sinnvoll, eine zusätzliche Menge S_{sig} einzuführen. Diese Menge stellt eine Teilmenge der

Gesamtsubstanzmenge S dar und umfasst ausschließlich die Signalstoffe.

$$S_{sig} = \{s_i \in S | typ_i = \text{Signal}\} \quad (34)$$

Die Interaktionsmatrix I_{sig} setzt sich aus zwei separaten Matrizen zusammen:

1. **Emission:** Matrix \mathcal{A} zeigt, welche Pflanzenart einen Signalstoff produzieren oder nicht produzieren kann.
2. **Empfänglichkeit:** Matrix \mathcal{B} zeigt, welche Pflanzenart einen Signalstoff empfangen oder nicht empfangen kann.

Die Matrix, die Informationen zur Produktion von Signalstoffen enthält, ist als $\mathcal{A} \in \{0, 1\}^{|P| \times |S_{sig}|}$ definiert und hat maximal die Dimension 16×15 . Die Einträge dieser Matrix können entweder den Wert 0 oder 1 annehmen:

$$a_{i,k} = \begin{cases} 1 & : p_i \text{ produziert } s_k \\ 0 & : \text{sonst} \end{cases} \quad (35)$$

Ähnlich verhält es sich mit der Matrix zur Empfänglichkeit, die als $\mathcal{B} \in \{0, 1\}^{|P| \times |S_{sig}|}$ definiert ist. Im Gegensatz zur ersten Matrix geben die Einträge in dieser Matrix an, ob eine Pflanzenart in der Lage ist, den jeweiligen Signalstoff wahrzunehmen und darauf zu reagieren.

$$b_{i,k} = \begin{cases} 1 & : p_i \text{ reagiert auf } s_k \\ 0 & : \text{sonst} \end{cases} \quad (36)$$

Aus diesen beiden Matrizen lässt sich die Interaktionsmatrix I_{sig} formal wie folgt definieren:

$$I_{sig} = (\mathcal{A}^\top, \mathcal{B}^\top) \quad (37)$$

Zur Veranschaulichung des Aufbaus der beiden Matrizen folgt ein Minimalbeispiel, das drei Pflanzenarten p_1, p_2 und p_3 und zwei Signalstoffe s_1 und s_2 berücksichtigt.

$$\mathcal{A}^\top = \begin{pmatrix} p_1 & p_2 & p_3 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \begin{matrix} s_1 \\ s_2 \end{matrix} \quad \mathcal{B}^\top = \begin{pmatrix} p_1 & p_2 & p_3 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad \begin{matrix} s_1 \\ s_2 \end{matrix} \quad (38)$$

Die Matrix \mathcal{A} zeigt, dass p_1 ausschließlich den Signalstoff s_1 , p_2 ausschließlich den Signalstoff s_2 und p_3 beide Signalstoffe produziert.

Die Matrix \mathcal{B} , welche die Empfänglichkeit der Pflanzenarten für Signalstoffe beschreibt, zeigt in diesem Beispiel, dass p_1 alle Signalstoffe empfangen kann, p_2 ausschließlich s_2 und p_3 nur s_1 .

3.7.2 Bedingungen zur Freisetzung von Signalstoffen

Damit ein Signalstoff von einer Pflanze produziert und freigelassen werden kann, ist ein Auslöser erforderlich. Es können jedoch unterschiedliche Cluster in verschiedenen Größen denselben Signalstoff auslösen. Daher ist es sinnvoll, für die Auslösebedingungen eine Menge Q_k zu definieren. Diese Menge umfasst die Kombination von Fressfeindarten und der Mindestanzahl an Individuen in einem Cluster, die notwendig sind, um einen Signalstoff zu aktivieren.

$$Q_k = \{(e_i, n_{i,min}) | e_i \in E, n_{i,min} \in \mathbb{N}\} \quad (39)$$

Die Menge Q_k ist direkt mit den Freisetzungsbedingungen des Signalstoffs s_k verknüpft.

Damit der Signalstoff freigesetzt werden kann, muss mindestens eine der Bedingungen aus Q_k erfüllt sein. Dies bedeutet konkret, dass die entsprechende Fressfeindart e_h in ausreichender Anzahl auf der Position der Pflanze vorhanden sein muss. Formal ausgedrückt: Es existiert mindestens ein Paar $(e_i, n_{i,min}) \in Q_k$, für das gilt: $\exists(e_i, n_{i,min}) \in Q_k$ mit $C_j = (e_h, n)$ sodass $e_h = e_i$ und $n \geq n_{i,min}$.

Um präzise zu bestimmen, wann ein Signalstoff freigesetzt wird, wird die Funktion f_r eingeführt. Diese Funktion nimmt je nach Situation den Wert 1 an, wenn der Signalstoff ausgelöst wird, oder 0, wenn keine Freisetzung erfolgt.

$$f_r(s_k, Y) = \begin{cases} 1 : & \exists(e_i, n_{i,min}) \in Q_k, \exists C_j \in Y \text{ mit } e_h = e_i, n \geq n_{i,min} \\ 0 : & \text{sonst} \end{cases} \quad (40)$$

Dabei umfasst Y , wie zuvor definiert, eine spezifische Teilmenge aller Fressfeindcluster, die sich auf dem Feld der Pflanze befinden.

In diesem Zusammenhang wird geprüft, ob mindestens eine Bedingung in Q_k durch mindestens ein Fressfeindcluster in Y erfüllt wird.

Um die Funktionsweise der Funktion anschaulich darzustellen, wird im Folgenden ein Beispiel mit drei Clustern betrachtet. Es sei $Y = \{C_1, C_2, C_3\}$ die Menge der Fressfeindcluster, die sich auf dem Feld einer Pflanze befinden. Die Cluster sind wie folgt definiert: $C_1 = (e_1, 4)$, $C_2 = (e_2, 6)$ und $C_3 = (e_3, 3)$. Darüber hinaus seien die Bedingungen für das Auslösen der Produktion des Signalstoffs s_k durch die Menge $Q_k = \{(e_1, 4), (e_2, 7), (e_3, 10)\}$ gegeben.

Die Funktion $f_r(s_k, \{C_1, C_2, C_3\})$ überprüft, ob der Signalstoff s_k ausgelöst wird, indem sie die vorliegenden Cluster C_1, C_2, C_3 mit den Bedingungen aus Q_k vergleicht und prüft, ob mindestens eine Bedingung erfüllt ist. Im vorliegenden Beispiel gilt für C_1 : $n = 4 \geq 4 = n_{1,min}$, sodass die Bedingung erfüllt ist. Die Cluster C_2 und C_3 erfüllen die Bedingungen hingegen nicht, da C_2 die Bedingung mit e_2 in Q_k nicht erfüllt, da $n = 6 < 7 = n_{2,min}$ gilt, und C_3 die erforderlichen Mindestanzahl $n_{3,min} = 10$ für e_3 nicht erreicht. Da C_1 eine Bedingung erfüllt lautet das Ergebnis der Funktion: $f_r(s_k, \{C_1, C_2, C_3\}) = 1$.

In einem alternativen Szenario, bei dem die Bedingungen in Q_k durch $\{(e_1, 5), (e_2, 7), (e_3, 4)\}$ festgelegt sind, erfüllt keines der Cluster C_1, C_2, C_3 die Voraussetzungen zur Auslösung des Signalstoffs s_k . Cluster $C_1 = (e_1, 4)$ verfehlt die Bedingung $(e_1, 5)$, da die Anzahl der Individuen kleiner als die erforderliche Mindestanzahl ist. Cluster $C_2 = (e_2, 6)$ scheitert an der Bedingung $(e_2, 7)$, da auch hier die geforderte Anzahl nicht erfüllt ist. Cluster $C_3 = (e_3, 3)$ erfüllt die Bedingung $(e_3, 4)$ ebenfalls nicht, da die vorhandene Anzahl an Individuen kleiner ist als die Mindestanzahl.

Da kein Cluster die Bedingungen aus Q_k erfüllt, lautete das Ergebnis der Funktion: $f_r(s_k, \{C_1, C_2, C_3\}) = 0$.

3.7.3 Mechanismen der Signalstoff-Ausbreitung

Ein Signalstoff kann sich auf zwei verschiedene Arten über das Grid ausbreiten: einerseits über die Luft und andererseits über die vorhandenen Gridverbindungen zwischen den Pflanzen. Es sei M eine Menge, die beide Ausbreitungsarten umfasst:

$$M = \{\text{Luft, Gridverbindung}\} \quad (41)$$

Für jeden Signalstoff s_k muss nun eine Zuordnung hinsichtlich der Ausbreitungsart vorgenommen werden. Zu diesem Zweck wird die Funktion f_s definiert:

$$f_s : S_{sig} \rightarrow M \quad (42)$$

Für jeden Signalstoff s_k gilt folgendes:

$$f_s(s_k) = \begin{cases} \text{Luft} & : \text{Ausbreitung von } s_k \text{ via Luft} \\ \text{Gridverbindung} & : \text{Ausbreitung von } s_k \text{ via Gridverbindung} \end{cases} \quad (43)$$

In Worten modelliert die Funktion f_s die Ausbreitung des Signalstoffs und ordnet jedem Signalstoff genau eine Art der Verbreitung zu. Dabei kann entweder eine freie Ausbreitung über die Luft oder eine gerichtete Ausbreitung über die Gridverbindungen erfolgen.

3.7.4 Die ungezielte Ausbreitung via Luft

Es sei $l((x^*, y^*), t^*)$ die Position der auslösenden Pflanze zum Zeitpunkt $t^* \in T$.

Ein Signalstoff wird nun über immer größer werdende Kreise über dem Grid verteilt, wobei die Ursprungspflanze den Mittelpunkt des Kreises bildet. Der Kreis lässt sich durch den Radius $r(t)$ beschreiben, der angibt, wie groß der Kreis nach $t - t^*$ Zeitschritten ist.

Weiterhin wird ein Parameter $t_l \in \mathbb{N}_{\geq 1}$ definiert, der als Intervall betrachtet werden kann und dem Nutzer angibt, nach wie vielen Zeitschritten der Radius um eine Gridebene vergrößert wird. Der Radius vergrößert sich genau dann, wenn die Gleichung $(t - t^*) \bmod t_l = 0$ erfüllt ist, d.h. wenn das Intervall t_l einmal durchlaufen wurde. Um eine kreisförmige Ausbreitung modellieren zu können, wird an dieser Stelle die euklidische Distanz verwendet, die wie folgt definiert ist:

$$d((x, y), (x^*, y^*)) = \sqrt{(x - x^*)^2 + (y - y^*)^2} \quad (44)$$

Die euklidische Distanz stellt sicher, dass alle Gridfelder (x, y) innerhalb des Kreises um (x^*, y^*) mit eingeschlossen werden. Weiterführend können nun alle involvierten Gridfelder zum Zeitpunkt t in einer Menge $G_l(t)$ gesammelt werden, die wie folgt definiert ist:

$$G_l(t) = \{(x, y) \in G \mid d((x, y), (x^*, y^*)) \leq r(t)\} \quad (45)$$

Um diesen Sachverhalt veranschaulicht darzustellen folgt ein Beispiel für $t^* = 3$ und $t_l = 2$ mit einer Ursprungspflanze auf dem Feld $(5, 5)$.

Zeitpunkt t	$(t - t^*) \bmod t_l = 0$	Radius $r(t)$	$G_l(t)$
3	0	0	$\{(5, 5)\}$
4	1	0	$\{(5, 5)\}$
5	0	1	$\{(4, 4), (4, 5), \dots, (6, 6)\}$
6	1	1	$\{(4, 4), (4, 5), \dots, (6, 6)\}$
7	0	2	$\{(3, 3), (3, 4), \dots, (7, 7)\}$
8	1	2	$\{(3, 3), (3, 4), \dots, (7, 7)\}$
9	0	3	$\{(2, 3), (2, 4), \dots, (8, 7)\}$

Tabelle 1: Anwendungsbeispiel für das Erzeugen eines Signalstoffradius

Es ist zu beobachten, dass der Radius alle zwei Zeitschritte seit der Auslösung um eine Stufe vergrößert wird, wodurch immer mehr Gridfelder in den Radius einbezogen werden.

Da nicht jede Pflanze alle Signalstoffe produzieren kann, sollte eine weitere Einschränkung bzgl. der Menge $G_l(t)$ gemacht werden, die nur die Pflanzen umfasst, die in der Lage sind, den betrachteten Signalstoff zu produzieren. Zu diesem Zweck kann eine Untermenge $G'_l(t, s_k) \subseteq G_l(t)$ gebildet werden, die wie folgt definiert ist:

$$G'_l(t, s_k) = \{(x, y) \in G_l(t) \mid \exists x, \exists y, (l((x, y), t) = (p_{i,j}, K) \wedge K \in \mathcal{P}(C)), \\ (a_{i,k} = 1 \vee b_{j,k} = 1)\} \quad (46)$$

Diese Menge umfasst nun alle Pflanzen, die sich im Radius $r(t)$ befinden und den Signalstoff s_k produzieren oder empfangen können. Dies wird durch den Zugriff auf die entsprechenden Einträge der Matrizen \mathcal{A} bzw. \mathcal{B} gewährleistet.

3.7.5 Die gezielte Ausbreitung via Gridverbindung

Eine Gridverbindung bzw. symbiotische Verbindung $\mathcal{V}(p_{i,j}, p_{k,m})$ kann zwischen zwei benachbarten Pflanzen ausgebildet werden, wobei $p_{i,j}$ die Pflanze an der Position

$l((x_1, y_1), t)$ und $p_{k,m}$ die Pflanze an der Position $l((x_2, y_2), t)$ darstellt.

$$\mathcal{V}(p_{i,j}, p_{k,m}) : |x_1 - x_2| + |y_1 - y_2| = 1 \quad (47)$$

Jede Pflanze kann bis zu vier benachbarte Pflanzen haben, woraus sich ableiten lässt, dass es entweder keine oder bis zu vier Gridverbindungen geben kann. Die Gridverbindungen für eine Pflanze $p_{i,j}$ lassen sich als Menge wie folgt definieren:

$$V_{i,j} = \{\mathcal{V}(p_{i,j}, p_{k,m}) ||x_1 - x_2| + |y_1 - y_2| = 1\} \quad (48)$$

Die Ausbreitung von Signalstoffen erfolgt ähnlich wie die Ausbreitung über die Luft. Es gibt ein Intervall $t_g \in \mathbb{N}$, das beschreibt, wie lange es dauert, bis ein Signalstoff bei der empfangenden Pflanze ankommt. Weiterhin sei t die aktuelle Zeit und t^* der Zeitpunkt, an dem der Signalstoff ausgelöst wird.

Ein Signalstoff erreicht die benachbarte Pflanze genau dann, wenn $(t - t^*) \bmod t_g = 0$ gilt, was bedeutet, dass ein Durchlauf des Intervalls t_g erfolgt ist. Zudem können mehrere Pflanzen durch mehrere Gridverbindungen in Reihe geschaltet sein, sodass eine maximale Anzahl fortlaufender Gridverbindungen existiert, die durch $g(t) \in \mathbb{N}$ beschrieben wird.

Es sei $d_g((x, y), (x^*, y^*))$ die Manhattan-Distanz, zwischen zwei Feldern (x^*, y^*) und (x, y) . Die Menge der Gridfelder $G_g(t)$, die zum aktuellen Zeitpunkt innerhalb der Reichweite $g(t)$ von der Position (x^*, y^*) erreicht sind, ist definiert als:

$$\begin{aligned} G_g(t) = \{(x, y) \in G &| \exists x^*, \exists y^*, (l((x^*, y^*), t) = (p_{i,j}, K_1) \wedge K_1 \in \mathcal{P}(C)), \\ &\exists x, \exists y, (l((x, y), t) = (p_{k,m}, K_2) \wedge K_2 \in \mathcal{P}(C)), \\ &d_g((x, y), (x^*, y^*)) \leq g(t), \exists \mathcal{V}(p_{i,j}, p_{k,m})\} \end{aligned} \quad (49)$$

In Worten ausgedrückt, handelt es sich bei $G_g(t)$ um die Felder, bei denen die Manhattan-Distanz zwischen den Feldern kleiner oder gleich der maximalen Reichweite der Gridverbindungen ist. Weiterhin muss auf den betrachteten Feldern jeweils eine Pflanze vorhanden sein, die durch Gridverbindungen miteinander verbunden sind. Hier ein Beispiel, wie ein Signalstoff über die Gridverbindungen an benachbarte Pflanzen gesendet wird. Gegeben sei $t^* = 3$, $t_g = 2$ und eine Ursprungspflanze auf $(x^*, y^*) = (5, 5)$.

Zeitpunkt t	$(t - t^*) \bmod t_g$	$d_g \leq g(t)$	$G_g(t)$
3	0	$0 \leq 1$	$\{(5, 5)\}$
Beispiel 1: 4 Gridverbindungen zu $(5, 4)$, $(4, 5)$, $(5, 6)$ und $(6, 5)$			
4	1	$0 \leq 1$	$\{(5, 5)\}$
5	0	$1 \leq 1$	$\{(5, 5), (5, 4), (4, 5), (5, 6), (6, 5)\}$
Beispiel 2: 2 Gridverbindungen zu $(5, 4)$ und $(5, 6)$			
4	1	$0 \leq 1$	$\{(5, 5)\}$
5	0	$1 \leq 1$	$\{(5, 5), (5, 4), (5, 6)\}$
Beispiel 3: Fortlaufenden Gridverbindungen über $(5, 6)$ nach $(5, 7)$			
4	1	$0 \leq 2$	$\{(5, 5)\}$
5	0	$1 \leq 2$	$\{(5, 5), (5, 6)\}$
6	1	$1 \leq 2$	$\{(5, 5), (5, 6)\}$
7	0	$2 \leq 2$	$\{(5, 5), (5, 6), (5, 7)\}$

Tabelle 2: Anwendungsbeispiele für die Ausbreitung über Gridverbindungen

Es ist zu erkennen, dass jedes Mal, wenn das Intervall durchlaufen ist, der Signalstoff bei der verbundenen Nachbarpflanze ankommt.

Auch bei der gezielten Ausbreitung über die Gridverbindungen muss eine Einschränkung der Menge $G_g(t)$ vorgenommen werden, sodass nur die Pflanzen berücksichtigt werden, die den gesendeten Signalstoff auch empfangen können. Zu diesem Zweck sei $G'_g(t, s_k) \subseteq G_g(t)$ eine Untermenge aller verbundenen Pflanzen, die wie folgt definiert ist:

$$G'_g(t, s_k) = \{(x, y) \in G_g(t) \mid \exists x, \exists y, (l((x, y), t) = (p_{m,n}, K) \wedge K \in \mathcal{P}(C)), \\ (a_{i,k} = 1 \wedge b_{j,k} = 1)\} \quad (50)$$

3.7.6 Start und Beendigung der Signalstoffproduktion

Es sei $Y \in \mathcal{P}(C)$ eine Teilmenge, die die Fressfeindcluster enthält, die sich zum Zeitpunkt t auf dem Feld (x, y) befinden, auf dem ebenfalls eine Pflanze $p_{i,j}$ platziert ist, sodass $l((x, y), t) = (p_{i,j}, Y)$ gilt. In diesem Fall wird die Pflanze von den Fressfeinden angegriffen.

Die Produktion eines Signalstoffs wird aufrechterhalten, solange folgende Bedingung erfüllt ist:

$$\exists Y \in \mathcal{P}(C), \exists C_j \in Y \text{ mit } n \geq n_{i,min} \text{ für das Paar } (e_i, n_{i,min}) \in Q_k \quad (51)$$

In Worten ausgedrückt muss mit der Menge $Y \in \mathcal{P}(C)$ sichergestellt werden, dass die Pflanze $p_{i,j}$ zum Zeitpunkt t von mindestens einem Fressfeindcluster aus der Menge Y angegriffen wird. Weiterhin muss das Fressfeindcluster die richtige Fressfeindart und eine Mindestgröße vorweisen, damit die Anforderungen Q_k zur Produktion eines Signalstoffs s_k erfüllt sind. Wenn diese Bedingungen für den Signalstoff s_k nicht erfüllt sind, wird die Produktion des Signalstoffs nicht ausgelöst.

Zusätzlich ist ein Signalstoff nicht sofort verfügbar, sondern erfordert eine konfigurierbare Produktionszeit. Sei $t_k \in T$ der Zeitpunkt, zu dem die Produktion des Signalstoffs startet, und $\mathcal{T}(s_k) \in \mathbb{N}$ die Anzahl der benötigten Produktionsschritte. Daraus folgt, dass der Signalstoff fertig produziert ist, sobald für den aktuellen Zeitpunkt t die Bedingung $t = t_k + \mathcal{T}(s_k)$ erfüllt ist.

Die Produktion eines Signalstoffs s_k endet genau dann, wenn für alle Y und für jedes Fressfeindcluster $C_i \in Y$ gilt, dass keine Fressfeindart die Mindestanforderungen der Auslösebedingung Q_k erfüllt. Dies führt zu folgender Bedingung:

$$\exists Y \in \mathcal{P}(C), \forall C_j \in Y \neg(n \geq n_{i,min} \text{ für ein Paar } (e_i, n_{i,min}) \in Q_k) \quad (52)$$

Es gibt zu keinem Zeitpunkt t ein angreifendes Fressfeindcluster C_j aus der Menge Y , das die Mindestanforderung erfüllt. Ab diesem Zeitpunkt wird dann die Produktion des Signalstoffs s_k eingestellt und damit wird die Pflanze zum Zeitpunkt $t + 1$ signalstofffrei bzgl. s_k sein.

3.7.7 Die Nachwirkzeit und der Wirkbereich eines Signalstoffs

Ein Signalstoff verschwindet nicht sofort nach dem Einstellen der Produktion, sondern ist über einen definierbaren Zeitraum noch darüber hinaus verfügbar.

Es sei $T_k \in \mathbb{N}$ die Anzahl an Zeitschritten, die ein Signalstoff nach der Beendigung der Produktion weiterhin verfügbar bleibt. Diese Zeitspanne wird als konstante Zahl definiert. Die Nachwirkzeit ermöglicht es der Pflanze, bei einem erneuten Angriff

innerhalb dieses Zeitraums schneller zu reagieren, ohne den gesamten Produktionsprozess des Signalstoffs s_k erneut durchlaufen zu müssen. Dies trägt möglicherweise dazu bei, dass die Pflanze überlebt und den Fressfeind einige Zeitschritte früher vertreiben oder verletzen kann.

Durch das Aussenden eines Signalstoffs an andere Pflanzen entsteht ein signalstoffspezifischer Wirkbereich, der unterschiedliche Bereiche des Grids umfasst und davon abhängt, ob der Signalstoff über die Luft oder über Gridverbindungen verbreitet wird. Formal wird der Wirkbereich $W(s_k, p_{i,j}, t)$ einer Pflanze $p_{i,j}$ für einen Signalstoff s_k als die Menge aller Gridfelder definiert, die aus $G'_l(t)$ im Fall der Luftverbreitung und aus $G'_g(t)$ im Fall der Verbreitung über die Gridverbindungen zum Zeitpunkt t resultieren.

$$W(s_k, p_{i,j}, t) = G'_l(t) \cup G'_g(t) \quad (53)$$

Der Wirkungsbereich eines Signalstoffs s_k , der von der Pflanze $p_{i,j}$ ausgesendet wird, verschwindet nicht unmittelbar nach Beendigung der Produktion, sondern erst, wenn die Nachwirkzeit T_k abgelaufen ist.

Die Nachwirkzeit endet genau dann, wenn $t = t_e + T_k$ gilt, wobei $t_e \in T$ der Zeitpunkt ist, an dem die Produktion des Signalstoffs eingestellt wird.

Nach Ablauf der Nachwirkzeit muss der durch den Signalstoff entstandene Wirkbereich entfernt werden. Das bedeutet, dass der Signalstoff auf den zuvor involvierten Gridfeldern in $W(s_k, p_{i,j}, t)$ nicht mehr aktiv ist und aus der Menge des Wirkbereichs entfernt wird. Da eine Pflanze verschiedene Signalstoffe produzieren kann und unterschiedliche Pflanzen denselben Signalstoff s_k freisetzen können, kann es durch die Ausbreitungsmöglichkeiten zu Überlappungen auf dem Grid kommen. Der überlappende Wirkbereich $W^*(s_k, t)$ des Signalstoffs s_k zum Zeitpunkt t wird als die Menge aller Gridfelder definiert, die durch die Wirkbereiche von mindestens einer Pflanze beeinflusst werden, die diesen Signalstoff zu diesem Zeitpunkt freisetzen.

Wenn die Wirkbereiche verschiedener Pflanzen sich auf dem Gridfeld überschneiden, wird dieses Feld ebenfalls in den überlappenden Wirkbereich aufgenommen. Dies führt zu folgender formaler Beschreibung:

$$W^*(s_k, t) = \bigcup_{p_{i,j} \in O_k(t)} W(s_k, p_{i,j}, t) \quad (54)$$

Mit $O_k(t) = \{p_{i,j} | a_{i,k} = 1\}$ wird die Menge aller Pflanzen bezeichnet, die den Signalstoff s_k zum Zeitpunkt t produzieren und freisetzen. Der signalstoffspezifische Wirkbereich einer Pflanze $p_{i,j}$, die den Signalstoff s_k freisetzt, ist weiterhin durch $W(s_k, p_{i,j}, t)$ definiert, wie zuvor beschrieben.

Ein Signalstoff s_k wird nach Ablauf der Nachwirkzeit von den betroffenen Gridfeldern entfernt. Der überlappende Wirkbereich $W^*(s_k, t)$ bleibt jedoch erhalten, sofern andere Pflanzen weiterhin Wirkbereiche von s_k zum Zeitpunkt t aktiv halten. Der spezifische Wirkbereich von s_k , bei dem die Nachwirkzeit abgelaufen ist, wird nicht mehr im überlappenden Wirkbereich berücksichtigt.

Anders ausgedrückt, die entsprechenden Gridfelder werden aus $W^*(s_k, t)$ entfernt. Dies führt zur folgenden Beschreibung des Entfernen eines spezifischen Wirkbereichs:

$$W^*(s_k, t + 1) = W^*(s_k, t) \setminus W(s_k, p_{i,j}, t) \quad (55)$$

Mit $W(s_k, p_{i,j}, t)$ als den signalstoffspezifischen Wirkbereich der Pflanze $p_{i,j}$ zum Zeitpunkt t und $W^*(s_k, t + 1)$ als überlappender Wirkbereich zum Zeitpunkt $t + 1$, wobei die Gridfelder, die bereits in $W(s_k, p_{i,j}, t)$ enthalten sind aus $W^*(s_k, t + 1)$ entfernt werden.

In einer kurzen Zusammenfassung wird der signalstoffspezifische Wirkbereich der Pflanze $p_{i,j}$ entfernt, wobei andere Wirkbereiche davon unberührt bleiben. Dies betrifft sowohl die Wirkbereiche anderer aktiver Signalstoffe an der Pflanze $p_{i,j}$ als auch die Wirkbereiche anderer Pflanzen, für denselben Signalstoff s_k , die nicht beeinträchtigt werden.

3.7.8 Die Interaktionsmatrix der Giftstoffe

Ähnlich wie bei den Signalstoffen existiert auch eine Menge von Giftstoffen, die ebenfalls als Untermenge der Substanzen dargestellt werden kann. Daraus ergibt sich die folgende formale Definition:

$$S_{tox} = \{s_i \in S | typ_i = \text{Gift}\} \quad (56)$$

Im Gegensatz zu den Signalstoffen besteht die Interaktionsmatrix der Giftstoffe nicht aus zwei separaten Matrizen. Stattdessen wird eine einzelne, umfassende Matrix

verwendet, die die Kombinationen von Signalstoffen und Arten von Fressfeinden abbildet, bei denen die Produktion des Giftstoffs s_x induziert wird.

Eine Kombination, die die Produktion eines Giftstoffs s_x auslöst, kann durch eine Interaktionsmenge $\mathcal{J}(s_x)$ wie folgt beschrieben werden:

$$\mathcal{J}(s_x) = \{(s_k, e_i, n_{i,min}) | s_k \in S_{sig}, e_i \in E, n_{i,min} \in \mathbb{N}\} \quad (57)$$

Eine Kombination, die zur Produktion eines Giftstoffs s_x führt, besteht aus drei wesentlichen Bestandteilen: Erstens muss ein spezifischer Signalstoff an der Pflanze vorhanden sein. Zweitens muss eine bestimmte Fressfeindart vorliegen. Drittens muss diese Fressfeindart in ausreichender Anzahl innerhalb eines Clusters auf demselben Gridfeld wie die Pflanze vorkommen. Diese drei Bedingungen müssen gleichzeitig erfüllt sein. Nur wenn alle Kriterien simultan gegeben sind, wird die Produktion des Giftstoffs s_x ausgelöst.

Um genauer auf die Interaktionsmatrix T_{tox} der Giftstoffe einzugehen: Sie ist mit dem jeweiligen Giftstoff s_x verknüpft und umfasst drei Spalten, die der definierten Interaktionsmenge $\mathcal{J}(s_x)$ entsprechen. Die Anzahl der Zeilen variiert, je nach der Anzahl der festgelegten Trigger, die zur Auslösung der Giftstoffproduktion erforderlich sind.

Ein Interaktionsmatrix für ein Giftstoff kann wie folgt aufgebaut sein:

$$I_{tox}(s_x) = \begin{pmatrix} s_1 & e_1 & 3 \\ s_2 & e_2 & 5 \\ s_3 & e_1 & 4 \\ s_1 & e_3 & 2 \\ \vdots & \vdots & \vdots \end{pmatrix} \quad (58)$$

Für die erste Kombination bedeutet dies, dass der Giftstoff s_x ausgelöst und die Produktion gestartet wird, wenn der Signalstoff s_1 vorhanden ist und gleichzeitig ein Fressfeindcluster mit der Art e_1 mit mindestens 3 Individuen vorliegt.

3.7.9 Der Start der Giftstoffproduktion

Ein Giftstoff muss nicht von jeder definierten Pflanzenart produziert werden können. Mit anderen Worten: Es gibt Pflanzenarten, die den Giftstoff produzieren können, und solche, die ihn überhaupt nicht produzieren. Aus diesem Grund existiert eine Teilmenge der bereits definierten Menge P , die sich auf die Pflanzenarten bezieht. Diese Teilmenge sei $P_{s_x} \subseteq P$ und umfasst alle Pflanzenarten aus P , die den Giftstoff s_x aktiv produzieren können. Formal wird diese Teilmenge wie folgt definiert:

$$P_{s_x} = \{p_i \in P \mid p_i \text{ kann Giftstoff } s_x \text{ produzieren}\} \quad (59)$$

Ähnlich wie ein Signalstoff ist auch ein Giftstoff nicht unmittelbar nach seiner Auslösung aktiv. Er benötigt eine gewisse Zeit, um produziert zu werden, und wird erst nach Ablauf dieser Zeitspanne wirksam. Erst ab diesem Zeitpunkt kann der Giftstoff Einfluss auf die Fressfeinde ausüben.

Die Produktion des Giftstoffs wird nur dann ausgelöst, wenn alle Bedingungen erfüllt sind: Die Pflanze gehört zur Menge P_{s_x} und mindestens eine Kombination aus der Interaktionsmatrix $I_{tox}(s_x)$ trifft zu, wobei alle drei Elemente der Kombination erfüllt sein müssen.

Wird die Produktion eines Giftstoffs s_x zum Zeitpunkt $t_x \in T$ ausgelöst und benötigt $\mathcal{T}(s_x) \in \mathbb{N}$ Zeitschritte, bis der Giftstoff aktiv ist, dann beginnt er ab dem Zeitschritt $t = t_x + \mathcal{T}(s_x)$, an der Pflanze gegen die Fressfeinde zu wirken, die auf diesen spezifischen Giftstoff reagieren.

Zum Zeitpunkt, an dem der Giftstoff s_x seine Wirkung entfaltet, wird die betroffene Pflanze $p_{i,j}$ auf dem Grid durch $p_{i,j}^*$ ersetzt - eine Pflanze, bei der die Produktionszeit des Giftstoffs abgeschlossen ist.

3.7.10 Die Wirkung der Giftstoffe

Für jeden Giftstoff s_x existiert eine konfigurierbare Funktion $\alpha(s_x)$, die die Fressfeindarten bestimmen, auf die der Giftstoff wirkt. Formal lässt sich diese Funktion wie folgt beschreiben:

$$\alpha(s_x) = \{e_i \in E \mid s_x \text{ wirkt auf } e_i\} \quad (60)$$

Für jedes Fressfeindcluster C_i , das sich auf dem Gridfeld befindet, auf dem eine giftige Pflanze $p_{i,j}^*$ steht, kann das Cluster auf zwei verschiedene Arten beeinflusst werden. Welche Wirkung eintritt, hängt davon ab, wie der Giftstoff zu Beginn definiert wurde.

Die Wirkung eines Giftstoffs s_x auf ein Fressfeindcluster C_i mit der Art e_h wird durch die folgende Funktion beschrieben:

$$\mathcal{W}(s_x, C_i) = \begin{cases} \text{Vergiftung von } C_i & : s_x \text{ tödlich und wirkt auf } e_h \\ \text{Ablenkung von } C_i & : s_x \text{ vertreibend und wirkt auf } e_h \end{cases} \quad (61)$$

In Worten ausgedrückt, gibt es zwei verschiedene Arten, wie ein Giftstoff auf die Fressfeinde wirken kann. Einerseits kann ein Giftstoff eine ablenkende Wirkung haben, bei der Fressfeinde lediglich von der Pflanze vertrieben werden, ohne dass Individuen des betroffenen Clusters sterben. Andererseits kann ein Giftstoff eine tödliche Wirkung besitzen.

Jeder Giftstoff s_x verfügt über einen konfigurierbaren Parameter, der die Eliminationsrate pro Zeitschritt angibt und definiert, wie viele Individuen eines Fressfeindclusters in einem Zeitschritt sterben. Sei $\beta(s_x, C_i)$ die Eliminationsrate, die die Anzahl der Individuen angibt, die pro Zeitschritt aufgrund des Giftstoffs sterben. Weiterhin sei $m \in \mathbb{N} \setminus \{0\}$ die exakte Anzahl der Individuen, die pro Zeitschritt eliminiert werden, und $n \in \mathbb{N}$ die Gesamtanzahl der Individuen im betroffenen Cluster. Im Fall einer ablenkenden Wirkung gilt $\beta(s_x, C_i) = 0$, da keine Individuen des Clusters sterben. Diese Zusammenhänge lassen sich formal wie folgt beschreiben:

$$\beta(s_x, C_i) = \begin{cases} m & : \mathcal{W}(s_x, C_i) \text{ tödlich} \\ 0 & : \mathcal{W}(s_x, C_i) \text{ vertreibend} \end{cases} \quad (62)$$

Im Fall einer tödlichen Wirkung des Giftstoffs s_x wird die Anzahl der zu eliminierenden Individuen so angepasst, dass sie nie die Gesamtanzahl der im Cluster vorhandenen Individuen n überschreitet.

3.7.11 Der Einfluss auf die Fressfeinde durch Giftstoffe

Abhängig von der Art und Weise, wie der Giftstoff die Pflanze $p_{i,j}^*$ vor Fressfeinden verteidigt, ergibt sich die Auswirkung auf das Fressfeindcluster.

Ein Cluster $C_i = (e_h, n)$, das sich auf demselben Gridfeld wie eine giftige Pflanze $p_{i,j}^*$ befindet und dessen Fressfeindart $e_h \in \alpha(s_x)$ ist, wird um $\beta(s_x, C_i)$ reduziert. Die neue Anzahl der Individuen im betroffenen Fressfeindcluster lässt sich daher wie folgt berechnen:

$$n(t+1) = \max(n(t) - \beta(s_x, C_i), 0) \quad (63)$$

Falls $\beta(s_x, C_i) \geq n(t)$ ist, gilt $n(t+1) = 0$, da die Eliminationsrate potenziell größer ist als die aktuelle Anzahl an Individuen. In diesem Fall erlischt das Fressfeindcluster und wird aus dem Grid entfernt.

Individuen sterben, solange das Cluster noch existiert, bis es entweder ausgestorben ist oder die Pflanze ihre benötigte Energie zum Überleben verbraucht hat. An diesem Punkt stirbt die Pflanze, und das Cluster kommt nicht mehr mit dem Gift in Kontakt, sodass keine weiteren Individuen mehr sterben.

Vertreibende Giftstoffe bewirken, dass ein Fressfeindcluster seinen Fokus auf die Pflanze verliert. Sei C_i ein Cluster der Fressfeindart e_h , das zu einem Zeitpunkt t auf eine Pflanze $p_{m,n}$ fokussiert ist und dieses Ziel anvisiert hat. Der Fokus eines Clusters kann als Funktion wie folgt beschrieben werden:

$$\delta(C_i, t) = p_{m,n} \quad (64)$$

Wenn der Giftstoff s_x mit einer weglenkenden Wirkung an der Pflanze aktiv wird, wird die Wahrnehmungsfunktion wie folgt angepasst:

$$\delta(C_i, t+1) \neq p_{m,n} \quad (65)$$

In Worten bedeutet dies, dass zum Zeitpunkt t der Giftstoff vollständig produziert wurde und nun seine Wirkung entfaltet. Infolgedessen verliert das Cluster C_i zum Zeitpunkt $t+1$ seinen Fokus auf die Pflanze $p_{m,n}$ und muss sich neu orientieren, um eine Pflanze zu finden, die zu diesem Zeitpunkt nicht durch den Giftstoff s_x blockiert ist.

An dieser Stelle wird der Algorithmus zur Suche nach der nächsten Pflanze aus Kapitel 3.4.1 aufgegriffen und leicht angepasst, um die Menge der möglichen Pflanzen einzuschränken.

Ein Fressfeindcluster soll ein Gedächtnis besitzen, in dem es sich merkt, von welchem Giftstoff es vertrieben wurde. Das bedeutet, dass an dieser Stelle nicht mehr das nächste Ziel aus der Menge aller möglichen Pflanzen ausgewählt wird, sondern aus einer Teilmenge der Pflanzen, die nicht durch den vertreibenden Giftstoff blockiert sind.

Auch hier sind ein Grid G , ein Startpunkt $\sigma = (x_s, y_s) \in G$ als die aktuelle Position des Fressfeindclusters C_i und die Menge der Bewegungsrichtungen Γ wie gewohnt gegeben. Zusätzlich wird nun eine Menge P_f benötigt, die alle Positionen des Grids enthält, auf denen eine Pflanze steht und die nicht durch den Giftstoff s_x blockiert sind, durch den das Fressfeindcluster C_i vertrieben wurde.

Das Ziel des Algorithmus besteht nun darin, den kürzesten Pfad Ω zur nächsten Pflanze zu finden, die sich in der Teilmenge P_f befindet.

Der Algorithmus besteht auch hier wieder aus einem Hauptteil, der eine Schleife enthält, in der überprüft wird, ob die Warteschlange q nicht leer ist. In jeder Iteration wird der vorderste Punkt $\psi = (x_p, y_p) \in G$ aus der Warteschlange q entnommen. Falls $\psi \in P_f$, wurde eine Position mit einer Pflanze gefunden, die nicht durch einen Giftstoff s_x blockiert ist, der den Fressfeind zuvor von einer anderen Pflanze vertrieben hat. In diesem Fall kann die Schleife beendet und die Pfadrekonstruktion eingeleitet werden. Sollte die gefundene Position $\psi \notin P_f$ erfüllen, dann wird der Algorithmus, wie in Kapitel 3.4.1 beschrieben, fortgesetzt, um die nächste Pflanze zu finden. Das weitere Vorgehen des Algorithmus bleibt weiterhin unverändert, sodass die detaillierte Vorgehensweise im entsprechenden Kapitel schon behandelt wurde.

3.7.12 Ende der Giftproduktion und Auswirkungen

Im Gegensatz zu Signalstoffen gibt es bei Giftstoffen keine Nachwirkzeit. Die Verfügbarkeit eines Giftstoffs s_x endet, sobald kein Fressfeindcluster C_i mehr vorhanden ist, das für diesen Giftstoff anfällig ist und sich auf demselben Gridfeld wie die produzierende Pflanze befindet. Das bedeutet, dass der Giftstoff s_x nicht mehr aktiv bleibt, wenn kein Cluster mit der Fressfeindart e_h existiert, das sowohl auf den Giftstoff reagiert als auch auf derselben Position wie die Pflanze liegt.

Zu diesem Zeitpunkt wird die Wirkung des Giftstoffs gleichzeitig mit seiner Verfüg-

barkeit eingestellt. Die Wirkung $w(s_x, t)$ des Giftstoffs s_x zum Zeitpunkt t lässt sich durch folgende Funktion beschreiben:

$$w(s_x, t) = \begin{cases} 0 & : t \geq t_f \\ 1 & : \text{sonst} \end{cases} \quad (66)$$

Dabei sei $t_f \in T$ der Zeitpunkt, an dem die Produktion von s_x endet und keine auslösenden Fressfeindcluster mehr vorhanden sind. Zu diesem Zeitpunkt muss der Normalzustand der Pflanze wiederhergestellt werden. Dies erfolgt, indem eine giftige Pflanze $p_{i,j}^*$ wieder in die ursprüngliche Darstellung $p_{i,j}$ umgewandelt wird. Hierfür wird eine Funktion $\mathcal{F}(p_{i,j}^*, s_x, t)$ eingeführt, die formal wie folgt beschrieben werden kann:

$$\mathcal{F}(p_{i,j}^*, s_x, t) = \begin{cases} 0 & : t \geq t_f \\ 1 & : \text{sonst} \end{cases} \quad (67)$$

Die Funktion $\mathcal{F}(p_{i,j}^*, s_x, t)$ gibt 0 zurück, wenn die Produktion des Giftstoffs s_x zum Zeitpunkt t endet. In diesem Fall wird die Pflanze von der giftigen Form $p_{i,j}^*$ in die ungiftige Form $p_{i,j}$ umgestellt, so dass sie zum Zeitpunkt $t+1$ auf dem Grid als solches sichtbar wird. Nachdem die Produktion des Giftstoffs s_x beendet ist und die Pflanze wieder frei ist, kann sie erneut zum Ziel für bereits vertriebene Fressfeindcluster werden:

$$\mathcal{Z}(C_i, p_{k,l}, t) = \begin{cases} 1 & : t \geq t_f \\ 0 & : \text{sonst} \end{cases} \quad (68)$$

Die Funktion $\mathcal{Z}(C_i, p_{k,l}, t)$ beschreibt, ob die Pflanze $p_{k,l}$ nach der Beendigung der Giftstoffproduktion wieder als Ziel für das Fressfeindcluster C_i verfügbar ist. Wenn für den Zeitpunkt $t \geq t_f$ gilt, wird die Pflanze wieder als Ziel für das Cluster zur Verfügung stehen, da der Zeitpunkt t_f , an dem die Verfügbarkeit des Giftstoffs entfernt wurde, überschritten ist.

3.7.13 Kosten für die Produktion einer Substanz

Eine Pflanze verfügt, wie bereits eingeführt, zu einem bestimmten Zeitpunkt t über ein Energieniveau $\mathcal{E}_{i,j}(t)$, das sie für ihr Überleben nutzt. Substanzen, die von einer

Pflanze produziert werden, können Signalstoffe oder Giftstoffe sein. Es ist möglich, dass beide Substanzarten gleichzeitig von derselben Pflanze produziert werden. Um eine doppelte Darstellung für die beiden Fälle zu vermeiden, wird im Folgenden die allgemeine Bezeichnung $c(s_i)$ verwendet, die die Energiekosten für eine beliebige Substanz $s_i \in S$ beschreibt.

Produziert die Pflanze $p_{i,j}$ zu einem Zeitpunkt t eine Substanz s_i , so reduziert sich ihr Energieniveau im darauffolgenden Zeitschritt $t + 1$ um den Wert $c(s_i)$.

Die Produktion von Substanzen ist nicht auf eine einzelne Substanz beschränkt; eine Pflanze kann gleichzeitig mehrere Substanzen $s_{i,1}, \dots, s_{i,l}$ herstellen. In einem solchen Fall summieren sich die jeweiligen Produktionskosten der Substanzen, wodurch die Gesamtkosten als Summe der Einzelkosten vom Energieniveau der Pflanze abgezogen werden. Die resultierende Änderung des Energieniveaus lässt sich wie folgt darstellen:

$$- \sum_{k=1}^l c(s_{i,k}) \quad (69)$$

wobei $c(s_{i,k})$ die Energiekosten für die Produktion der k -ten Substanz beschreibt.

3.8 Gesamtenergiebilanz der Pflanze

Zur Zusammenfassung der Prozesse, die Einfluss auf die Gesamtenergiebilanz einer Pflanze haben, werden im Folgenden die relevanten Komponenten dargestellt und erläutert, wie sie die Gesamtbilanz beeinflussen.

Wichtige Einflussgrößen, die die Gesamtenergie einer Pflanze bestimmen, umfassen zum einen die Energiezunahme pro Zeitschritt, wie in Formel 14 dargestellt. Darüber hinaus spielen auch die Kosten für die Produktion von Substanzen, die in Formel 69 spezifiziert sind, sowie die verzehrten Energieeinheiten durch Fressfeinde, wie in Formel 23 beschrieben, eine bedeutende Rolle und beeinflussen die Gesamtbilanz der Energie einer Pflanze.

Es sei nun $\mathcal{E}_{m,n}(t) \in \mathbb{R}^+$ die Gesamtenergie der Pflanze $p_{m,n}$ zum Zeitpunkt t , die sich folglich durch die nachstehende Gleichung auf den Zeitpunkt $t + 1$ aktualisieren lässt:

$$\mathcal{E}_{m,n}(t + 1) = \mathcal{E}_{m,n}(t) + \frac{r_{m,n}}{100} - \min\{\eta(C_j), \mathcal{E}_{m,n}(t)\} - \sum_{k=1}^l c(s_{i,k}) \quad (70)$$

In Worten ausgedrückt, beeinflussen alle drei Terme die Energie der Pflanze und haben maßgeblichen Einfluss auf die Gesamtenergiebilanz zum Zeitpunkt $t + 1$. Der erste Term führt zu einer Erhöhung der Energie, beispielsweise durch die Photosynthese, bei der Lichtenergie in für die Pflanze nutzbare chemische Energie in Form von Glukose umgewandelt und gespeichert wird [27]. Die beiden weiteren Terme hingegen verringern die Energie, und zwar durch den Verzehr von Energie sowie durch die Produktion von Substanzen. Daraus ergibt sich die tatsächliche Energie, die der Pflanze für den nächsten Zeitpunkt zu Verfügung steht.

3.9 Die Simulation und ihre Abbruchbedingungen

Für die Simulation kann die Zeitspanne eines einzelnen Schritts definiert werden. Hierzu wird die Zeitspanne durch ein $\Delta t \in \mathbb{R}^+$ beschrieben, welches die Dauer eines Zeitschritts eindeutig festlegt.

Es sollen verschiedene Bedingungen konfiguriert werden können, bei denen die Simulation beendet wird.

Die erste Abbruchbedingung basiert auf einer fest definierten Anzahl an Simulationsstufen. Mit anderen Worten existiert eine Teilmenge $T_{max} \subseteq T$, die wie folgt definiert ist: $T_{max} = \{t | t \in \mathbb{N}_0, t \leq t_{end}\}$, wobei t_{end} den vom Benutzer festgelegten Zeitschritt bezeichnet, an dem die Simulation terminiert.

Diese Abbruchbedingung kann durch die Unterscheidung der folgenden Zustände beschrieben werden:

$$Z_1(t_{end}) = \begin{cases} 1 & : t \geq t_{end} \\ 0 & : \text{sonst} \end{cases} \quad (71)$$

Die zweite Abbruchbedingung tritt ein, wenn eine bestimmte, frei wählbare Pflanzenart p_i ausgestorben ist. Eine Pflanzenart gilt als ausgestorben, wenn keine Exemplare der gewählten Art mehr auf dem Grid vorhanden sind bzw. die Summe der Energie der Pflanzen dieser Art gleich 0 ist.

Auch diese Bedingung kann formal durch eine Fallunterscheidung beschrieben werden:

$$Z_2(p_i) = \begin{cases} 1 & : \sum_{j=1}^N \mathcal{E}_{i,j}(t) = 0 \\ 0 & : \text{sonst} \end{cases} \quad (72)$$

Dabei bezeichnet N die Anzahl der Pflanzen, die der Art p_i zugehörig sind.

Die dritte Abbruchbedingung, welche das Ende der Simulation bestimmt, tritt ein, wenn keine einzige Pflanze mehr auf dem Grid vorhanden ist. Mit anderen Worten: Die Fressfeinde haben keine Beute mehr, sodass eine Fortsetzung der Simulation als nicht zielführend erachtet wird. Auch in diesem Fall wird über eine Fallunterscheidung geprüft, ob noch Pflanzen auf dem Grid existieren:

$$Z_3 = \begin{cases} 1 & : \sum_{i=1}^{16} \sum_{j=1}^N \mathcal{E}_{i,j}(t) = 0 \\ 0 & : \text{sonst} \end{cases} \quad (73)$$

Auch hier ist N die Anzahl der Pflanzen einer bestimmten Art.

Da Pflanzen in der Lage sind, tödliche Giftstoffe zu produzieren, kann es zu der Situation kommen, dass bestimmte Fressfeindarten aussterben. Daher wird eine weitere Abbruchbedingung definiert, welche das Ende der Simulation festlegt, wenn eine spezifische Fressfeindart e_h im Modell ausgestorben ist.

$$Z_4(e_h) = \begin{cases} 1 & : n(t) = 0 \\ 0 & : \text{sonst} \end{cases} \quad (74)$$

Dabei bezeichnet $n(t)$ die Anzahl der Individuen der Fressfeindart e_h , die sich zum Zeitpunkt t auf dem Grid befinden.

Die fünfte Abbruchbedingung ist äquivalent zur dritten Bedingung, mit dem Unterschied, dass hier nicht überprüft wird, ob alle Pflanzen ausgestorben sind, sondern ob alle Fressfeinde ausgestorben sind.

$$Z_5 = \begin{cases} 1 & : \sum_{i=1}^N n_i(t) = 0 \\ 0 & : \text{sonst} \end{cases} \quad (75)$$

Dabei beschreibt N die Anzahl der gesetzten Fressfeindcluster auf dem Grid. Das bedeutet, wenn die Summe aller Clustergrößen gleich 0 ist, existieren keine Fressfeinde mehr auf dem Grid.

Die vorletzte und sechste Abbruchbedingung überprüft, ob die Gesamtenergie der Pflanzen eine konfigurierbare Energiegrenze überschreitet. Überschreitet die Gesamt- energie diesen festgelegten Schwellwert, wird die Simulation beendet.

$$Z_6(\mathcal{E}_{limit}) = \begin{cases} 1 & : \sum_{i=1}^{16} \sum_{j=1}^N \mathcal{E}_{i,j}(t) \geq \mathcal{E}_{limit} \\ 0 & : \text{sonst} \end{cases} \quad (76)$$

Dabei beschreibt N erneut die Anzahl der Pflanzen einer bestimmten Art.

Die letzte und siebte Abbruchbedingung überprüft, ob die Gesamtanzahl der Individuen aller Fressfeindcluster eine konfigurierbare Grenze überschreitet. Wird diese festgelegte Grenze überschritten, wird die Simulation ebenfalls beendet.

$$Z_7(n_{limit}) = \begin{cases} 1 & : \sum_{i=1}^N n_i(t) \geq n_{limit} \\ 0 & : \text{sonst} \end{cases} \quad (77)$$

Dabei beschreibt N hier wieder die Anzahl der gesetzten Fressfeindcluster.

Die Simulation wird beendet, sobald mindestens eine der sieben Abbruchbedingungen erfüllt ist, das heißt, sobald sie den Wert 1 liefert. Dies kann formal wie folgt dargestellt werden:

$$Z = (Z_1 \vee Z_2 \vee Z_3 \vee Z_4 \vee Z_5 \vee Z_6 \vee Z_7) \quad (78)$$

In Worten: Sobald eine Abbruchbedingung erfüllt ist, gilt $Z = 1$, was das Ende der Simulation zur Folge hat.

4 Implementierung des Modells

Im folgenden Kapitel wird die Implementierung des Modells ausführlich erläutert. Die Programmiersprache Python in der Version 3.12.5 wurde für die Umsetzung des Modells gewählt. Angesichts der Komplexität des Modells und der Gefahr, den Überblick zu verlieren, wurde zur besseren Strukturierung und Übersichtlichkeit eine Modell-View-Controller (MVC) Architektur eingesetzt. Diese Architektur sorgt für eine klare Trennung der Modellkomponenten und unterteilt sie in die folgende Struktur:

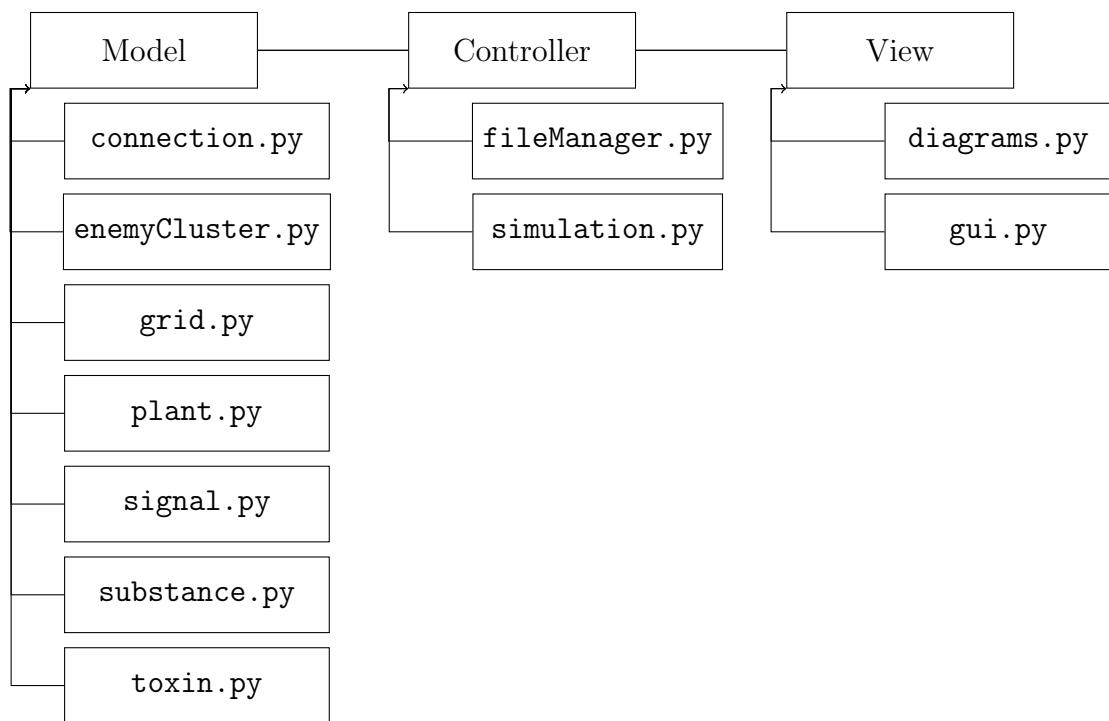


Abbildung 4: Skizze der Architektur des Modells

Der Inhalt der einzelnen Dateien wird im folgenden Kapitel ausführlich erläutert. Zur Verwaltung der Abhängigkeiten und zur Erstellung einer isolierten Entwicklungsumgebung wurde das Werkzeug Poetry [12] verwendet. Dieses ermöglicht die einfache Verwaltung der benötigten Pakete, Abhängigkeiten und sorgt für die Reproduzierbarkeit der Arbeitsumgebung.

Für die Berechnungen innerhalb des Modells kam die Bibliothek NumPy [9] zum Einsatz. Zur Visualisierung des Ablaufs einer Simulation wurde Matplotlib [19] für die Erstellung von Diagrammen verwendet.

4.1 Die Pflanze

In der Klasse der Pflanzen werden verschiedene Aspekte berücksichtigt, um ein möglichst realistisches Verhalten der auf dem Grid platzierten Pflanzen zu modellieren. Zu diesem Zweck verfügen die Objekte über essenzielle Attribute, die eine realitätsnahe Simulation ermöglichen.

Ein Pflanzenobjekt besitzt die folgenden Attribute:

Attribute der Klasse Plant					
#	Attribut	Typ	#	Attribut	Typ
1	<code>name</code>	String	2	<code>initEnergy</code>	int
3	<code>currEnergy</code>	int	4	<code>growthRateEnergy</code>	int
5	<code>minEnergy</code>	int	6	<code>reproductionInterval</code>	int
7	<code>offspringEnergy</code>	int	8	<code>minDist</code>	int
9	<code>maxDist</code>	int	10	<code>position</code>	(int, int)
11	<code>grid</code>	Grid	12	<code>color</code>	Liste

Tabelle 3: Attribute eines Pflanzen-Objekts mit Datentypen

Auf dem Grid können verschiedene Pflanzenarten existieren. Um diese zu unterscheiden, besitzt jede Pflanze das Attribut `name`, das sie einer spezifischen Art zuordnet. Entsprechend der Menge P muss mindestens eine Pflanzenart vorhanden sein, wobei maximal 16 verschiedene Arten existieren können. Da die konkreten Namen der Pflanzenarten für den weiteren Verlauf nicht relevant sind und eine direkte Zuordnung durch den Nutzer nicht erforderlich ist, werden die Arten lediglich durch die Bezeichnungen '`p1`', '`p2`', ..., '`p16`' differenziert. Dadurch wird sichergestellt, dass jedes Pflanzenobjekt eindeutig einer bestimmten Art zugewiesen ist.

Wird eine Pflanze manuell auf dem Grid platziert, befindet sich das Simulationssystem im Zeitpunkt $t = 0$. Zu diesem Zeitpunkt verfügt die platzierte Pflanze über eine initiale Energiemenge, die durch das Attribut `initEnergy` beschrieben und vom Nutzer festgelegt wird.

Wie bereits im Kapitel zur formalen Modellbeschreibung erläutert, verfügt eine Pflanze zu jedem Zeitpunkt über eine bestimmte Energiemenge. Diese wird durch das Attribut `currEnergy` repräsentiert.

Solange eine Pflanze keinem besonderen Energieverbrauch unterliegt - also weder

Substanzen produziert noch von einem Feind angegriffen wird - nimmt ihre verfügbare Energie im Zeitverlauf zu. Das Wachstum erfolgt mit einer vom Nutzer definierten Wachstumsrate, die durch das Attribut `growthRateEnergy` festgelegt wird. Da das Überleben in der Realität ohne eine Mindestmenge an Energie, die lebenswichtige Prozesse aufrechterhält, nicht möglich ist, wird in Übereinstimmung mit der formalen Beschreibung das Attribut `minEnergy` verwendet. Dieses definiert die minimale Energiemenge, die eine Pflanze zum Überleben benötigt.

Im folgenden werden zwei Funktionen vorgestellt, die das Zusammenspiel von initia-ler Energie, aktueller Energie und Wachstumsrate modelliert.

Die erste Funktion `grow()` beschreibt das Wachstumsverhalten einer Pflanze im Zeitverlauf.

```
1 def grow(self):
2     self.currEnergy += self.initEnergy * (self.growthRateEnergy / 100)
3     self.age += 1
```

Quelltext 1: Wachstumsfunktion einer Pflanze

Die Methode `grow()` aktualisiert die aktuelle Energie der Pflanze basierend auf der initialen Energie und der Wachstumsrate. In jedem Zeitschritt wird die aktuelle Energie um einen bestimmten Prozentsatz der initialen Energie erhöht. Dieser Prozentsatz wird durch die Wachstumsrate festgelegt, sodass die Pflanze in jedem Zeitschritt entsprechend der Rate an Energie zunimmt. Die zweite Funktion, `survive()`, überprüft, ob eine Pflanze die Mindestenergieanforderung erfüllt. Sie stellt sicher, dass nur Pflanzen auf dem Grid existieren, deren aktuelle Energie den durch das Attribut `minEnergy` definierten Schwellwert nicht unterschreiten.

```
1 def survive(self):
2     if self.currEnergy < self.minEnergy:
3         self.grid.removePlant(self)
```

Quelltext 2: Überprüfung der Mindestenergie einer Pflanze

In dieser Funktion wird die aktuell verfügbare Energie überprüft, um festzustellen, ob sie unter der definierten Mindestenergie liegt. Ist die aktuelle Energie geringer als der Schwellwert, verfügt die Pflanze nicht über ausreichend Energie und vergeht

bzw. wird vom Grid entfernt. Liegt die aktuelle Energie hingegen über der Mindestenergie, bleibt die Pflanze weiterhin auf dem Grid bestehen.

In Übereinstimmung mit der formalen Beschreibung soll sich eine Pflanze in regelmäßigen Intervallen reproduzieren und eine zufällige Anzahl von Pflanzen der gleichen Art hervorbringen. Der Zeitraum, in dem die Reproduktion stattfindet, wird durch das Attribut `reproductionInterval` für jedes Pflanzenobjekt festgelegt, das genau dieses Intervall definiert.

Basierend auf dem Reproduktionsintervall entstehen durch die Reproduktion neue Pflanzenobjekte, die ebenfalls eine initiale Energie benötigen. Da die Vergabe dieser Energie nicht manuell erfolgt, sondern automatisch durch die Mutterpflanze, wird die Energie der Nachkommen im Attribut `offspringEnergy` festgehalten.

Ein Nachkomme einer Pflanze kann nicht auf jedem beliebigen Gridfeld platziert werden, sondern nur innerhalb eines bestimmten Bereichs um die Mutterpflanze. Dieser Bereich wird durch die `minDist` und `maxDist` definiert, innerhalb dessen die Nachkommen der Pflanze platziert werden können.

Auch in diesem Fall verfügt die Klasse über zwei relevante Funktionen, die es einer Pflanze ermöglichen, ihre Nachkommen innerhalb der festgelegten Radiusgrenzen auf dem Grid zu platzieren. Zur Verteilung der Samen wird die Funktion `scatterSeed()` verwendet, die eine zufällige Anzahl von Pflanzenobjekten der gleichen Art verstreuht und auf dem Grid platziert, sobald das Reproduktionsintervall durchlaufen ist.

```

1 def scatterSeed(self):
2     if self.reproductionInterval == 0:
3         pass
4     elif self.age % self.reproductionInterval == 0:
5         for _ in range(random.randint(1, 4)):
6             offspringPosition = self.setOffspringPos()
7             if offspringPosition:
8                 offspring = Plant(...,
9                                 position = offspringPosition,
10                                ...)
11             self.grid.addPlant(offspring)

```

Quelltext 3: Erzeugung von Nachkommen einer sich reproduzierenden Pflanze

Zur Anmerkung: Alle Attribute der Nachkommen sind identisch mit denen der Mutterpflanze, mit Ausnahme der Attribute `position` und `currEnergy`. Das Attribut `position` beschreibt die Position der jeweiligen Pflanze auf dem Grid und besteht aus zwei Ganzzahlen im Bereich von 1 bis 80, entsprechend der Menge G für das Grid.

Bei der Initialisierung des Pflanzenobjekts wird das Attribut `currEnergy` auf den Wert des Parameters `offspringEnergy` gesetzt, wodurch der anfängliche Energie-level der Nachkommen bestimmt wird.

Die Position der Nachkommen wird durch eine zusätzliche Funktion festgelegt, die auch die minimale und maximale Distanz zur Mutterpflanze berücksichtigt.

Die Funktion, die für die Bestimmung der Position der Nachkommen verantwortlich ist, trägt den Namen `setOffspringPos()` und ist wie folgt aufgebaut:

```

1 def setOffspringPos(self):
2     directions = self.getDirections()
3     random.shuffle(directions)
4     for dx, dy in directions:
5         newX, newY = self.position[0] + dx, self.position[1] + dy
6         if self.grid.isWithinBounds(newX, newY):
7             if not self.grid.isOccupied((newX, newY)):
8                 return (newX, newY)
9             else:
10                 pass
11         else:
12             pass
13     return None

```

Quelltext 4: Positionsbestimmung der pflanzlichen Nachkommen

Die Methode sucht nach einer freien und gültigen Position innerhalb des Grids, die den festgelegten Radius berücksichtigt, um einen Nachkommen zu platzieren. Sie überprüft alle möglichen Ausbreitungsrichtungen der Mutterpflanze und gibt die erste gültige Position zurück, die sie findet. Wird keine geeignete Position gefunden, so wird `None` zurückgegeben. Dies bedeutet, dass der betrachtete Nachkomme nicht erzeugt wird, da keine freie Fläche verfügbar ist, auf der er platziert werden könnte.

In dieser Modellierung könnte man damit argumentieren, dass ein Nachkomme, der nicht platziert werden kann, aufgrund der Konkurrenz mit anderen Pflanzen keinen geeigneten Platz findet. Pflanzen, die zu diesem Zeitpunkt stärker oder dominanter sind, können Raum für die Ausbreitung des Nachkommens blockieren. Das Fehlen eines Nachkommens in dieser Funktion lässt sich somit als eine Form des natürlichen Wettbewerbs verstehen, bei dem Pflanzen nicht immer die Möglichkeit haben, sich in einer bereits dicht besiedelten Umgebung durchzusetzen.

Die letzten beiden Eigenschaften beziehen sich auf das Attribut `grid`, das ein Objekt der Klasse Grid ist und in einem späteren Kapitel behandelt wird, sowie auf `color`, eine Liste von 16 Strings. Jeder dieser Strings repräsentiert einen Farbcde, der einer bestimmten Pflanzenart zugeordnet ist. Mit anderen Worten, jede Pflanzenart hat eine zugewiesene Farbe, die in der graphischen Oberfläche des Grids sichtbar wird.

4.2 Die Feinde und ihre Fressfeindcluster

Um ein Objekt eines Fressfeindclusters zu erzeugen, wurde zuvor eine Klasse für die Fressfeindarten mit dem Namen `Enemy` definiert, die ihre Attribute an die Klasse `EnemyCluster` vererbt.

Das Attribut der Klasse für die Fressfeindarten ist `name`, welches ähnlich wie bei den Pflanzen zur Unterscheidung der verschiedenen Feindarten dient.

Die Klasse `EnemyCluster` weist die folgenden Attribute auf:

Attribute der Klasse EnemyCluster					
#	Attribut	Typ	#	Attribut	Typ
1	<code>enemy</code>	<code>Enemy</code>	2	<code>num</code>	<code>int</code>
3	<code>speed</code>	<code>int</code>	4	<code>position</code>	<code>(int, int)</code>
5	<code>grid</code>	<code>Grid</code>	6	<code>eatVictory</code>	<code>int</code>
7	<code>eatingSpeed</code>	<code>int</code>			

Tabelle 4: Attribute eines Fressfeindcluster-Objekts mit Datentypen

Das Attribut `enemy` ist ein Objekt der `Enemy`-Klasse. Wie bereits erwähnt, wird auf diese Weise der Name sowie die Art der Fressfeinde an die Klasse `EnemyCluster` weitervererbt und nutzbar gemacht. Diese Vorgehensweise stellt sicher, dass ein `EnemyCluster`-Objekt immer nur Fressfeinde einer einzigen Art enthält, wodurch

eine Vermischung verschiedener Feindarten innerhalb eines Clusters vermieden wird. Das Attribut `num` definiert die Größe des Fressfeindclusters, also die Anzahl der Individuen der jeweiligen Fressfeindart, die innerhalb dieses Clusters vorhanden ist, entsprechend der formalen Definition C_i .

Das Attribut `speed` definiert, wie schnell sich das erzeugte Objekt des Fressfeindclusters von einem Gridfeld zum nächsten bewegt. Dieses Attribut repräsentiert somit das Zeitintervall Δt_i aus Kapitel 3.4.2, welches durchlaufen werden muss, um zu bestimmen, wann sich ein Fressfeindcluster bewegen darf.

Weiterhin wird auch ein Objekt des Fressfeindclusters, ähnlich wie ein Pflanzenobjekt, manuell vom Nutzer vor Beginn der Simulation auf dem Grid platziert. Aus diesem Grund besitzt es ebenfalls das Attribut `position`, welches als Tupel von zwei Ganzzahlen die Position auf dem Grid repräsentiert. Da der Nutzer die Fressfeindcluster auf dem Grid platziert, spielt auch hier das Grid eine Rolle. Weitere Details zu diesem Attribut werden im entsprechenden Kapitel zur Grid-Klasse behandelt.

Das sechste Attribut in dieser Klasse ist `eatVictory`, welches die Menge an Energieeinheiten definiert, die benötigt werden, um ein neues Individuum der entsprechenden Fressfeindart hervorzu bringen und damit das Cluster zu vergrößern.

Das letzte Attribut dieser Klasse ist `eatingSpeed`, welches, wie der Name bereits vermuten lässt, für die Essgeschwindigkeit des Fressfeindclusters zuständig ist. Dieses Attribut definiert in Form einer natürlichen Zahl, wie viele Energieeinheiten das Cluster einer Pflanze pro Zeitschritt verzehrt.

Ein Fressfeindcluster bewegt sich, wie im Kapitel 3.4 schon formal definiert und beschrieben. Zur Veranschaulichung wird im Folgenden die Funktion dargestellt, die beschreibt, wie ein Fressfeindcluster seine Zielpflanze auswählt.

```

1 def chooseRandomPlant(self, start):
2     if self.targetPlant is not None:
3         current_plants = self.detectPlant(self.grid.helperGrid())
4         if self.targetPlant not in current_plants:
5             self.targetPlant = None
6         else:
7             return self.findShortestPath(start, self.targetPlant)

```

```

8     helperGrid = self.grid.helperGrid()
9
10    pPos = self.detectPlant(helperGrid)
11
12    shortestPaths = []
13
14    spLength = None #shortestPathLength
15
16    if len(pPos) == 0:
17
18        print('no plant. stop simulation')
19
20        return []
21
22    for pos in pPos:
23
24        path = self.findShortestPath(start, pos)
25
26        if path is not None:
27
28            if spLength is None or len(path) < spLength:
29
30                spLength = len(path)
31
32                shortestPaths = [path]
33
34            elif len(path) == spLength:
35
36                shortestPaths.append(path)
37
38    if len(shortestPaths) > 0:
39
40        chosenPath = random.choice(shortestPaths)
41
42        self.targetPlant = chosenPath[-1]
43
44        return chosenPath
45
46    else:
47
48        return None

```

Quelltext 5: Auswahl einer zufälligen nächstgelegenen Pflanze

Die Funktion `chooseRandomPlant(start)` nimmt einen Parameter entgegen, der unter anderem als Startpunkt für die Suche nach der nächsten Pflanze in der aufgerufenen Funktion `findShortestPath` dient. Mit anderen Worten, in `findShortestPath` wird eine Breitensuche durchgeführt, um den kürzesten Pfad zu finden. Dieser Pfad wird dann an die Funktion `chooseRandomPlant(start)` übergeben, wo die Länge des gefundenen kürzesten Pfads zunächst in der Variable `spLength` gespeichert wird.

Es kann vorkommen, dass ein noch kürzerer Weg gefunden wird. In diesem Fall wird der bisher kürzeste Pfad durch den neuen kürzesten Pfad ersetzt. Dieser Vorgang wiederholt sich, bis kein kürzester Pfad mehr gefunden wird. In der Situation, dass mehrere kürzeste Pfade existieren, werden alle gefundenen Pfade in einer Liste gespeichert. Aus dieser Liste wird zufällig ein Pfad ausgewählt, wobei die

Zielpflanze am Ende des ausgewählten Pfads steht.

Es gibt eine Funktion `move(path)`, die den gefundenen bzw. ausgewählten Pfad als Parameter entgegennimmt und das Fressfeindcluster entlang dieses Pfades zur Pflanze bewegt. Dabei wird über den Pfad iteriert und das Fressfeindcluster schrittweise entlang der Position im Pfad verschoben. Die Funktion selbst wird jedoch nicht in der `EnemyCluster`-Klasse verwendet, sondern in der später besprochenen `Grid`-Klasse.

Jedes Mal, wenn ein Cluster einen Schritt im Pfad gemacht hat, wird mit der Funktion `checkAndUpdatePath(currentPos)` überprüft, ob sich eine andere Pflanze näher zur aktuellen Position befindet. Falls dies zutrifft, wird der aktuelle Pfad durch den neu gefundenen Pfad ersetzt, sodass sich das Fressfeindcluster in Richtung der neu identifizierten Pflanze bewegt. Es sei darauf hingewiesen, dass ein Objekt des Fressfeindclusters ein internes Attribut besitzt, das nicht vom Nutzer definiert wird und dem Cluster sozusagen ein „Gedächtnis“ verleiht. Dieses Attribut wird jedes Mal auf die gefundene Pflanze gesetzt, sobald das Fressfeindcluster die Position der aktuell anvisierten Pflanze erreicht hat. Die Suche nach einer neuen Zielpflanze erfolgt unter der Voraussetzung, dass die neue Zielpflanze nicht mit dem Pflanzenobjekt identisch ist, dass in `lastPlant` gespeichert wurde. Mit diesem Mechanismus wird sichergestellt, dass sich ein Fressfeindcluster nicht ständig zwischen dem Feld der Pflanze und einem benachbarten Gridfeld bewegt, bis die Pflanze nicht mehr genügend Energie besitzt und stirbt.

Immer wenn ein Fressfeindcluster an einer Pflanze ankommt, befindet es sich in der Situation, sich von der Pflanze zu ernähren und Energie für die Nachkommen zu sammeln. In diesem Fall wird in der `Grid`-Klasse die Funktion `eatPlant(ec, plant)` aufgerufen. Diese Funktion prüft, ob sich das Fressfeindcluster auf einer Position mit einer Pflanze befindet. Falls dies zutrifft, wird die Energie der Pflanze in jedem Zeitschritt, in dem das Fressfeindcluster auf dem Gridfeld verweilt, um den Wert von `eatingSpeed` reduziert. Gleichzeitig erhöht sich der Energiespeicher des Fressfeindclusters um diesen Wert. Sollte die Pflanze aufgrund von zu wenig Energie versterben, wird sie vom Grid entfernt. Gleichzeitig erfolgt eine Benachrichtigung an andere Fressfeindcluster, dass diese Pflanze nicht mehr existiert und sie ein

neues Ziel suchen müssen.

Sobald ein Fressfeindcluster genügend Energie aufgenommen hat, erzeugt es neue Nachkommen. Dies tritt ein, wenn die verzehrte Energie den vom Nutzer festgelegten Wert in `eatVictory` überschreitet. Zur Modellierung der Reproduktion eines Fressfeindclusters wurde die Funktion `reproduce()` implementiert.

```

1 def reproduce(self):
2     if self.eatedEnergy >= self.eatVictory:
3         self.newBorns += int(self.eatedEnergy / self.eatVictory)
4         self.num += self.newBorns
5         self.eatedEnergy -= self.newBorns * self.eatVictory
6         self.newBorns = 0

```

Quelltext 6: Reproduktion von Fressfeinden

Sobald die verzehrte Energie das festgelegte Ziel überschreitet, wird die Anzahl der Individuen ermittelt, die mit der aufgenommenen Energie erzeugt werden können. Dies ist notwendig, da die Werte für `eatVictory` und `eatingSpeed` so gewählt werden können, dass pro Zeitschritt mehr Energie verzehrt wird, als für die Erzeugung eines einzelnen Nachkommens benötigt wird. Aus diesem Grund wird zunächst die Anzahl der Nachkommen berechnet, die daraufhin zur Clustergröße `num` addiert werden. Anschließend muss die verwendete Energie von dem Energiespeicher `eatedEnergy` subtrahiert werden, um eine Doppelverwendung der Energie zu verhindern.

An diesem Punkt wurde die Funktionsweise der Klasse für die Fressfeindcluster erläutert. Die Klasse umfasst jedoch weitere Funktionen, deren detaillierte Behandlung im Kapitel zur `Grid`-Klasse angebracht ist. Dies liegt daran, dass an dieser Stelle noch Informationen fehlen, die in den folgenden Kapiteln behandelt werden.

4.3 Die Substanzen

Für die Modellierung von Substanzen sind an dieser Stelle drei Klassen von Bedeutung: Die Klasse `Substance` dient als Oberklasse und vererbt ihre Attribute an die beiden Unterklassen `Signal` und `Toxin`.

Die Klasse **Substance** verfügt über zwei Attribute, die den Namen sowie den Typ der jeweiligen Substanz definiert.

Attribute der Klasse Substance					
#	Attribut	Typ	#	Attribut	Typ
1	name	String	2	type	String

Tabelle 5: Attribute eines **Substanz**-Objekts mit Datentypen

Hier wird der Name nicht mehr vom System vorgegeben, sondern kann vom Nutzer frei bestimmt werden, unter dem die Substanz in der Simulation agiert. Der Typ einer Substanz hängt davon ab, an welche UnterkLASSE die Attribute vererbt werden. Mit anderen Worten: Handelt es sich um einen Signalstoff, sollte `type = 'signal'` gesetzt werden, während bei einem Giftstoff `type = 'toxin'` verwendet wird.

4.3.1 Die Klasse der Signalstoffe

Die Klasse **Signal** besitzt, ähnlich wie die zuvor beschriebenen Klassen, mehrere Attribute. Zum einen erbt sie die beiden Attribute der **Substance**-Klasse, zum anderen verfügt sie über signalspezifische Attribute, die gemäß der folgenden Tabelle definiert sind:

Attribute der Klasse Signal					
#	Attribut	Typ	#	Attribut	Typ
1	substance	Substance	2	emit	Liste
3	receiver	Liste	4	triggerCombination	Liste
5	prodTime	int	6	spreadType	String
7	sendingSpeed	int	8	energyCosts	int
9	afterEffectTime	int			

Tabelle 6: Attribute eines **Signal**-Objekts mit Datentypen

In der Liste `emit` werden alle Pflanzenarten in Form von Strings, die die jeweilige Pflanzenart repräsentieren (z.B. `['p1', 'p4', ...]`) gespeichert. Diese Pflanzenarten sind in der Lage, unter den in der Liste `triggerCombination` definierten Bedingungen einen Signalstoff zu produzieren. Ebenso enthält die Liste `receiver`

alle Pflanzenarten, die in der Lage sind, einen Signalstoff von anderen Pflanzenobjekten zu empfangen.

Die Definition der Pflanzenarten in `receiver` erfolgt analog zu `emit`.

Das Attribut `triggerCombination` ist, wie bereits erwähnt, eine Liste von Bedingungen, unter denen die Produktion eines Signalstoffs ausgelöst wird. Eine solche Bedingung legt fest, welche Fressfeindart und welche Mindestanzahl an Individuen auf dem Feld der Pflanze vorhanden sein müssen, damit die Signalstoffproduktion startet.

Die `triggerCombination`-Liste besteht aus mehreren Unterlisten, die jeweils zwei Elemente enthalten. Das erste Element ist der Name einer Fressfeindart als String, während das zweite Element die Mindestgröße des entsprechenden Fressfeindclusters als ganze Zahl angibt. Die Struktur der `triggerCombination`-Liste kann beispielsweise wie folgt aussehen: `[['e1', 4], ['e3', 5], ...]`. Die konkrete Ausgestaltung dieser Liste hängt von der individuellen Definition des Nutzers ab.

Die Attribute `prodTime`, `sendingSpeed`, `energyCosts` und `afterEffectTime` werden jeweils als ganze Zahl vom Nutzer festgelegt und erfüllen folgende Funktionen:

- `prodTime`: Gibt an, wie viele Zeitschritte benötigt werden, bis der produzierte Signalstoff verfügbar ist.
- `sendingSpeed`: Bestimmt die Geschwindigkeit, mit der sich der Signalstoff zu einer empfangenden Pflanze aus der `receiver`-Liste ausbreitet.
- `energyCosts`: Definiert die Energiemenge, die für die Produktion des Signalstoffs erforderlich ist.
- `afterEffectTime`: Beschreibt die Nachwirkzeit des Signalstoffs, also die Anzahl der Zeitschritte, die er nach Abschluss der Produktion noch an der Pflanze verbleibt, wenn die Pflanze durch keinen Triggerfeind belagert wird.

Das Attribut `spreadType` legt fest, auf welche Weise ein Signalstoff an empfangende Pflanzen weitergeleitet wird. Es wird als String definiert, wobei eine Luftverbreitung durch `'air'` und eine Übertragung über symbiotische Verbindungen durch `'symbiotic'` gekennzeichnet wird.

Die Methoden in einem **Signal**-Objekt umfassen drei verschiedene Funktionen. Die erste Funktion reduziert in jedem Zeitschritt, in dem der Signalstoff produziert bzw. aktiv ist, die aktuell verfügbare Energie der produzierenden Pflanze um die festgelegten Energiekosten.

Die zweite und dritte Funktion sind zwei Hilfsfunktionen, die innerhalb der Grid-Klasse aufgerufen werden. Die zweite Funktion aktiviert das Signal, sobald es fertig produziert wurde, während die dritte Funktion das Signal deaktiviert, sobald die Nachwirkzeit abgelaufen ist.

4.3.2 Die Klasse der Giftstoffe

Die Klasse **Toxin** für einen Giftstoff ist ähnlich aufgebaut wie die Klasse für einen Signalstoff, weist jedoch wesentliche Unterschiede auf. Ein Objekt der Klasse **Toxin** besitzt folgende Attribute, die der nachfolgenden Tabelle entnommen werden können.

Attribute der Klasse Toxin					
#	Attribut	Typ	#	Attribut	Typ
1	substance	Substance	2	plantTransmitter	Liste
3	energyCosts	int	4	triggerCombination	Liste
5	prodTime	int	6	deadly	bool
7	eliminationStrength	int			

Tabelle 7: Attribute eines **Toxin**-Objekts mit Datentypen

Die Attribute **substance**, **energyCosts** und **prodTime** sind identisch mit den gleichnamigen Attributen aus der Klasse **Signal**, mit der Ausnahme von **substance**, bei dem das Attribut **type = 'toxin'** gesetzt ist. Daher wird auf eine erneute Erläuterung dieser Attribute verzichtet.

Das Attribut **plantTransmitter** ist identisch wie das **emit**-Attribut aufgebaut und enthält die Pflanzenarten, die diesen Giftstoff produzieren können. Diese Pflanzenarten werden als Strings gespeichert.

Das Attribut **triggerCombination** enthält eine Liste von Kombinationen, die die Produktion des Giftstoffs auslösen. Jede Kombination besteht aus drei Elementen: Das erste Element ist der Name eines Signalstoffs, da ein Giftstoff nur dann produ-

ziert wird, wenn der entsprechende Signalstoff aktiv ist. Die weiteren Elemente sind wie bei den Signalstoffen die Fressfeindart als String und die Mindestgröße des Fressfeindclusters, die notwendig ist, um die Produktion des Giftstoffs zu aktivieren. Die `triggerCombination`-Liste in der `Toxin`-Klasse könnte daher folgende Form haben:

```
[[‘s1’, ‘e1’, 4], [‘s2’, ‘e3’, 5], ...].
```

Das Attribut `deadly` ist ein Wahrheitswert, der angibt, ob der jeweilige Giftstoff eine tödliche oder vertreibende Wirkung hat. Wenn `deadly = False` gesetzt ist, handelt es sich um einen vertreibenden Giftstoff, der Fressfeindcluster vertreibt, die auf diesen Giftstoff reagieren. In diesem Fall werden nur diejenigen Feindarten beeinflusst, die auf den Giftstoff ansprechen, während andere Feinde unbeeinträchtigt bleiben und ihr Vorhaben fortsetzen. Wenn jedoch `deadly = True` ist, wird der Giftstoff für die Fressfeinde, die darauf reagieren, tödlich. In diesem Fall kommt das Attribut `eliminationStrength` zum Tragen, um die Stärke der tödlichen Wirkung des Giftstoffs zu bestimmen.

Das Attribut `eliminationStrength` beschreibt die Intensität eines tödlichen Giftstoffs und gibt an, wie viele Individuen eines betroffenen Fressfeindclusters pro Zeitschritt durch die Vergiftung eliminiert werden.

Ähnlich wie die Klasse `Signal` verfügt auch die `Toxin`-Klasse über eine Funktion, die die Energiekosten eines Giftstoffs pro Zeitschritt reduziert, wenn dieser produziert wird.

Darüber hinaus enthält die Klasse eine Funktion `displaceEnemies(ec, plant)`, die bei einer vertreibenden Wirkung eines Giftstoffs aufgerufen wird. Sie nimmt zwei Parameter entgegen: das Fressfeindcluster (`ec`) und eine Pflanze (`plant`). Die Funktion bestimmt einen neuen Pfad für das Fressfeindcluster zu einer anderen Pflanze, die nicht diejenige ist, von der es gerade verjagt wird. Der Ablauf dieser Funktion ist wie folgt:

```
1 def displaceEnemies(self, ec, plant):
2     np = []
3     tp = None
4     if not ec.currentPath:
5         np = ec.newPathAfterDisplace(plant, self)
6     if np:
```

```

7         tp = np[-1]
8     else:
9         pass
10    return np, tp

```

Quelltext 7: Funktion zum Vertreiben von Fressfeinden

Die Funktion liefert zum einen den neu ermittelten Pfad und die Position der neuen Zielpflanze, die sich an der letzten Stelle des Pfades befindet und durch `np[-1]` bestimmt wird.

Da, wie bereits erwähnt, ein Giftstoff eine tödliche Wirkung haben kann, gibt es auch eine Funktion, die das Fressfeindcluster interne Attribut `intoxicated = True` setzt, wenn das Fressfeindcluster durch ein tödliches Gift vergiftet wurde.

Die letzte Funktion, `killEnemies(ec)`, wird verwendet, um die Anzahl der Individuen eines Fressfeindclusters, das mit einem tödlichen Giftstoff vergiftet wurde, entsprechend des `eliminationStrength`-Attributs in jedem Zeitschritt zu reduzieren.

Diese Funktion wird jedoch in der Klasse zum Grid verwendet, sodass ihre Auswirkungen besser im nun folgenden Kapitel zur `Grid`-Klasse demonstriert werden können.

4.4 Die Gridverbindung zwischen zwei Pflanzen

Die Klasse `SymbioticConnection` steuert und verwaltet die Gridverbindungen und somit die gezielte Ausbreitung von Signalstoffen. Sie verfügt über zwei zentrale Attribute, die in der folgenden Tabelle dargestellt sind:

Attribute der Klasse SymbioticConnection					
#	Attribut	Typ	#	Attribut	Typ
1	plant1	Plant	2	plant2	Plant

Tabelle 8: Attribute eines `SymbioticConnection`-Objekts mit Datentypen

Die Attribute repräsentieren die Objekte der beiden benachbarten Pflanzen, die durch eine Gridverbindung symbiotisch miteinander verknüpft werden.

Dabei bezeichnet `plant1` die sendende Pflanze, während `plant2` die Pflanze ist, die

den Signalstoff empfängt.

Innerhalb dieser Klasse existieren zwei Funktionen. Die erste Funktion, `getDistance(pos1, pos2)`, dient als Hilfsfunktion. Sie erhält die Positionen der jeweiligen Pflanzen als Parameter und berechnet den Abstand zwischen ihnen anhand der Manhattan-Distanz.

Die zweite Funktion, `createConnection()`, stellt die eigentliche Gridverbindung zwischen den beiden Pflanzen her. Zunächst überprüft sie, ob die Distanz zwischen den Pflanzen genau 1 beträgt, da sie nur in diesem Fall als benachbart gelten und eine Gridverbindung hergestellt werden kann. Ist diese Bedingung erfüllt, wird die Verbindung bidirektional in einem Dictionary gespeichert, das als internes Attribut der `Grid`-Klasse definiert ist.

Dieses Dictionary, in dem die Pflanzen als Schlüssel verwendet werden, dient in der `Grid`-Klasse als zentrales Element für den Zugriff auf die Gridverbindungen. Es wird in den relevanten Funktionen zur Kommunikation zwischen den Pflanzen genutzt. Eine detaillierte Erläuterung dazu folgt im Kapitel zur `Grid`-Klasse.

4.5 Das Grid und die Simulation

In diesem Kapitel werden die Klassen rund um `Grid` und `Simulation` hinsichtlich ihres Aufbaus und ihrer Funktionsweise erläutert. Darüber hinaus wird demonstriert, wie ein Grid im Hintergrund des Tools strukturiert ist und dargestellt wird.

Da die Klasse `Grid` über eine Vielzahl von Funktionen verfügt, werden im Folgenden zentrale Schlüsselfunktionen sowie deren Funktionsweise erläutert. Zudem wird ihr Zusammenspiel mit der Simulation und den zugehörigen Klassen erläutert.

Attribute der Klasse Grid					
#	Attribut	Typ	#	Attribut	Typ
1	height	int	2	width	int
Interne Attribute der Klasse Grid					
3	plants	Liste	4	enemies	Liste
5	signals	Liste	6	toxins	Liste
7	grid	2D Liste			

Tabelle 9: Attribute und interne Attribute eines `Grid`-Objekts mit Datentypen

4.5.1 Initialisierung des Grids

Ein Objekt der Klasse `Grid` verfügt über zwei vom Nutzer definierte Attribute sowie mehrere interne Attribute, die bei der Erstellung des Objekts automatisch generiert werden und für die Funktionsweise der Klasse relevant sind.

Die Attribute `height` und `width` legen die gewünschte Größe des Grids fest, das zur Simulation eines Biotops verwendet wird. Diese Werte werden vom Nutzer als natürliche Zahlen vorgegeben.

Mit der Erstellung eines `Grid`-Objekts werden zusätzlich mehrere interne Attribute initialisiert, die in der zweiten Hälfte von Tabelle 9 aufgeführt sind. Konkret werden für die Biotopelemente separate leere Listen angelegt, in denen die auf dem Grid platzierten Objekte – darunter Pflanzen, Fressfeindcluster, Signalstoffe und Giftstoffe – gespeichert werden. Diese Listen ermöglichen während der Simulation einen effizienten Zugriff auf die entsprechenden Objekte.

Das interne Attribut `grid` ist eine zweidimensionale Liste, die zum Zeitpunkt der Initialisierung die folgende Struktur aufweist:

(None, [])	(None, [])	(None, [])	...
(None, [])	(None, [])	(None, [])	...
(None, [])	(None, [])	(None, [])	...
:	:	:	..

Abbildung 5: Initiale Struktur der 2D-Liste `grid`

Die 2D-Liste, die das Grid repräsentiert, kann als eine Matrix betrachtet werden, in der jede Zelle einer Positionskombination aus der bereits definierten Menge G entspricht.

Genauso wie in der formalen Definition besteht eine Position aus einem Tupel, bei dem die erste Komponente für eine potenziell platzierte Pflanze reserviert ist. Die zweite Komponente ist eine Liste, die die auf dem Gridfeld befindlichen Fressfeindcluster zu einem bestimmten Zeitpunkt speichert und verwaltet.

Diese Struktur bildet die formale Definition des dynamischen Grids sowie die Funktion l zur Belegung eines Gridfelds ab.

4.5.2 Hinzufügen und Entfernen von Klassen-Objekten

Um Pflanzen, ihre schützenden Substanzen sowie die Fressfeindcluster auf dem Grid zu platzieren, wurden für jede Objektart entsprechende Funktionen implementiert, die alle nach dem gleichen Prinzip arbeiten. Die relevanten Funktionen sind folgende:

Funktion	Beschreibung
<code>addPlant(plant)</code>	Fügt eine Pflanze dem Grid hinzu.
<code>removePlant(plant)</code>	Entfernt eine Pflanze vom Grid.
<code>addEnemies(ec)</code>	Platziert ein Fressfeindcluster auf dem Grid.
<code>removeEnemies(ec)</code>	Entfernt ein Fressfeindcluster vom Grid.
<code>addSubstance(substance)</code>	Fügt eine Substanz dem Grid hinzu.

Tabelle 10: Funktionen zum Hinzufügen und Entfernen von Objekten

Beim Hinzufügen einer Pflanze wird die Position des übergebenen Objekts verwendet, um die entsprechende Position im Grid zu identifizieren. Diese wird dann an erster Stelle des Tupels im Grid gespeichert und gleichzeitig im Klassenattribut `plants` abgelegt, um einen späteren Zugriff auf das Objekt zu ermöglichen. Beim Entfernen wird auf die Datenstrukturen zugegriffen, in denen die übergebene Pflanze gespeichert wurde. Sie wird entfernt, und im Fall des Positionstupels wird der Wert auf `None` zurückgesetzt.

Für die Fressfeindcluster wird das gleiche Prinzip angewendet, wobei im Positionstupel auf die zweite Position zugegriffen wird. Da für die Fressfeinde im Positionstupel eine Liste der tatsächlich auf dem Gridfeld befindlichen Objekte angelegt wurde, wird diese beim Entfernen nicht wie bei den Pflanzen auf `None` gesetzt, sondern das betreffende Fressfeindcluster wird lediglich aus der Liste entfernt. Falls zu diesem Zeitpunkt andere Fressfeindcluster ebenfalls auf demselben Gridfeld platziert sind, bleiben diese unverändert.

Da es bei den Substanzen zwei verschiedene Objektarten und Listen gibt, jedoch

nur eine globale Funktion zum Hinzufügen, ist ein zusätzlicher Zwischenschritt erforderlich. Dieser besteht darin, dass die übergebene Substanz zunächst auf ihren Typ überprüft wird – ob es sich um einen Signalstoff oder einen Giftstoff handelt. Entsprechend dieser Eigenschaft wird die Substanz entweder in die Liste `signals` oder `toxins` eingefügt.

4.5.3 Visuelle Darstellung des Grids in der Konsole

In einem Grid können die verschiedenen Zellen unterschiedliche Formate aufweisen. Neben den unterschiedlichen Positionsformaten muss das Grid auch in der grafischen Oberfläche visuell simuliert werden. Daher ist eine klare Trennung der verschiedenen Gridfeld-Formate erforderlich.

Um diese klare Trennung zu gewährleisten, wurde in der Klasse `Grid` eine Funktion `displayGrid` implementiert, die verschiedene Unterfunktionen enthält, die jeweils ein bestimmtes Positionsformat behandeln. Zur besseren Nachvollziehbarkeit wird das Grid so formatiert, dass es zu jedem Zeitpunkt der Simulation in der Konsole angezeigt werden kann.

Ein Simulationsschritt in einer Simulation kann dabei wie folgt aussehen:

```

Simulation-Step: 12
Grid-Energy: 284.80000000000007
Enemy-Number: 21
-----
----- ----- ----- ----- ----- ----- ----- -----
----- ----- ----- ----- ----- ----- ----- -----
----- ----- ----- ----- ----- ----- ----- -----
?????? ???? ?? ?????? ----- ----- ----- -----
?????? ???? ?? ?????? ?????? ----- ----- -----
96.0%* e1-#10 ???? ?? ?????? 97.6%+ e2-#9 ?????? 101.0%! e3-#2 -----
?????? ???? ?? ?????? ?????? ----- ----- -----
?????? ???? ?? ?????? ----- ----- -----
#####

```

Abbildung 6: Beispiel für die Ausgabe des Grids in der Konsole

Jedes Symbol in dieser Ausgabe hat eine spezifische Bedeutung. Zu sehen sind jeweils

drei Fressfeindcluster und Pflanzen. Die Pflanzen werden durch einen Prozentsatz dargestellt, der angibt, wie viel Prozent der initialen Energie sie noch besitzen, bezogen auf die vom Nutzer definierten Anfangsenergie. Die Fressfeindcluster werden in dieser Anzeige so gekennzeichnet, dass sowohl die Art als auch die Anzahl der Individuen, die zu diesem Zeitpunkt in dem Cluster vorhanden sind, ersichtlich sind. Es gibt jedoch nicht nur diese beiden Zellformate; weitere Formate sind der folgenden Tabelle zu entnehmen:

Zeichen	Bedeutung
-----	Leeres Gridfeld
$\mathcal{E}_{i,j}(t)\%$!	Mindestens ein Signalstoff wird produziert
$\mathcal{E}_{i,j}(t)\%$ >	Mindestens ein Signalstoff ist aktiv.
$\mathcal{E}_{i,j}(t)\%$ +	Mindestens ein Giftstoff wird produziert
$\mathcal{E}_{i,j}(t)\%$ *	Mindestens ein Giftstoff ist aktiv.
??????	Gridfeld innerhalb des Signalstoffradius
eh-#num	Fressfeindcluster der Art eh mit der Anzahl num.
eh-#num *	Vergiftetes Fressfeindcluster.

Tabelle 11: Bedeutungen der Gridfeld-Formate in der Konsolenausgabe

Ergänzend zur Tabelle werden die Signalstoffe, die über die Luft verbreitet werden und auf gemeinsamen Gridfeldern aktiv sind, durch mehrere Zeilen von Fragezeichen dargestellt. Dabei entspricht die Anzahl der Fragezeichen-Zeilen der Anzahl an verschiedenen Signalstoffen, die entweder unterschiedliche Namen tragen oder von unterschiedlichen Pflanzen produziert werden.

4.5.4 Handhabung der Fressfeindcluster im Grid

Zur Verarbeitung der Bewegung der verschiedenen Fressfeindcluster im Grid wurde die Funktion `collectAndManageEnemies()` implementiert. Diese Funktion durchläuft alle Positionen im Grid, entpackt die Tupel auf den Gridfeldern und speichert – sofern eine Liste mit Fressfeindclustern vorhanden ist – die einzelnen Fressfeindcluster zusammen mit ihrer Position als Tupel in einer Liste.

Anschließend ruft die Funktion `collectAndManageEnemies()` die Funktion `manageEnemyClusters(moveArr)` auf, wobei die zuvor erstellte Liste der Fressfeindcluster und deren Positionen als Parameter übergeben wird.

In der Funktion `manageEnemyClusters(moveArr)` wird über die Elemente der zuvor erstellten Liste iteriert und geprüft, ob sich das Fressfeindcluster entsprechend der vom Nutzer definierten Bewegungsgeschwindigkeit um ein Feld weiterbewegen darf.

```

1 def manageEnemyClusters(self, moveArr):
2     for ec, oldPos in moveArr:
3         if not isinstance(ec, EnemyCluster):
4             continue
5         if not self.canMove(ec):
6             continue
7         path = ec.chooseRandomPlant(ec.position)
8         newPos = self.processEnemyMovement(ec, oldPos, path)
9         self.updateEnemyClusterPos(ec, oldPos, newPos)
10        self.processInteractionWithPlant(ec)
11        self.reduceClusterSize(ec)
12        self.processNewPathAfterDisplace()
13        self.afterEffectTimeAfterDeath()

```

Quelltext 8: Verarbeitung der Bewegung der Fressfeindcluster auf dem Grid

Basierend auf dem Pfad, der durch die Funktion `chooseRandomPlant(ec, position)` ermittelt wird, bestimmt die Funktion `processEnemyMovement(ec, oldPos, path)` die nächste Position entlang dieses Pfades. Nachdem die neue Position des Fressfeindclusters festgelegt wurde, wird das Cluster mithilfe der Funktion `updateEnemyClusterPos(ec, oldPos, newPos)` auf die ermittelte Position verschoben, sofern eine Bewegung zulässig ist.

4.5.5 Interaktionen zwischen Fressfeindclustern und Pflanzen

Da ein Fressfeindcluster zu unterschiedlichen Zeitpunkten eine Zielpflanze erreichen kann, wird in diesem Fall die Funktion `processInteractionWithPlant(ec)` aufgerufen. Diese Funktion dient der Regelung der Interaktion zwischen dem Fressfeindcluster und der Pflanze.

```

1 def processInteractionWithPlant(self, ec):
2     for i, row in enumerate(self.grid):
3         for j, (plant, _) in enumerate(row):
4             if isinstance(plant, Plant):
5                 dist = self.getDistance(ec.position, (i, j))
6                 self.plantAlarmAndSignalProd(ec, dist, plant)
7                 self.plantAlarmAndPoisonProd(ec, dist, plant)
8             if (i, j) == ec.position:
9                 ec.lastPlant = plant
10                ec.currentPath = []
11                self.eatAndReproduce(ec, plant)
12                self.processSignalEffects(ec, plant)
13                self.processToxinEffects(ec, plant)
14 ...

```

Quelltext 9: Verarbeitung der Interaktion zwischen Pflanze und Fressfeindcluster

Hier wird über die Felder des Grids iteriert, wobei verschiedene Funktionen ausgelöst werden, die spezifische Prozesse initiieren, sobald die Fressfeinde mit einer Pflanze interagieren.

Einerseits werden die Funktionen `plantAlarmAndSignalProd(ec, dist, plant)` und `plantAlarmAndPoisonProd(ec, dist, plant)` aufgerufen. In diesen Funktionen wird die Alarmierung der Pflanze sowie die Produktion der jeweiligen Substanz eingeleitet. Die Produktion eines Signal- oder Giftstoffs erfolgt jedoch nur, sofern die entsprechenden Bedingungen gemäß der formalen Definition sowie den vom Nutzer festgelegten Triggern erfüllt sind.

Zudem wird in der oben genannten Funktion im Quelltext 9 überprüft, ob die Position der Pflanze und des Fressfeindclusters identisch sind. In diesem Fall wird das interne Attribut `lastPlant` des Fressfeindclusters – wie bereits erwähnt – auf die aktuell besuchte Pflanze aktualisiert.

Da sich das Fressfeindcluster und die Pflanze auf demselben Gridfeld befinden, können die Fressfeinde Energie von der Pflanze aufnehmen. Entsprechend den vom Nutzer festgelegten Parametern zur Essgeschwindigkeit und zum Fresserfolg können sie

neue Individuen erzeugen. Dieser Prozess wird durch die Funktion `eatAndReproduce(ec, plant)` gesteuert.

Da sich die Pflanzen verteidigen können, müssen bei der Präsenz eines Signal- oder Giftstoffs deren Effekte auf die Fressfeindcluster berücksichtigt werden, da sie das zukünftige Verhalten der Cluster beeinflussen können, was dann in einem noch folgenden Kapitel thematisiert wird.

4.5.6 Substanzproduktion bei Interaktion

Die beiden Funktionen `plantAlarmAndSignalProd(ec, dist, plant)` und `plantAlarmAndPoisonProd(ec, dist, plant)` sind ähnlich aufgebaut. Im Fall der ersten Funktion wird über die Signalstoffe in der Liste `signals` iteriert. Innerhalb dieser Schleife wird überprüft, ob die übergebenen Parameter der Funktion mit mindestens einer Triggerkombination übereinstimmen.

Falls eine Triggerkombination erfüllt ist, werden zwei weitere Funktionen aufgerufen, die dann die Alarmierung der Pflanze und dann die Produktion des Signalstoffs verarbeiten.

```

1 def processSignalAlarm(self, ec, dist, plant, signal, trigger):
2     triggerEnemy, minClusterSize = trigger
3
4     if ec.num < minClusterSize and ec.num > 0:
5         return
6
7     if not plant.isSignalAlarmed(signal) and not plant.isSignalPresent
8         (signal) and dist < 1:
9         plant.enemySignalAlarm(signal)
10        signal.signalCosts(plant)

```

Quelltext 10: Funktion zur Verarbeitung der Alarmierung einer Pflanze

Innerhalb dieser Funktion wird überprüft, ob die Mindestanzahl an Individuen des betrachteten Fressfeindclusters die Mindestanforderungen für das Auslösen der Signalstoffproduktion erfüllt. Falls dies nicht der Fall ist, wird zurück gegangen in die übergeordnete Funktion `plantAlarmAndSignalProd(ec, dist, plant)`. Falls aber die Mindestanforderungen erfüllt sind, wird die Produktion des betrachteten Signalstoffs gestartet. In anderen Worten wird im Hintergrund ein internes Attribut

des Signal-Objekts auf alarmiert gesetzt. Diese Umstellung bedeutet, dass die Signalstoffproduktion gestartet ist, wodurch die verfügbare Energie der Pflanze nun in jedem Zeitschritt um die Kosten des Signalstoffs reduziert wird.

Im Hintergrund wird ein Zähler pro Zeitschritt erhöht, der dann in der Funktion `processSignalProduction(ec, plant, signal)` überprüft wird.

```

1 def processSignalProduction(self, ec, plant, signal):
2     if plant.isSignalPresent(signal):
3         return
4     if plant.isSignalAlarmed(signal) and not plant.isSignalPresent(
5         signal):
6         if plant.getSignalProdCounter(ec, signal) < signal.prodTime:
7             plant.incrementSignalProdCounter(ec, signal)
8         else:
9             plant.makeSignal(signal)
10            signal.activateSignal()
11            signal.signalCosts(plant)

```

Quelltext 11: Funktion zur Verarbeitung der Produktion eines Signalstoffs

Sobald der Produktionszähler die vom Nutzer definierte Produktionszeit erreicht hat, wird der Signalstoff auf aktiv gesetzt, wodurch er seine Wirkung an der Pflanze entfaltet.

Eine mögliche Wirkung des Signalstoffs ist, dass die Produktion eines Giftstoffs, der von diesem spezifischen Signalstoff abhängt, ausgelöst wird. Die Prozeduren zur Produktion und Aktivierung eines Giftstoffs sind mit denen für den Signalstoff identisch, wobei jedoch bei den Triggern zusätzlich die Präsenz eines Signalstoffs überprüft wird. Mit anderen Worten, die Funktionen rund um `plantAlarmAndPoisonProd(ec, dist, plant)`, `processToxinAlarm(ec, dist, plant, signal, toxin, trigger)` und `processToxinProduction(ec, plant, toxin)` sind gleich aufgebaut, wurden jedoch für Giftstoffe angepasst, sodass sie die entsprechenden Datenstrukturen für Giftstoffe im Hintergrund verwenden.

4.5.7 Substanzeffekte auf die Fressfeinde

Wie bereits bei der Vorstellung der Funktion `processInteractionWithPlant(ec)` erläutert, wird jeweils eine Funktion zur Verarbeitung der Effekte von Signalstoffen und Giftstoffen aufgerufen.

`processSignalEffects(ec, plant)` verarbeitet die Handlungen, wenn ein Signalstoff aktiv ist. In diesem Zusammenhang bedeutet „Handlung“, dass überprüft wird, ob es sich um einen Signalstoff handelt, der sich über Gridverbindungen oder über die Luft verbreitet. Trifft dies zu, ruft `processSignalEffects(ec, plant)` weitere Funktionen auf, die jeweils die Kommunikation der beiden verschiedenen Signalstoffarten verarbeiten. Die Funktion ist wie folgt aufgebaut:

```

1 def processSignalEffects(self, ec, plant):
2     for signal in self.signals:
3         for trigger in signal.triggerCombination:
4             triggerEnemy, minClusterSize = trigger
5             if triggerEnemy == ec.enemy.name:
6                 self.symCommunication(ec, plant, signal)
7                 self.airCommunication(ec, plant, signal)
8             if plant.currEnergy < plant.minEnergy:
9                 self.removeSignalRadius(plant, signal)

```

Quelltext 12: Funktion zur Verarbeitung von Signalstoffeffekten

Des Weiteren wird in dieser Funktion überprüft, ob die aktuelle Energie der Pflanze über der Mindestenergie liegt. Falls ein Signalstoff über die Luft verbreitet wird und die Pflanze aufgrund zu geringer Energie verstirbt, muss der Signalstoffradius ordnungsgemäß vom Grid entfernt werden. Dieser Fall wird durch die Funktion `removeSignalRadius(plant, signal)` berücksichtigt.

Auch die Funktion `processToxinEffects(ec, plant)` für die Giftstoffe ist ähnlich aufgebaut wie die Funktion zur Verarbeitung der Signalstoffeffekte. Der Unterschied besteht jedoch darin, dass hier eine Unterscheidung der Giftstoffe hinsichtlich ihrer Wirkung getroffen wird.

In anderen Worten, es werden zwei Funktionen aufgerufen: `processNonDeadlyToxin(toxin, ec, plant, signal)` zur Verarbeitung der Effekte von vertreibenden Giftstoffen und `processDeadlyToxin(toxin, ec, plant, signal)` zur Verarbeitung der Effekte von tödlichen Giftstoffen.

4.5.8 Zurücksetzen der Substanzverfügbarkeit

In einer Simulation kann es immer wieder zu Situationen kommen, in denen die Verfügbarkeit von Signalstoffen und Giftstoffen zurückgesetzt werden muss. Dafür wurden verschiedene Funktionen implementiert, die diese Situationen berücksichtigen und die jeweilige Verfügbarkeit der Substanzen an der Pflanze zurücksetzt.

Eine klassische Situation tritt ein, wenn die Nachwirkzeit eines Signalstoffs abgelaufen ist, weil die Pflanze nicht mehr von einem Fressfeindcluster bedroht wird, was den Signalstoff ausgelöst hat. In diesem Fall ist die Funktion `handleAfterEffectTime(ec, plant, signal)` zuständig. Diese wird in der Funktion `plantAlarmAndSignalProd(ec, dist, plant)` aufgerufen und sorgt dafür, dass der Signalstoff von der Pflanze entfernt wird, sobald der vom Nutzer definierte Parameter für die Nachwirkzeit abgelaufen ist.

Ähnlich muss auch bei den Giftstoffen die Verfügbarkeit eines Giftstoffs entfernt werden. Hier wird jedoch nicht über eine Nachwirkzeit gearbeitet, sondern geprüft, ob ein Fressfeindcluster, das den betrachteten Giftstoff auslösen kann, noch auf dem gleichen Feld wie die produzierende Pflanze steht. Es gibt Situationen, in denen das Fressfeindcluster von der Pflanze verjagt wird und kein anderer auslösender Fressfeind auf dem Feld vorhanden ist. In diesem Fall wird die Verfügbarkeit des Giftstoffs zurückgesetzt. Auch wenn die Wirkung des Giftstoffs tödlich für das auslösende Fressfeindcluster ist, kann es dazu kommen, dass das Cluster ausstirbt. In diesem Fall wird die Pflanze ebenfalls nicht mehr von diesem Fressfeindcluster bedroht, und die Verfügbarkeit des Giftstoffs wird auch hier zurückgesetzt.

Um die Verfügbarkeit des Giftstoffs zurückzusetzen, wurde die Funktion `resetToxically(ec, toxin, plant)` implementiert, die in der Funktion `plantAlarmAndPoisonProd(ec, dist, plant)` aufgerufen wird und den zuvor beschriebenen Sachverhalt verarbeitet und umsetzt.

4.5.9 Handhabung der Kommunikation zwischen Pflanzen

Wie bereits bei der Vorstellung der Funktion `processSignalEffects(ec, plant)` erwähnt, werden zwei Funktionen, `symCommunication(ec, plant, signal)` und `airCommunication(ec, plant, signal)`, aufgerufen. Diese Funktionen sind dafür verantwortlich, die Kommunikation zwischen Pflanzen zu überprüfen bzw. zu verarbeiten.

`symCommunication(ec, plant, signal)` überprüft, ob benachbarte Pflanzen miteinander verbunden sind. Falls dies der Fall ist, wird die Funktion `symInteraction(sPlant, rPlant, signal, ec)` aufgerufen. Diese Funktion ist dafür zuständig, das Senden von Signalstoffen über die Gridverbindungen zu verarbeiten. Ein Signalstoff wird an der empfangenden Pflanze als verfügbar gekennzeichnet, wenn der vom Nutzer definierte Parameter für die Sendegeschwindigkeit nicht kleiner ist als der aktuelle Sendezähler für das Senden von Signalstoffen. In diesem Fall wird die Funktion `sPlant.sendSignal(rPlant, signal)` aufgerufen, die dafür sorgt, dass die empfangende Pflanze `rPlant` nun den gesendeten Signalstoff besitzt. Gleichzeitig wird der Sendezähler für diesen Signalstoff zurückgesetzt, sodass der Signalstoff bei Erreichen der Sendegeschwindigkeit des Sendezählers an eine benachbarte Pflanze weitergesendet werden kann. Dies ermöglicht die Verbreitung des Signalstoffs über mehrere Gridverbindungen. Wenn der Sendezähler jedoch kleiner ist als die Sendegeschwindigkeit, wird eine Funktion aufgerufen, die den Sendezähler um 1 erhöht, da der Wert der Sendegeschwindigkeit noch nicht erreicht wurde.

Einen ähnlichen Aufbau besitzt die Funktion zur Kommunikation über die Luftverbreitung. Hier wird neben der Berechnung des sich über die Zeitschritte vergrößernden Signalstoffradius auch die Reaktion empfangender Pflanzen in der Funktion `airInteraction(plant, signal, ec)` verarbeitet.

Ähnlich wie bei den Funktionen zur Verarbeitung von gesendeten Signalstoffen über die Gridverbindungen, wird in `airInteraction(plant, signal, ec)` die Verbreitung eines Signalstoffs bearbeitet. Der Unterschied besteht darin, dass sich diese Funktion auf Signalstoffe bezieht, die sich über die Luft verbreiten. Auch hier wird ein Sendezähler hochgezählt.

Es wird geprüft, ob der vom Nutzer definierte Parameter für die Sendegeschwindigkeit größer oder gleich dem Sendezähler ist. Wenn der Sendezähler kleiner ist als der definierte Parameter, wird er um 1 erhöht. Sobald die zwei Werte für den spezifischen Signalstoff übereinstimmen, wird der Radius entsprechend der euklidischen Distanz um eine Gridfeldebene erweitert.

4.5.10 Verbindung zwischen der Grid- und der Simulations-Klasse

Die Verbindung zwischen der Klasse `Simulation` und der Klasse `Grid` besteht darin, dass ein Objekt der Klasse `Simulation` ein Attribut besitzt, das auf ein Objekt der `Grid`-Klasse verweist.

In der Klasse `Simulation` ist folgende Funktion realisiert:

```
1 def runStep(self):
2     for plant in self.grid.plants[:]:
3         plant.grow()
4         plant.survive()
5         plant.scatterSeed()
```

Quelltext 13: Funktion für die Simulation eines Zeitschritts für die Pflanzen

In der Klasse `Simulation` wird durch die Liste der Pflanzen iteriert, die im `Grid`-Objekt hinterlegt sind. In jeder Iteration werden drei Funktionen ausgeführt: Zuerst wächst die Pflanze entsprechend ihren festgelegten Parametern. Danach wird überprüft, ob die Mindestanforderungen für das Überleben der Pflanze erfüllt sind. Schließlich wird geprüft, ob das Reproduktionsintervall erreicht ist, sodass neue Pflanzen gesetzt werden können.

Zusätzlich werden sechs Funktionen definiert, die die Abbruchbedingungen für eine Simulation festlegen und überprüfen. Diese Funktionen bestimmen, ob bestimmte Kriterien erfüllt sind, die das Ende der Simulation festlegen. Jede Funktion entspricht einer bestimmten Abbruchbedingung, die in der folgenden Tabelle 12 festgehalten sind. Die Überprüfung dieser Bedingungen stellt sicher, dass die Simulation gestoppt wird, sobald eine der festgelegten Kriterien erfüllt ist.

Funktion	Beschreibung
noSpecificPlantBreak	Überprüft, ob es keine Pflanze einer bestimmten Art gibt.
noSpecificEnemyBreak	Überprüft, ob es kein Fressfeindcluster mit einer bestimmten Fressfeindart gibt.
noEnemiesBreak	Überprüft, ob es keine Fressfeindcluster auf dem Grid gibt.
noPlantsBreak	Überprüft, ob es keine Pflanzen auf dem Grid gibt.
upperGridEnergyBreak	Überprüft, ob die gesamte Energie der Pflanzen eine Grenze überschreitet.
upperEnemyNumBreak	Überprüft, ob die Gesamtanzahl an Individuen der Feinde eine Grenze überschreitet.

Tabelle 12: Funktionsnamen für die Abbruchbedingungen und deren Beschreibung

Um den Verlauf einer Simulation in Form von Diagrammen darzustellen, werden in dieser Klasse Funktionen zur Erfassung und Aufbereitung der Simulationsdaten implementiert. Eine zentrale Funktion ist dabei `getPlantData(timeStep)`, die Daten über die Pflanzen sammelt und zur Visualisierung sowohl der verfügbaren Energie als auch der Anzahl an pflanzlichen Vorkommen im Grid dient. Für die Erfassung der Daten der Fressfeinde wird die Funktion `getEnemyData(timeStep)` verwendet. Im Gegensatz zu den Pflanzen werden hier sowohl die Anzahl der Individuen einer bestimmten Fressfeindart als auch die Anzahl der Cluster einer bestimmten Art in den Diagrammen dargestellt.

Um eine Simulation zu starten, wurde die Funktion `run(maxSteps, plant, ec, maxGridEnergy, maxEnemyNum)` implementiert. Diese Funktion initialisiert zunächst das Grid und startet anschließend eine `while`-Schleife, die so lange ausgeführt wird, bis eine der vordefinierten Abbruchbedingungen erfüllt ist. Ein entsprechender Ausschnitt dieser Funktion ist im folgenden Quelltext dargestellt:

```

1 def run(self, maxSteps, plant, ec, maxGridEnergy, maxEnemyNum):
2     self.displayInit()
3     count = 1

```

```
4     while True:
5
6         if count - 1 == maxSteps:
7
8             break
9
10        ...
11
12        self.clearConsole()
13
14        print('\nSimulation-Step:', count)
15
16        self.runStep()
17
18        self.grid.collectAndManageEnemies()
19
20        ...
```

Quelltext 14: Ausschnitt der `run`-Funktion aus der Klasse `Simulation`

Innerhalb dieser Schleife wird unter anderem die zentrale Funktion `grid.collectAndManageEnemies()` aus der `Grid`-Klasse aufgerufen. Diese Funktion ist essenziell für die Steuerung der Feindbewegung, die Produktion von Signal- und Giftstoffen, das Zurücksetzen der Substanzverfügbarkeit sowie die Kommunikation zwischen Pflanzen über den jeweils gewählten Kanal. Mit anderen Worten stellt diese Schlüsselfunktion das zentrale Element dar, auf dem die gesamte Simulation des Netzwerks basiert.

Zusammenfassend lässt sich festhalten, dass die `Grid`-Klasse das Gesamtmodell repräsentiert, einschließlich der Akteure und Substanzen, während die Klasse `Simulation` dieses Modell mithilfe definierter Parameter steuert und mit der `run`-Funktion die Simulation ausführt.

4.6 Der Dateimanager

Damit die erzeugten Simulationen nicht verloren gehen, wenn das Tool geschlossen wird, wurde ein Dateimanager implementiert. Dieser umfasst die Klasse `Export`, die für die Erstellung von Dateien und das Speichern der konfigurierten Netzwerke zuständig ist, sowie die Klasse `Import`, die der Rekonstruktion der abgespeicherten Netzwerke dient und eine ausgewählte Datei wieder in das Tool einliest.

In diesem Kapitel werden diese beiden Klassen sowie ihre Implementierung erläutert.

Für die Umsetzung wurde dabei auf die Funktionen des Python-Modul `pickle` zur Erstellung und zum Laden von Dateien zurückgegriffen.

4.6.1 Die Klasse zum Exportieren

Ein Objekt der Klasse `Export` besitzt zwei Attribute, zum einen einen String für den Pfad zum Speicherort. Das zweite Attribut ist ein Objekt der Klasse `Grid`, die alle wichtigen Informationen zu Gridgröße, Parametern, Pflanzen, Fressfeindclustern und Substanzen beinhaltet, die es zum Exportieren eines Netzwerks benötigt.

Attribute der Klasse Export					
#	Attribut	Typ	#	Attribut	Typ
1	path	String	2	grid	Grid

Tabelle 13: Attribute eines `Export`-Objekts mit Datentypen

Damit die verschiedenen Daten erfasst werden können, wurde eine Funktion `getData()` implementiert. Diese Funktion sammelt die relevanten Netzwerkinformationen, indem sie auf das Attribut `grid` zugreift. Konkret bedeutet dies für Pflanzen, Fressfeinde und Substanzen, dass jeweils auf die in der Klasse `Grid` als interne Attribute deklarierten Speicherlisten zugegriffen wird.

Für die Gridverbindungen wird über die Pflanzen iteriert und überprüft, ob ein Eintrag für die betrachtete Pflanze im internen Attribut `gridConnections` der Pflanze existiert. Bei diesem Attribut handelt es sich um ein Dictionary, in dem die Gridverbindungen zu benachbarten Pflanzen gespeichert werden. Existiert ein entsprechender Eintrag in dieser Datenstruktur, so liegt eine Gridverbindung vor.

Während der Iteration wird ein neues Dictionary erzeugt, das die erkannten Gridverbindungen über alle Pflanzen hinweg speichert.

Die dazugehörige Funktion wurde wie folgt implementiert:

```

1 def getDate(self):
2     grid = self.grid
3     plants = self.grid.plants
4     cluster = self.grid.enemies
5     signals = self.grid.signals
6     toxins = self.grid.toxins

```

```
7     plant_connections = {plant.name: plant.gridConnections for plant
8         in plants}
9
10    return {'grid': grid, 'plants': plants, 'cluster': cluster,
11            'signals': signals, 'toxins': toxins, 'connections':
12            plant_connections}
```

Quelltext 15: Hilfsfunktion zum Sammeln von Informationen für den Export

Es ist zu erkennen, dass die Funktion ein Dictionary mit entsprechenden Schlüsseln für die spezifischen Informationen zurückgibt.

Die Funktion dient in diesem Sinne als Hilfsfunktion für die eigentliche Speicherung eines Netzwerks. Zum Speichern wurde die Funktion `save()` implementiert, die den Rückgabewert der Hilfsfunktion übernimmt und versucht, ihn in einer Datei an einem vom Nutzer festgelegten Speicherort abzulegen. Die Speicherung erfolgt, indem die Datei im Binärmodus geöffnet und mit den Daten des konfigurierten Netzwerks mit `pkl.dump(data, file)` aus dem Python-Modul `pickle` beschrieben wird.

4.6.2 Die Klasse zum Importieren

Die Klasse `Import` verfügt über nur ein Attribut, das den Pfad zur ausgewählten Datei in Form eines Strings speichert. Um ein Netzwerk einlesen zu können und die Konfigurationen korrekt in der graphischen Oberfläche anzuzeigen, müssen die Daten aus der eingelesenen Datei rekonstruiert werden.

Zur Rekonstruktion wurde folgende Funktion implementiert:

```
1 def reconstructData(self, data):
2     grid = data['grid']
3
4     self.reconstructPlants(data, grid)
5     self.reconstructEnemies(data, grid)
6     self.reconstructSignals(data, grid)
7     self.reconstructToxins(data, grid)
8     self.reconstructConnections(data, grid)
9
10    return grid
```

Quelltext 16: Funktion zum Rekonstruieren des Netzwerks

Der Parameter für diese Funktion wird durch die Funktion `load(path)` erzeugt, wobei die von `pickle` bereitgestellte Funktion `pkl.load(file)` die Datei öffnet und

den Inhalt in der Variable `data` speichert.

Da ein konfiguriertes Netzwerk als Dictionary in der Datei gespeichert wurde, wird beim Einlesen ebenfalls ein Dictionary ausgelesen. In der Funktion aus dem Quelltext 16 kann nun mit dem entsprechenden Schlüssel auf die Werte im Dictionary zugegriffen werden.

Der Einfachheit halber wurde für die Pflanzen, Fressfeindcluster, Signalstoffe, Giftstoffe und Gridverbindungen jeweils eine Hilfsfunktion zur Rekonstruktion implementiert, die dann in `reconstructData(data)` aufgerufen werden und die entsprechenden Daten in den internen Attributen des `Grid`-Objekts speichern.

Zusammenfassend lässt sich sagen, dass in der Klasse zum Importieren die Daten rekonstruiert werden, sodass sie in der grafischen Oberfläche verwendet werden können, um die Konfigurationen zu laden und den abgespeicherten Zustand wieder anzuzeigen.

4.7 Die graphische Oberfläche und ihre Bedienung

Für die graphische Oberfläche ist es sinnvoller, die implementierten Elemente anhand der resultierenden Oberfläche zu präsentieren, da die Klasse `GUI` einen sehr umfangreichen Quelltext enthält.

In diesem Kapitel werden die verschiedenen Elemente der grafischen Benutzeroberfläche sowie deren Verwendungszweck erläutert. Zudem wird eine eigenständige Benutzeranleitung bereitgestellt, die ausreicht, um das Tool ohne weitere Erläuterungen zu bedienen.

4.7.1 Der Aufbau des Hauptfensters

Im Wesentlichen besteht die graphische Oberfläche aus drei Bereichen, darunter eine Kontrollleiste, mit der die Rahmenbedingungen für das Grid festgelegt werden können. Dazu gehört die Definition der Gridgröße sowie die Festlegung der Anzahl der verschiedenen Akteure und Substanzen.

Zudem stehen verschiedene Buttons zur Verfügung, um eine Simulation des erstellten Netzwerks zu steuern und spezifische Aktionen auszuführen.

Grid-Size:	30	#Plants:	4	#Enemies:	2	#Substances:	3	<input type="button" value="Apply"/>
------------	----	----------	---	-----------	---	--------------	---	--------------------------------------

(a) Eingabefelder in der Kontrollleiste



(b) Bedienelemente der Kontrollleiste

Abbildung 7: Elemente der Kontrollbar

In Abbildung 7a sind die vier Eingabefelder dargestellt, die jeweils eine natürliche Zahl als Eingabe erwarten.

Die weiteren Bedienelemente in 7b sind verschiedene Buttons, mit denen unterschiedliche Aktionen ausgeführt werden können. Eine detaillierte Beschreibung dazu findet sich in der Bedienungsanleitung im nächsten Kapitel.

Sind die Eingaben in der Kontrollleiste gültig, wird ein leeres Grid mit der entsprechenden Anzahl an Akteuren erzeugt. Unterhalb der Kontrollleiste befinden sich zwei nebeneinander liegende Bereiche: Auf der linken Seite sind Eingabeelemente zur Parameterkonfiguration für die Akteure und die Substanzen, während auf der rechten Seite das Grid angezeigt wird.

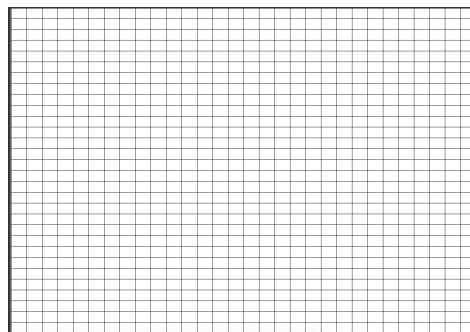


Abbildung 8: Leeres Grid nach manueller Initialisierung mit einer Größe von

30x30

Die linke Seite für die Parameterkonfiguration ist in drei Tabs unterteilt. Im ersten Tab werden die Eingabefelder für die verschiedenen Parameter der definierten Pflanzenarten angezeigt. Im zweiten Tab erscheinen die Eingabefelder für die vom Nutzer definierbaren Parametern der Fressfeindarten. Der dritte Tab ist für die Eingabe der Parameter der verschiedenen Substanzen vorgesehen.

Im Nachfolgenden die visuelle Darstellung der drei Tabs:

The figure consists of three side-by-side screenshots of a software interface. Each screenshot shows a tabbed interface with three tabs: 'Plants', 'Enemies', and 'Substances'. The 'Plants' tab (a) shows four entries for 'Plant 1' through 'Plant 4', each with a small green icon, followed by input fields for 'Init-Energy', 'Growth-Rate', 'Min-Energy', 'Repro-Interval', 'Offspring-Energy', and 'Min-Distance'. The 'Enemies' tab (b) shows two entries for 'Enemy 1' and 'Enemy 2', each with a small grey icon, followed by input fields for 'Cluster-Size', 'Speed', 'Eat-Speed', and 'Eat-Victory'. The 'Substances' tab (c) shows three entries for 'Substance 1', 'Substance 2', and 'Substance 3', each with a small grey icon, followed by dropdown menus for 'Signal' or 'Deadly Toxin' and 'Symbiotic' or 'Non-Symbiotic', and input fields for 'Name', 'Producer', 'Receiver', 'Trigger', 'Prod-Time', 'Send-Speed', 'Energy-Costs', 'AfterEffectTime', and 'Eff-Strength'.

(a) Pflanzen

(b) Fressfeinde

(c) Substanzen

Abbildung 9: Ansicht der Tabs für die Parameter von Pflanzen, Fressfeinden und Substanzen

4.7.2 Bedienungsanleitung zur Verwendung des Tools

Beim Starten des Tools wird zunächst nur ein leeres Fenster mit der Kontrollleiste angezeigt. Um die Tabs für die Parameter und das leere Grid sichtbar zu machen, müssen über die Eingabefelder die Gridgröße sowie die Anzahl an Pflanzenarten, Fressfeindarten und Substanzen definiert und mit dem **Apply**-Button bestätigt werden.

Die Eingabefelder für ein Szenario akzeptieren jedoch nicht beliebige Werte, sondern sind durch festgelegte Grenzen eingeschränkt. Sie akzeptieren nur ganze Zahlen innerhalb der in der Implementierung definierten Modellgrenzen. Der Wert für die Gridgröße muss im Bereich von 1 bis 80 liegen, die Anzahl der Pflanzenarten im Bereich von 1 bis 16 und die Eingabewerte für die Fressfeinde und Substanzen dürfen nur im Bereich von 0 bis 15 liegen. Sollte ein Wert außerhalb dieser Grenzen eingegeben werden, gibt das Tool eine Fehlermeldung aus, die den Nutzer darüber informiert, dass die Eingabe fehlerhaft war, und bittet ihn, die falschen Parameter zu korrigieren, indem es den zulässigen Zahlenbereich anzeigt.

Bei erfolgreicher Eingabe wird ein leeres Szenario erstellt, in dem die Parameter für

Pflanzen, Fressfeinde und Substanzen individuell festgelegt werden können. Die Akteure mit ausgefüllten Parametern können an diesem Punkt auf dem Grid platziert werden.

Bevor jedoch die Tabs und das Grid behandelt werden, werden im Folgenden zunächst die Funktionen der weiteren Elemente der Kontrollleiste erläutert.

Der **Breakups**-Button öffnet ein kleines Fenster, in dem die verschiedenen Abbruchbedingungen für eine Simulation festgelegt werden können.



(a) Keine Abbruchbedingungen (b) Festgelegte Abbruchbedingungen

Abbildung 10: Fenster für die Abbruchbedingungen einer Simulation

In der obigen Abbildung 10 sind zum einen ein leeres Fenster zu sehen, wenn keine Abbruchbedingungen definiert sind, sowie ein Fenster mit ausgefüllten Eingabefeldern.

Im Fall von 10b bedeutet dies, dass eine Simulation beendet wird, wenn entweder 1000 Zeitschritte vergangen sind, alle Pflanzen der Art **p1** verstorben sind, alle Feinde der Art **e1** verstorben sind, eine obere Grenze von 10000 Einheiten pflanzlicher Energie erreicht oder eine maximale Anzahl von 20000 Fressfeinden erreicht wurde. Als nächstes in der Kontrollleiste folgt der **Play**-Button, dessen Funktion sich schnell erklären lässt: Er startet die Simulation, sobald der Nutzer mit der Konfiguration des Netzwerks fertig ist und diese starten möchte.

Das nächste Element in der Kontrollleiste ist ein farblich markiert Fläche, die den aktuellen Zeitschritt der Simulation anzeigt. „Farblich markiert“ bedeutet in diesem Zusammenhang, dass dem Nutzer über eine Signalfarbe mitgeteilt wird, ob die Simulation noch nicht gestartet wurde, ob sie aktuell läuft oder ob sie bereits beendet wurde.

Signalfarbe	Bedeutung
Rot	Simulation läuft noch nicht
Orange	Simulation läuft aktuell
Grün	Simulation wurde beendet

Tabelle 14: Bedeutung der Signalfarben der Zeitschrittanzeige in der Kontrollleiste

Der nächste Button in der Kontrollleiste ist der Plot-Button. Dieser öffnet ein Fenster, in dem für eine erfolgreich durchgeführte Simulation vier verschiedene Plots zu den Ereignissen angezeigt werden können.

Die Plots sind zur Gewährleistung einer übersichtlichen Darstellung in Tabs organisiert, zwischen denen der Nutzer wechseln kann.

Die vier Tabs und der jeweils enthaltene Plot sind der folgenden Tabelle zu entnehmen.

#	Titel	Plotinhalt
1	Plants-Energy	Gesamtenergie einer Pflanzenart über die Zeit
2	Plants-Count	Anzahl der Vorkommen einer Pflanzenart über die Zeit
3	Enemies-Size	Anzahl der Individuen einer Fressfeindart über die Zeit
4	Enemies-Count	Anzahl der Fressfeindcluster einer Art über die Zeit

Tabelle 15: Bedeutung der in den Plots dargestellten Informationen über die Zeit

Es ist wichtig zu beachten, dass das Fenster für die Plots nur angezeigt wird, wenn eine Simulation durchgeführt wurde, da die benötigten Informationen während der Simulation gesammelt werden. Andernfalls blockiert das Tool den Zugriff und gibt dem Nutzer eine Meldung aus, dass zunächst eine Simulation gestartet werden muss, um die zeitlichen Verläufe grafisch darzustellen.

Der Import-Button öffnet ein Fenster, in dem der Dateimanager des Endgeräts angezeigt wird. Der Nutzer kann dort nach exportierten Dateien suchen und diese auswählen, um sie in das Tool zu laden. Nach der Auswahl einer Datei werden alle definierten Parameter automatisch in entsprechende Eingabefelder übernommen, ebenso wie die Platzierung der Akteure auf dem Grid.

Der **Export**-Button öffnet auch ein Fenster, in dem ein konfiguriertes Netzwerk unter einem selbst gewählten Namen als Datei gespeichert werden kann. Da der Dateimanager des Tools auf dem Python-Modul `pickle` basiert, werden die Dateien im `.pk1`-Format gespeichert. Diese Dateiendung sollte nicht geändert werden, um eine reibungslose Wiederherstellung der Datei zu gewährleisten.

Der letzte Button in der Kontrollleiste ist der **Reset**-Button. Er setzt eine beendete Simulation auf den initialen Zustand zurück, sodass eine erneute Simulation gestartet werden kann, ohne dass das Netzwerk neu konfiguriert oder importiert werden muss.

Damit Pflanzen auf dem Grid platziert werden können, müssen die entsprechenden Eingabefelder im Tab für die Pflanzenarten, die platziert werden sollen, ausgefüllt werden. Eine Pflanzenart verfügt über sieben verschiedene Eingabefelder, die der folgenden Tabelle entnommen werden können.

Parameter	Beschreibung	Werte
Init-Energy	Initiale Energie einer Pflanze	\mathbb{N}
Growth-Rate	Wachstumsrate in % der initialen Energie	\mathbb{N}
Min-Energy	Mindestenergie zum Überleben	\mathbb{N}
Repro-Interval	Intervall nach dem Nachkommen erzeugt werden	\mathbb{N}_0
Offspring-Energy	Initiale Energie der Nachkommen	\mathbb{N}
Min-Distance	Minimale Reichweite für die Nachkommen	\mathbb{N}
Max-Distance	Maximale Reichweite für die Nachkommen	\mathbb{N}

Tabelle 16: Erklärung der Eingabefelder für die Parameter der Pflanze

Es ist wichtig zu erwähnen, dass das Eingabefeld für das Reproduktionsintervall alternativ zur Eingabe einer 0 auch leer gelassen werden kann, falls die Pflanze sich nicht reproduzieren soll. In diesem Fall interpretiert die graphische Oberfläche das leere Feld als 0.

Zudem wurde bei den Eingabefeldern für die Nachkommen berücksichtigt, dass, falls versehentlich ein kleinerer Wert für die maximale Reichweite als für die minimale Reichweite eingegeben wird, die maximale Reichweite automatisch auf den Wert

der minimalen Reichweite plus 1 gesetzt wird. Dadurch wird sichergestellt, dass die maximale Reichweite stets größer als die minimale Reichweite ist.

Eine Fressfeindart im Tab für die Fressfeinde verfügt über vier Eingabefelder für die Parameter, die der folgenden Tabelle entnommen werden können.

Parameter	Beschreibung	Wert
Cluster-Size	Anzahl an Individuen im Cluster	N
Speed	Laufgeschwindigkeit in Zeitschritten	N
Eat-Speed	Essgeschwindigkeit in pflanzlicher Energie	N
Eat-Victory	Benötigte Energie für einen Nachkommen	N

Tabelle 17: Erklärung der Eingabefelder für die Parameter der Fressfeinde

Bei den Signalstoffen gibt es aufgrund der verschiedenen Arten mehr Auswahl sowie mehr Eingabefelder. Zunächst kann der Nutzer über ein Auswahlmenü entscheiden, ob es sich um einen Signalstoff oder einen Giftstoff handelt. Handelt es sich um einen Signalstoff, kann der Nutzer zunächst auswählen, ob dieser sich über die Luft oder über die Gridverbindungen verbreiten kann.

Handelt es sich jedoch um einen Giftstoff, wird das Auswahlmenü für die Ausbreitung blockiert, sodass keine Auswahl möglich ist. Stattdessen wird die Checkbox rechts neben der Substanzart aktiviert, in der festgelegt werden kann, ob der Giftstoff eine tödliche Wirkung haben soll. Wird das Häkchen nicht gesetzt, hat der Giftstoff eine vertreibende Wirkung auf die Fressfeinde.

Für die Substanzen kann ein beliebiger Name gewählt werden. Der Eingabewert unterliegt keiner spezifischen Anforderung und kann sowohl Buchstaben als auch Zahlen enthalten, ganz nach den Wünschen des Nutzers.

Das zweite Eingabefeld dient der Angabe der Pflanzenarten, die die Substanz produzieren können. Eine Pflanzenart wird dabei mit „p“ gefolgt durch eine Zahl zwischen 1 und 16 entsprechend der Art eingegeben. Wenn mehrere Pflanzen die Substanz produzieren, können auch mehrere Arten eingegeben werden, die durch ein Komma getrennt sind. Das gleiche Prinzip gilt für das nächste Eingabefeld, in dem die Pflanzenarten angegeben werden, die einen Signalstoff empfangen können.

Das nächste Feld für die Trigger einer Substanz bietet unterschiedliche Eingabemöglichkeiten, je nachdem, ob es sich um einen Signalstoff oder einen Giftstoff handelt. Bei einem Signalstoff werden die Fressfeindarten, die den Signalstoff auslösen, durch „e“ gefolgt von einer Zahl zwischen 1 und 15 eingegeben. Da auch die Clustergröße eine Rolle spielt, wird diese durch ein Komma getrennt nach der Fressfeindart in das Eingabefeld eingetragen. Diese Eingabe aus zwei Elementen bildet einen Trigger, der erfüllt sein muss, damit der Signalstoff ausgelöst wird. Wenn mehrere Trigger existieren sollen, werden diese durch ein Semikolon getrennt. Eine akzeptierte Eingabe könnte beispielsweise `e1,1; e2,5; e3,2` sein.

Bei den Giftstoffen besteht ein Trigger aus drei Elementen, die jeweils durch ein Komma getrennt werden. Neben den Triggerelementen, wie sie auch bei den Signalstoffen verwendet werden, wird hier zusätzlich der Name des Signalstoffs eingegeben, der an der ersten Stelle des Triggers stehen muss. Mit anderen Worten: In den Triggern für die Giftstoffe wird zuerst der Signalstoffname, dann die Fressfeindart und schließlich die Clustergröße angegeben. Wie bei den Signalstoffen werden die verschiedenen Trigger durch ein Semikolon getrennt. Eine akzeptierte Eingabe könnte beispielsweise `s1,e1,1; s2,e2,5; s1,e3,2` sein.

Parameter	Beschreibung	Art	Aktiv
Prod-Time	Produktionszeit in Zeitschritten	Signalstoff	Ja
		Giftstoff	Ja
Energy-Costs	Energiekosten pro Zeitschritt	Signalstoff	Ja
		Giftstoff	Ja
Eli-Strength	Sterberate pro Zeitschritt	Signalstoff	Nein
		Giftstoff	Ja
Send-Speed	Sendegeschwindigkeit in Zeitschritten	Signalstoff	Ja
		Giftstoff	Nein
AfterEffectTime	Nachwirkzeit in Zeitschritten	Signalstoff	Ja
		Giftstoff	Nein

Tabelle 18: Eingabefelder für numerische Substanzparameter.

An dieser Stelle wurden nun alle wesentlichen Bestandteile behandelt, die erforderlich sind, um die Akteure auf dem Grid zu platzieren.

Damit Pflanzen bzw. Fressfeindcluster auf dem Grid platziert werden können, muss die zugehörige Checkbox aktiviert werden. Durch die Aktivierung wird der Akteur ausgewählt und kann damit auf dem Grid platziert werden.

Bei den Substanzen verhält es sich anders, da diese an die Pflanzen gekoppelt sind. Sie verfügen auch über eine Checkbox, die aktiviert werden muss, um dem Tool mitzuteilen, welche Substanzen in die Simulation aufgenommen werden sollen.

Ein Gridfeld besteht aus zwei Bereichen, einem inneren und einem äußeren, sodass es sich visuell wie folgt darstellen lässt.

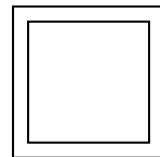


Abbildung 11: Aufbau eines Felds im Grid

Der innere Bereich ist für die Farben der ausgewählten Pflanzen zuständig, die beim Setzen den Bereich einfärbt. Der äußere Bereich hingegen zeigt die Verfügbarkeit einer Substanz an, die je nach Art in unterschiedlichen Farben dargestellt wird. Die verschiedenen Farben und deren Bedeutung können der folgenden Tabelle entnommen werden.

Farbe	Bedeutung
Gelb	Mindestens ein Signalstoff wird produziert und keins ist aktiv
Dunkelorange	Mindestens ein Signalstoff ist aktiv
Koralle	Mindestens ein Giftstoff wird produziert und keins ist aktiv
Rot	Mindestens ein Giftstoff ist aktiv

Tabelle 19: Bedeutung der Farben für die Substanzaktivität

Bei den Signalstoffen wird der Radius durch ein orangefarbenes Highlight auf den Feldern angezeigt, wobei der äußere Bereich eingefärbt wird, um den Nutzer zu zeigen, wo ein Signalstoff aktiv ist. Da mehrere Signalstoffe auf einem Gridfeld aktiv

sein können, wenn sie über die Luft verbreitet werden, wird die Farbe für jeden Signalstoff um einen kleinen Tick dunkler dargestellt.

Für die Signalstoffe, die über die Gridverbindungen versendet werden, muss die jeweilige Verbindung manuell zwischen den Pflanzen erstellt werden. Dazu muss mit einem Rechtsklick auf eine Pflanze ein Popup-Fenster geöffnet werden, in dem eine Liste der Nachbarpflanzen angezeigt wird. Eine Verbindung zur Nachbarpflanze kann nun nach Belieben hergestellt werden, indem die entsprechende Checkbox aktiviert wird.

Eine Gridverbindung wird durch eine schwarze Linie zwischen den benachbarten Pflanzen angezeigt. Sobald jedoch ein Signalstoff über die Verbindung gesendet wird, ändert sich die Farbe der Linie in grau.

Die Fressfeindcluster werden durch kleine Marker auf dem Gridfeld angezeigt, die zwei verschiedene Farben haben können. Sie erscheinen dunkelblau, wenn die Fressfeinde nicht tödlich vergiftet sind, also wenn sie sich im Normalzustand befinden. Sind sie jedoch tödlich vergiftet, wird der Marker auf dem Grid rot dargestellt.

In der graphischen Oberfläche existieren zwei Mechanismen zur Vergrößerung eines Clusters. Die erste Methode basiert auf der klassischen Wachstumsweise, bei der die Anzahl der Individuen innerhalb eines Clusters durch die Aufnahme pflanzlicher Energie zunimmt. Dieser Ansatz führt jedoch zu einem statischen Verhalten der Simulation. Da die Vermehrung der Fressfeinde ausschließlich auf dem Prinzip beruht, ist es schwierig, ein dynamisches Wechselspiel zwischen Pflanzen und Fressfeinden zu erzeugen, bei dem sich die Vorherrschaft beider Gruppen im Verlauf der Simulation abwechseln kann.

Um die Dynamik der Simulation zu erhöhen, wurde eine zusätzliche Logik implementiert. Diese überprüft, ob sich die Anzahl der Individuen im Vergleich zu initialen Clustergröße verdoppelt hat. Sobald diese Bedingung erfüllt ist, wird das Cluster in zwei separate Cluster aufgeteilt. In der graphischen Oberfläche äußert sich dies durch die Darstellung von zwei Markern, die jeweils die Hälfte der Individuen des ursprünglichen Clusters enthalten.

Damit die Anzahl der Cluster nicht unbegrenzt wächst, wurde die maximale Clusteranzahl auf die Hälfte der verfügbaren Gridfelder begrenzt. Diese Einschränkung

ermöglicht es den Fressfeinden, eine bestimmte Größe zu erreichen, sodass die Pflanzen eine Chance haben, sich zu behaupten und die Fressfeinde gegebenenfalls wieder zu verdrängen. Der genaue Verlauf der Simulation hängt jedoch maßgeblich von den gewählten Parametern ab.

Abschließend wird in diesem Kapitel eine Abbildung gezeigt, die darstellt, wie das Grid mit Pflanzen und Fressfeinden in der Simulation aussehen kann.

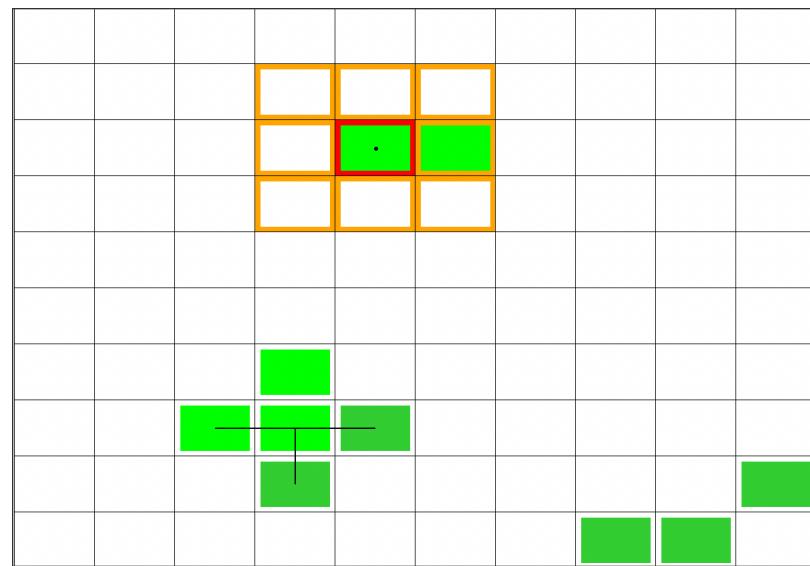


Abbildung 12: Beispiel eines befüllten Grids in der Simulation

5 Simulationsstudie

In diesem Kapitel wird eine Simulationsstudie durchgeführt, um die Funktionalität des Tools zu demonstrieren und die Ergebnisse anhand von vier Plots zu visualisieren. Die Simulation erstreckt sich über einen längeren Zeitraum, sodass zeitliche Veränderungen in den Plots nachvollziehbar dargestellt werden können.

Zunächst wird ein Szenario konfiguriert, das in das Tool eingegeben werden kann. Anschließend werden die Simulationsergebnisse in Abhängigkeit davon präsentiert, ob es sich um die Pflanzen oder deren Fressfeinde handelt.

5.1 Konfigurierung des Netzwerks

Um eine aussagekräftige Dynamik in der Simulation zu gewährleisten, wird ein Biotop mit einer Größe von $15 \cdot 15 = 225$ Gridfeldern gewählt. In diesem Biotop sind zwei verschiedene Pflanzenarten, zwei Arten von Fressfeinden sowie drei verschiedene Substanzen vorhanden.

Die Konfiguration der Akteure basiert auf spezifischen Parametern, die der folgenden Tabelle entnommen werden können.

Parameter	Plant 1	Plant 2	Parameter	Enemy 1	Enemy 2
Init-Energy	123	90	Cluster-Size	30	10
Growth-Rate	3	2	Speed	1	4
Min-Energy	20	10	Eat-Speed	1	10
Repro-Interval	30	35	Eat-Victory	20	30
Offspring-Energy	30	50			
Min-Dist	1	2			
Max-Dist	2	6			

Tabelle 20: Gewählte Parameter der Akteure für die Simulationsstudie

Der größte Unterschied zwischen den beiden Pflanzen besteht darin, dass sich die erste Art ausschließlich lokal verbreitet, während die zweite Art Nachkommen in einem größeren Radius verbreitet. Um diesen Unterschied auszugleichen, weist die erste Pflanze ein kürzeres Reproduktionsintervall auf als die zweite.

Der größte Unterschied zwischen den beiden Fressfeinden liegt im Energiebedarf für die Reproduktion sowie in der Energieaufnahme pro Zeitschritt. Während der

Feind der ersten Fressfeindart 20 Zeitschritte benötigt, um ein neues Individuum zu erzeugen, benötigen die Feinde der zweiten Art mehr Energie für die Reproduktion, nimmt jedoch gleichzeitig pro Zeitschritt mehr Energie auf.

In der folgenden Tabelle 21 sind die für die Simulation gewählten Parameter der Substanzen aufgelistet.

Parameter	Substance 1	Substance 2	Substance 3
Type	Signal	Toxin	Toxin
Deadly	-	Nein	Ja
Spread	Air	-	-
Name	s1	s2	s3
Production	p1,p2	p1	p2
Receiver	p1,p2	-	-
Trigger	e1,10;e2,30	s1,e1,20;s1,e2,40	s1,e1,40;s1,e2,15
Prod-Time	3	3	2
Send-Speed	10	-	-
Energy-Costs	1	1	1
AfterEffectTime	1	-	-
Eli-Strength	-	-	6

Tabelle 21: Gewählte Parameter der Substanzen für die Simulationsstudie

Die Pflanze der ersten Art verfolgt eine vertreibende Strategie, während die Pflanzen der zweiten Art eine tödliche Strategie anwenden. Zur Warnung andere Pflanzen wird ein Signalstoff über die Luft verbreitet, wobei sich der Wirkbereich alle 10 Zeitschritte vergrößert.

Für die initiale Belegung des Grids wurde eine vielfältige Mischung aus den beiden Pflanzenarten und den beiden Fressfeinden gewählt, um ein realistisches Biotop zu simulieren. Dabei wurden die Pflanzen in verschiedenen Konstellationen angeordnet: einmal benachbart zur selben Art und einmal benachbart zur jeweiligen anderen Pflanzenart.

Ähnlich verhält es sich mit den Fressfeinden, die gleichmäßig über das Grid verteilt wurden.

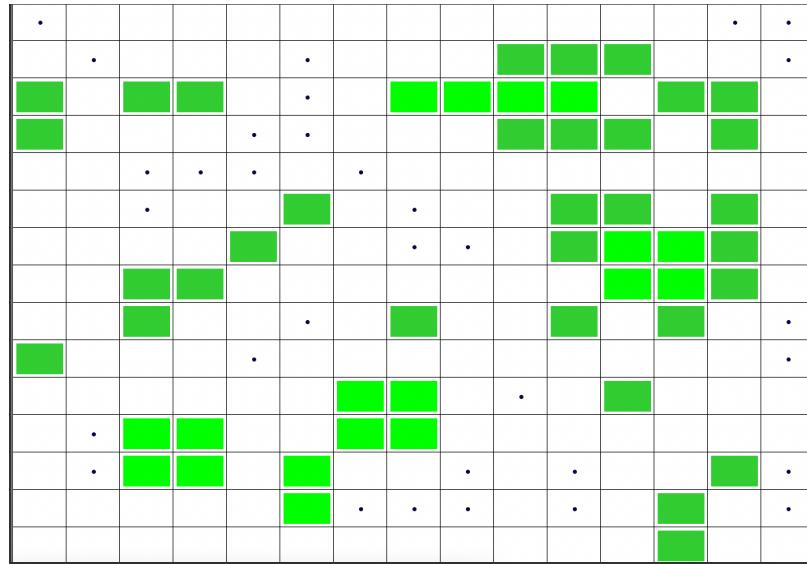


Abbildung 13: Initiale Belegung des Grids für die Simulationsstudie

5.2 Ergebnisse der Simulation

In diesem Kapitel werden die Ergebnisse der Simulationsstudie, basierend auf dem konfigurierten Szenario des vorherigen Kapitels, in Form der Plots präsentiert. Die Plots werden in zwei Abschnitten behandelt: Der erste Abschnitt befasst sich mit den Ergebnissen für die Pflanzen, während der zweite Abschnitt die Ergebnisse für die Fressfeinde präsentiert.

Da die Simulation mit zwei Pflanzenarten und zwei Fressfeindarten durchgeführt wurde, enthalten die Plots jeweils drei Zeitverlaufslinien. Die schwarze, gestrichelte Linie stellt dabei die Summe der jeweiligen gemessenen Werte dar.

5.2.1 Entwicklung der Pflanzenpopulation über die Zeit

Für die Population der Pflanzen wurde die Energie jeder Pflanzenart erfasst, um diese in einer zusammengefassten Zeitverlaufslinie für jede Art darzustellen. Hierfür wurde über alle Pflanzen der jeweiligen Art iteriert, und für jeden Zeitschritt die aktuell verfügbare Energie aufsummiert, um sie im Plot als eine gebündelte Linie zu visualisieren.

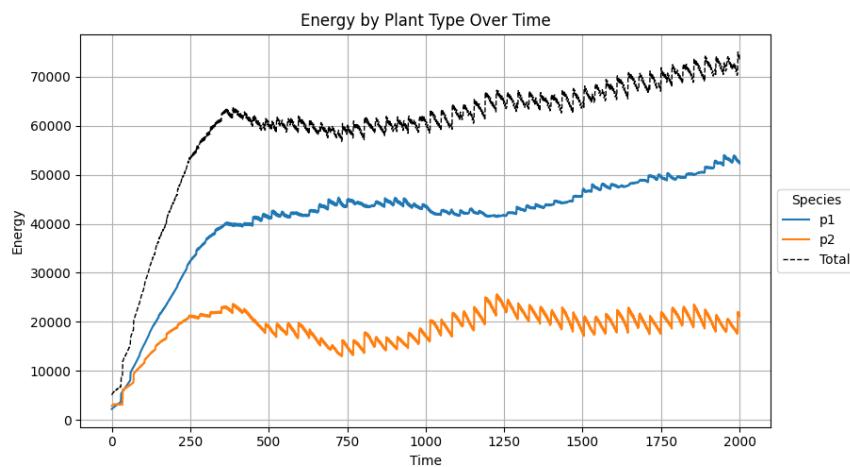


Abbildung 14: Zeitverlauf der Energie jeder Pflanzenart

Für den zweiten Plot wurde die Anzahl der Pflanzen einer Art zu jedem Zeitpunkt ermittelt. Dieser Plot zeigt zum einen, wie viele Pflanzen einer Art zu jedem Zeitschritt auf dem Grid vorhanden sind. Zum anderen wird auch hier die Summe aller auf dem Grid befindlichen Pflanzen in Form der schwarzen Linie dargestellt.

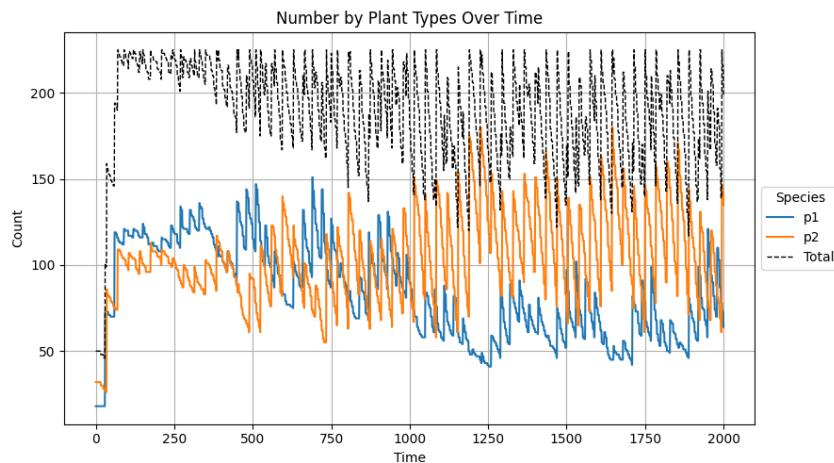


Abbildung 15: Zeitverlauf der Vorkommen jeder Pflanzenart

5.2.2 Entwicklung der Fressfeindpopulation über die Zeit

Ähnlich sind auch die Plots für die Fressfeinde aufgebaut. Der erste Plot stellt jedoch nicht die Energie dar, sondern zeigt die Anzahl der Individuen einer spezifischen Art über alle platzierten Cluster hinweg im Zeitverlauf.

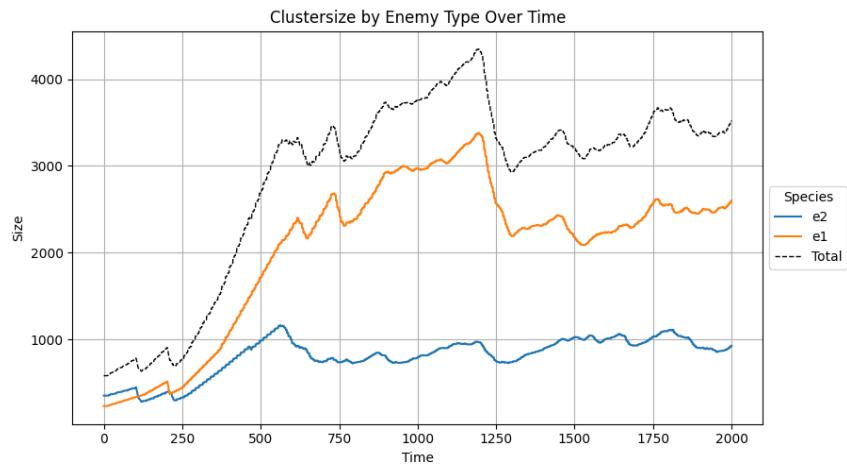


Abbildung 16: Zeitverlauf der Individuen jeder Fressfeindart

Ähnlich wie bei den Pflanzen, wo die Anzahl der Pflanzen einer spezifischen Art erfasst wurde, ist auch bei den Fressfeinden die Anzahl der Cluster einer spezifischen Art eine messbare Größe. Daher zeigt der vierte Plot die Anzahl der Cluster der verschiedenen Fressfeindarten im Zeitverlauf.

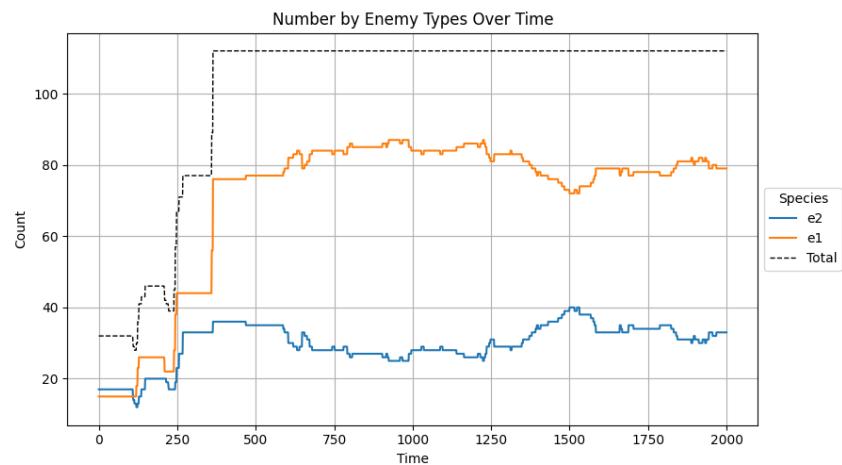


Abbildung 17: Zeitverlauf der Anzahl an Clustern für jede Fressfeindart

6 Diskussion und Schlussfolgerung

In diesem Kapitel werden die Simulationsergebnisse diskutiert, ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten gegeben.

6.1 Diskussion der Ergebnisse

Die Ergebnisse zeigen deutlich die dynamischen Veränderungen der Populationen von Pflanzen und Fressfeinden über die Zeit. Um aussagekräftige Erkenntnisse zu gewinnen, sollte die Simulation mindestens 1000 Schritte durchlaufen, da sich in allen vier Plots eine anfängliche Aufbauphase zeigt, in der sich das Modell ausrichtet. Die Dauer dieser Phase hängt von der initialen Konfiguration ab.

Eine längere Simulationsdauer führt erfahrungsgemäß zu noch eindeutigeren Ergebnissen. Zur besseren Vergleichbarkeit mit den in Kapitel 5 dargestellten Resultaten werden daher nachfolgend die Ergebnisse einer Simulation mit 4000 Zeitschritten bei identischen Anfangsbedingungen präsentiert.

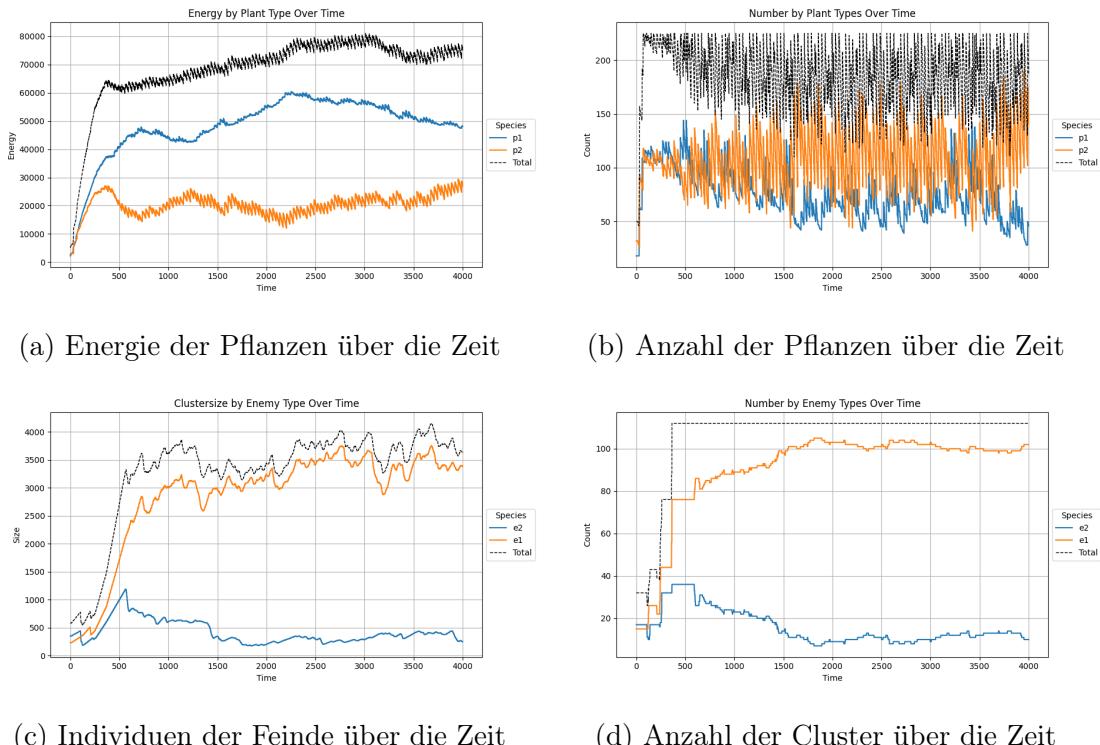


Abbildung 18: Simulation über 4000 Zeitschritte

In diesen Ergebnissen ist die Dynamik der Systementwicklung deutlich erkennbar, insbesondere im Plot 18a, der die Energie der Pflanzen darstellt. Hier zeigt sich eine wellenartige Dynamik, die darauf hindeutet, dass die Verfügbarkeit der Energie für die Pflanzen zeitlichen Schwankungen unterliegt.

Auch die Ergebnisse zur Anzahl der Pflanzen einer bestimmten Art zeigt ein dynamisches Verhalten in Form wellenartiger Verläufe. Diese Dynamik ist hier noch deutlicher erkennbar als bei den Ergebnissen zur Energiemenge der Pflanzen, wo sie ebenfalls vorhanden ist. Zudem lässt sich bei mehreren Pflanzenarten - in diesem Fall zwei - eine gegenläufige Dynamik beobachten. Dies bedeutet, dass eine Zunahme der Population einer Art mit einer Abnahme der Population der anderen Art einhergeht und umgekehrt.

Auch die Plots zur Anzahl der Individuen einer spezifischen Fressfeindart zeigen eine anfängliche Anpassungsphase, in der sich das Populationsniveau an die gegebenen Bedingungen angleicht. Anschließend oszilliert die Anzahl der Individuen in einer eher unregelmäßigen Dynamik um ein ungefähr konstantes Niveau. Diese Schwankungen hängen unter anderem davon ab, welche Pflanzen die Fressfeinde bevorzugen und wie das Biotop mit den Pflanzen besetzt ist. Zudem spielt ein gewisser Zufallsfaktor eine Rolle, da die Nachkommen der Pflanzen zufällig platziert werden und ihre Anzahl variiert.

Eine genauere Betrachtung der Plots 18b und 18c zeigt, dass bei etwa Zeitschritt 1300 ein lokales Minimum in der Anzahl der Individuen der Fressfeindart e1 auftritt, während die Anzahl der Pflanzen der Art p2 gleichzeitig ein lokales Maximum erreicht.

Die bisher diskutierten Ergebnisse zeigen ein Verhalten, das auch im Lotka-Volterra-Modell beobachtet werden kann. Dieses Modell besteht aus zwei Differentialgleichungen und beschreibt die Wechselwirkung zwischen Räuber und Beute in einem System [2].

In einer vereinfachten Darstellung wird die Populationsdynamik des Lotka-Volterra-Modells durch zwei Kurven repräsentiert: eine für die Räuber und eine für die Beute. Diese Kurven spiegeln die Populationsgröße im System über die Zeit.

Wird der Sachverhalt des Lotka-Volterra-Modells visuell in einem Plot dargestellt, so könnte dies wie folgt aussehen:

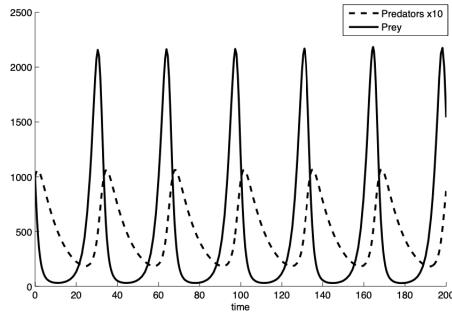


Abbildung 19: Vereinfachte Darstellung des Lotka-Volterra-Modells

Quelle: [34]

Um auf die Ergebnisse zurückzukommen: Auch die Simulationsergebnisse zeigen ein oszillierendes Verhalten zwischen den Pflanzen als Beute und den Fressfeinden als Räuber. Besonders in der Anfangsphase, in der sich die Simulation stabilisiert, lässt sich dieses Verhalten gut in den Plots zur Energie der Pflanzen und zur Anzahl der Individuen der Fressfeinde beobachten. In beiden Plots steigt die gemessene Größe zunächst an, bis eine kritische Anzahl an Individuen erreicht ist. Ab diesem Punkt beginnt die Energie der Pflanzen zu sinken.

Der generierte Plot zur Anzahl der Cluster mit den verschiedenen Fressfeindarten zeigt in der Gesamtheit nahezu konstant das Maximale der möglichen Clusteranzahl auf dem Grid. Dies lässt sich zum einen durch die Wahl der maximalen Clusteranzahl erklären, die während der Simulation durch Teilung entstehen können, aber auch durch die Wahl der verwendeten Parameter.

Andererseits ist in der Anzahl der einzelnen Cluster einer spezifischen Art, wie auch in den anderen Ergebnissen, ein leichtes periodisches Verhalten erkennbar. Dieses Verhalten deutet darauf hin, dass die Anzahl der Cluster einer bestimmten Art im Verlauf der Zeit sowohl ansteigt als auch wieder abnimmt. Gleichzeitig lässt sich erneut eine gegenläufige Dynamik zwischen den Clustern der verschiedenen Arten beobachten.

Ein weiterer Aspekt, der es wert ist, diskutiert zu werden, ist die Tatsache, dass der Verlauf einer Simulation stark von den gewählten Parametern abhängt. Es hat sich

gezeigt, dass schon kleine Veränderungen eines Parameters zu signifikanten Änderungen im Simulationsverlauf führen können.

Für einen besseren Vergleich wurde daher die exakt gleiche Konfiguration des Systems gewählt, wie sie im Kapitel zur Simulationsstudie verwendet wurde, hinsichtlich der Anzahl der Akteure, Substanzen und der Belegung des Grids. Kurz gesagt, das konfigurierte System wurde in das Tool importiert und einer kleinen Änderung unterzogen. Diese Änderung betrifft einen Wert im Input für den Trigger der tödlichen Substanz, die nun bei 35 Individuen für den Fressfeind e_2 ausgelöst wird, anstatt bei 15, wie es in der Simulationsstudie des vorangegangenen Kapitels der Fall war.

Die folgenden Ergebnisse wurden mit dieser kleinen Änderung generiert.

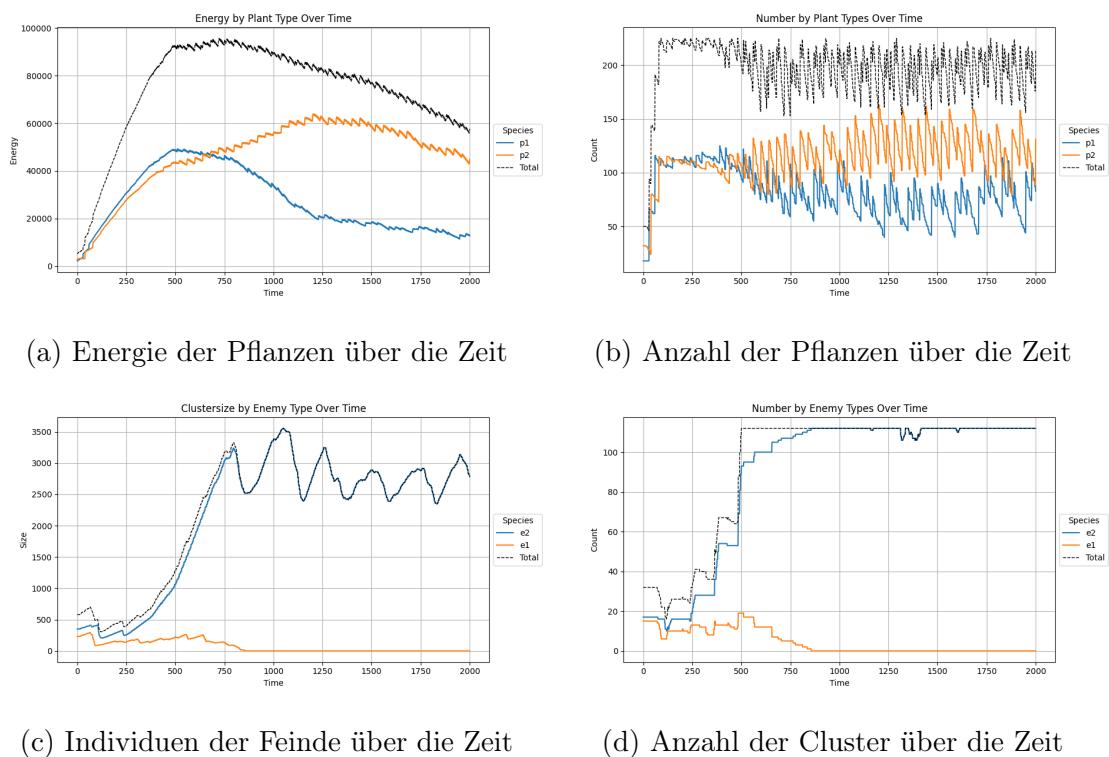


Abbildung 20: Simulation über 2000 Zeitschritte mit einer Parameterveränderung

Die Ergebnisse der kleinen Veränderung sind deutlich erkennbar. Durch die leichte Erhöhung der Individuenanzahl für die Fressfeindart e_2 kann sich die Fressfeindart e_1 nicht mehr langfristig halten und stirbt relativ schnell aus. Gleichzeitig etabliert sich die Fressfeindart e_2 und zeigt ein periodisches Verhalten, wie in dem Plot 20c gut zu erkennen ist.

Abschließend lässt sich festhalten, dass die Ergebnisse der Simulation stark von den gewählten Parametern abhängen. Bereits kleine Änderungen der Parameter können zu signifikant unterschiedlichen Ergebnissen führen, obwohl die Ausgangssituation nahezu identisch bleibt. Neben den initialen Parametern spielt auch der Zufall eine entscheidende Rolle im Verlauf der Simulation. Die Entscheidungen der Fressfeinde, welche Pflanze sie als nächstes ansteuern, könnte weitreichende Konsequenzen haben und im schlimmsten Fall zum Aussterben des Fressfeindclusters führen.

6.2 Schlussfolgerungen dieser Arbeit

Abschließend lässt sich festhalten, dass in dieser Arbeit eine umfassende Darstellung eines Modells zur Simulation pflanzlicher Schutzmechanismen gegen Fressfeinde präsentiert wurde. In Kapitel 1 wurde eine Einleitung gegeben, die die Sachlage im Hinblick auf drahtlose Sensornetzwerke und deren Ähnlichkeit zu pflanzlichen Schutzmechanismen erläutert.

In Kapitel 2 wurden die biologischen Grundlagen eines Biotops behandelt. Dabei wurden verschiedene Schutzmechanismen von Pflanzen erläutert, einschließlich ihrer Funktionsweise. Es ist wichtig zu betonen, dass dieser biologische Hintergrund lediglich einen ersten Einblick in dieses weitreichende und komplexe Thema bietet. Aufgrund der Komplexität des Themas wurde der Fokus auf die wesentlichen Aspekte der pflanzlichen Schutzmechanismen gelegt.

Im darauffolgenden Kapitel 3 wurde die Aufgabe behandelt, ein formales Modell zu entwickeln, das den biologischen Sachverhalt möglichst genau abbildet. Dabei wurden die verschiedenen, jedoch wesentlichen Bestandteile eines Biotops berücksichtigt, um diese später in einer Simulation zu integrieren. Konkret wurde für das formale Modell ein Grid für das Biotopt erstellt, auf dem die verschiedenen Akteure - in Form von Pflanzen und Fressfeinden - platziert werden können. Zudem wurde die Interaktion zwischen diesen Akteuren berücksichtigt, insbesondere was geschieht, wenn sich die Gegenspieler auf demselben Gridfeld befinden. Ein wichtiger Aspekt war die Formalisierung der Produktion verschiedener Substanzen, die einerseits zur Kommunikation zwischen Pflanzen in Form von Signalstoffen und andererseits zur Verteidigung gegen Fressfeinde durch Giftstoffe dienen.

Kapitel 4 befasst sich mit der Implementierung des formalisierten Modells. Dadurch wurden die verschiedenen Akteure und Bestandteile des Modells aufgegriffen und zur besseren Übersichtlichkeit in verschiedene Klassen unterteilt. Ein wesentlicher Aspekt der Implementierung war die Entwicklung einer intuitiven graphischen Oberfläche, die es ermöglicht, sowohl ein Netzwerk zu erstellen als auch die Simulation durchzuführen. Dadurch wird das Modell potenziell für jeden zugänglich.

Im folgenden Kapitel 5 wurde eine Simulationsstudie durchgeführt, um ein Ergebnis zu präsentieren, das durch das Tool generiert wird. Zunächst wurde auf die Konfiguration der verschiedenen Parameter eingegangen, die für eine Simulation von 2000 Zeitschritten verwendet wurden. Ein weiterer Bestandteil dieses Kapitel waren die Darstellung der Ergebnisse in Form verschiedener Plots.

In einem Unterkapitel des letzten Kapitels wurden die Ergebnisse der Simulation diskutiert, mit dem Lotka-Volterra-Modell verglichen und zusätzlich Ergebnisse einer Simulation mit längerer Simulationszeit sowie einer Simulation mit einer leichten Anpassung der Parameter präsentiert. Diese wurden verwendet, um die Empfindlichkeit der gewählten Parameter auf den Simulationsverlauf des Modells zu verdeutlichen. Um aus dem Modell Erkenntnisse gewinnen zu können, müssen im Rahmen weiterer Forschung zahlreiche Simulationen durchgeführt werden, um tiefere Einblicke in pflanzliche Schutzmechanismen zu erhalten. Gleichzeitig bietet das Modell Potenzial für Erweiterungen, um die Lücke zwischen der Realität und dem aktuellen Zustand des Modells zu verringern. Eine mögliche Erweiterung wäre die Einführung einer natürlichen Sterberate für die Fressfeinde, wie sie im Lotka-Volterra-Modell vorkommt. Diese könnte bspw. dann greifen, wenn ein Fressfeindcluster über einen bestimmten Zeitraum keine Nahrung aufgenommen hat. Ähnlich wie in der realen Welt, in der Fressfeinde nicht unbegrenzt ohne Nahrung überleben können. Derzeit sterben die Fressfeinde lediglich, wenn sie mit einem Giftstoff in Kontakt kommen. Innerhalb der graphischen Oberfläche könnten weitere Bedienelemente implementiert werden, die die Konfiguration der Simulation vereinfachen. Ein Beispiel wäre die Möglichkeit, Pflanzen und Fressfeindcluster vom Grid zu entfernen, was derzeit noch nicht realisiert ist. Dies könnte durch einen artspezifischen Button erfolgen, der bei Betätigung alle Objekte der entsprechenden Art vom Grid entfernt.

Eine weitere Idee zu Erweiterung der graphischen Oberfläche wäre die Einführung eines zusätzlichen Tabs für die Diagramme. In diesem Tab könnten wichtige Kenngrößen des Simulationsverlauf, wie das arithmetische Mittel, Minimum, Maximum oder andere relevante Werte, dargestellt werden, um die Analyse der Ergebnisse weiter zu vertiefen.

Zusammenfassend lässt sich festhalten, dass das Thema dieser Arbeit durch weitere Forschung, Verfeinerungen und Erweiterungen in Zukunft noch weiter vertieft werden kann, um ein noch umfassenderes Verständnis der pflanzlichen Schutzmechanismen und ihrer Simulation zu erlangen.

Literaturverzeichnis

- [1] Aljbory, Zainab and Chen, Ming-Shun. Indirect plant defense against insect herbivores: a review. *Insect Sci.*, 25(1):2–23, feb 2018.
- [2] Anisiu, Mira-Cristiana. Lotka, Volterra and their model. *Didáctica matemática*, 32(01), 2014.
- [3] Ari, Ado and Gueroui, Abdelhak and Labraoui, Nabila and Yenke, Blaise. Concepts And Evolution Of Research In The Field Of Wireless Sensor Networks. *International Journal of Computer Networks & Communications*, 7:81–98, 01 2015.
- [4] Babikova, Zdenka and Gilbert, Lucy and Bruce, Toby JA and Birkett, Michael and Caulfield, John C and Woodcock, Christine and Pickett, John A and Johnson, David. Underground signals carried through common mycelial networks warn neighbouring plants of aphid attack. *Ecology letters*, 16(7):835–843, 2013.
- [5] Bhattacharyya, Pranab and Jha, Dhruva. *Mycorrhizal Symbiosis in the formation of Antioxidant compounds*, pages 252–281. 01 2015.
- [6] Bruce, Toby JA. Interplay between insects and plants: dynamic and complex interactions that have coevolved over millions of years but act in milliseconds. *Journal of Experimental Botany*, 66(2):455–465, 2015.
- [7] Buratti, Chiara and Conti, Andrea and Dardari, Davide and Verdone, Roberto. An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors*, 9(9):6869–6896, 2009.
- [8] Chakraborty, Suman and Gershenson, Jonathan and Schuster, Stefan. Selection pressure by specialist and generalist insect herbivores leads to optimal constitutive plant defense. A mathematical model. *Ecology and Evolution*, 13, 12 2023.
- [9] Charles R. Harris and K. Jarrod Millman and Stéfan J. van der Walt and Ralf Gommers and Pauli Virtanen and David Cournapeau and Eric Wieser

-
- and Julian Taylor and Sebastian Berg and Nathaniel J. Smith and Robert Kern and Matti Picus and Stephan Hoyer and Marten H. van Kerkwijk and Matthew Brett and Allan Haldane and Jaime Fernández del Río and Mark Wiebe and Pearu Peterson and Pierre Gérard-Marchant and Kevin Sheppard and Tyler Reddy and Warren Weckesser and Hameer Abbasi and Christoph Gohlke and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [10] Chen, Ming-Shun. Inducible direct plant defense against insect herbivores: a review. *Insect science*, 15(2):101–114, 2008.
- [11] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford. *Introduction to Algorithms*. MIT Press, Cambridge, 2009.
- [12] Eustace, Sébastien and The Poetry contributors. Poetry: Python packaging and dependency management made easy. Computer software, 2024. Zugriff am 14. September 2024 <https://github.com/python-poetry/poetry>.
- [13] Farmer, Edward E. Surface-to-air signals. *Nature*, 411(6839):854–856, 2001.
- [14] Förster, Benjamin and Langendörfer, Peter and Hinze, Thomas. Novel Approach to a Plant Inspired Distributed Security Scheme for Wireless Sensor Networks. In *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, number , pages 1–6, 2023.
- [15] Förster, Benjamin and Langendörfer, Peter and Hinze, Thomas. Determining Distributions of Security Means for WSNs Based on the Model of a Neighborhood Watch. *IEEE Access*, page 74343–74366, 2024.
- [16] Gabora, Liane and Aerts, Diederik. Evolution as context-driven actualisation of potential: toward an interdisciplinary theory of change of state. *Interdisciplinary Science Reviews*, 30(1):69–88, March 2005.
- [17] GeoDZ. Mykorrhiza, 2024. Zugriff am 30. November 2024
<http://www.geodz.com/deu/d/Mykorrhiza>.

-
- [18] Gupta, C.P. and Kumar, Arun. Wireless Sensor Networks: A Review. *International Journal of Sensors Wireless Communications and Control*, 3:, 12 2013.
 - [19] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
 - [20] Jan Bruin and Marcel Dicke. Chemical information transfer between wounded and unwounded plants: backing up the future. *Biochemical Systematics and Ecology*, 29(10):1103–1113, 2001. Chemical information transfer between wounded and unwounded plants.
 - [21] Lev-Yadun, Simcha. *Aposematic (Warning) Coloration in Plants*, volume 2, pages 167–202. 03 2009.
 - [22] Lev-Yadun, Simcha. *Aposematic Coloration in Thorny, Spiny and Prickly Plants*, pages 119–134. Springer International Publishing, Cham, 2016.
 - [23] Lev-Yadun, Simcha. Testing of two of the visual anti-herbivory strategies of red young leaves: aposematic versus undermining herbivorous insect’s camouflage. *Arthropod-Plant Interactions*, 17:1–5, 06 2023.
 - [24] Meike Burow and Barbara Ann Halkier. How does a plant orchestrate defense in time and space? Using glucosinolates in Arabidopsis as case study. *Current Opinion in Plant Biology*, 38:142–147, 2017. 38 Biotic interactions 2017.
 - [25] Myburg, Alexander and Sederoff, Ronald. *Xylem Structure and Function*. 04 2001.
 - [26] Mélanie K. Rich and Nicolas Vigneron and Cyril Libourel and Jean Keller and Li Xue and Mohsen Hajheidari and Guru V. Radhakrishnan and Aurélie Le Ru and Seydina Issa Diop and Giacomo Potente and Elena Conti and Danny Duijsings and Aurélie Batut and Pauline Le Faouder and Kyoichi Kodama and Junko Kyozuka and Erika Sallet and Guillaume Bécard and Marta Rodriguez-Franco and Thomas Ott and Justine Bertrand-Michel and Giles E. D. Oldroyd

-
- and Péter Szövényi and Marcel Bucher and Pierre-Marc Delaux . Lipid exchanges drove the evolution of mutualism during plant terrestrialization. *Science*, 372(6544):864–868, 2021.
- [27] Müller-Esterl, Werner. *Biochemie - Eine Einführung für Mediziner und Naturwissenschaftler - Unter Mitarbeit von Ulrich Brandt, Oliver Anderka, Stefan Kerscher, Stefan Kieß und Katrin Ridinger*. Springer-Verlag, Berlin Heidelberg New York, edition, 2017.
- [28] Nourmohammad, Armita and Eksin, Ceyhun. Optimal evolutionary control for artificial selection on molecular phenotypes. *Physical Review X*, 11(1):011044, 2021.
- [29] Pieterse, Corné and Zamioudis, Christos and Berendsen, Roeland and Weller, David and van Wees, Saskia and Bakker, Peter. Induced Systemic Resistance by Beneficial Microbes. *Annual review of phytopathology*, 52:347–375, 09 2014.
- [30] Pocheville, Arnaud. The ecological niche: history and recent controversies. *Handbook of evolutionary thinking in the sciences*, pages 547–586, 2015.
- [31] Rojas, Cristian Antonio and Hemerly, Adrianna Silva and Ferreira, Paulo Cavalcanti Gomes. Genetically modified crops for biomass increase. Genes and strategies. *GM crops*, 1(3):137–142, 2010.
- [32] Schopfer, Peter and Brennicke, Axel. *Pflanzenphysiologie*. Springer-Verlag, Berlin Heidelberg New York, edition, 2016.
- [33] Sun, Tongjun and Zhang, Yuelin. Short-and long-distance signaling in plant defense. *The Plant Journal*, 105(2):505–517, 2021.
- [34] Weisberg, Michael and Reisman, Kenneth. The Robust Volterra Principle. *Philos. Sci.*, 75:, 01 2008.
- [35] Zander, C Dieter. Das Konzept der ökologischen Nische und seine Anwendung beim zoogeografischen Vergleich der Rifffische des Galapagos-Archipels. *Verhandlungen der Gesellschaft für Ichthyologie*, 5:231–246, 2006.

Abbildungsverzeichnis

1	Beispiele wie Warnfarben aussehen können	19
2	Die Formen der Mykorrhiza im Vergleich	26
3	Schematischer Ablauf einer Pflanzen-Insekten-Symbiose	28
4	Skizze der Architektur des Modells	68
5	Initiale Struktur der 2D-Liste <code>grid</code>	84
6	Beispiel für die Ausgabe des Grids in der Konsole	86
7	Elemente der Kontrollbar	101
8	Leeres Grid nach manueller Initialisierung mit einer Größe von 30x30	101
9	Tabs für Pflanzen, Fressfeinde und Substanzen	102
10	Fenster für die Abbruchbedingungen einer Simulation	103
11	Aufbau eines Felds im Grid	108
12	Beispiel eines befüllten Grids in der Simulation	110
13	Initiale Belegung des Grids für die Simulationsstudie	113
14	Zeitverlauf der Energie jeder Pflanzenart	114
15	Zeitverlauf der Vorkommen jeder Pflanzenart	114
16	Zeitverlauf der Individuen jeder Fressfeindart	115
17	Zeitverlauf der Anzahl an Clustern für jede Fressfeindart	115
18	Simulation über 4000 Zeitschritte	116
19	Vereinfachte Darstellung des Lotka-Volterra-Modells	118
20	Simulation über 2000 Zeitschritte mit einer Parameterveränderung .	119

Tabellenverzeichnis

1	Anwendungsbeispiel für das Erzeugen eines Signalstoffradius	52
2	Anwendungsbeispiele für die Ausbreitung über Gridverbindungen . . .	54
3	Attribute eines Pflanzen-Objekts mit Datentypen	69
4	Attribute eines Fressfeindcluster-Objekts mit Datentypen	73
5	Attribute eines Substanz-Objekts mit Datentypen	78
6	Attribute eines Signal-Objekts mit Datentypen	78
7	Attribute eines Toxin-Objekts mit Datentypen	80
8	Attribute eines SymbioticConnection-Objekts mit Datentypen . . .	82
9	Attribute und interne Attribute eines Grid-Objekts mit Datentypen .	83
10	Funktionen zum Hinzufügen und Entfernen von Objekten	85
11	Bedeutungen der Gridfeld-Formate in der Konsoleausgabe	87
12	Funktionsnamen für die Abbruchbedingungen und deren Beschreibung	96
13	Attribute eines Export-Objekts mit Datentypen	98
14	Bedeutung der Signalfarben der Zeitschrittanzeige in der Kontrollleiste	104
15	Bedeutung der in den Plots dargestellten Informationen über die Zeit	104
16	Erklärung der Eingabefelder für die Parameter der Pflanze	105
17	Erklärung der Eingabefelder für die Parameter der Fressfeinde	106
18	Eingabefelder für numerische Substanzparameter.	107
19	Bedeutung der Farben für die Substanzaktivität	108
20	Gewählte Parameter der Akteure für die Simulationsstudie	111
21	Gewählte Parameter der Substanzen für die Simulationsstudie	112

Quelltextverzeichnis

1	Wachstumsfunktion einer Pflanze	70
2	Überprüfung der Mindestenergie einer Pflanze	70
3	Erzeugung von Nachkommen einer sich reproduzierenden Pflanze . .	71
4	Positionsbestimmung der pflanzlichen Nachkommen	72
5	Auswahl einer zufälligen nächstgelegenen Pflanze	74
6	Reproduktion von Fressfeinden	77
7	Funktion zum Vertreiben von Fressfeinden	81
8	Verarbeitung der Bewegung der Fressfeindcluster auf dem Grid	88
9	Verarbeitung der Interaktion zwischen Pflanze und Fressfeindcluster .	89
10	Funktion zur Verarbeitung der Alarmierung einer Pflanze	90
11	Funktion zur Verarbeitung der Produktion eines Signalstoffs	91
12	Funktion zur Verarbeitung von Signalstoffeffekten	92
13	Funktion für die Simulation eines Zeitschritts für die Pflanzen	95
14	Ausschnitt der <code>run</code> -Funktion aus der Klasse <code>Simulation</code>	96
15	Hilfsfunktion zum Sammeln von Informationen für den Export	98
16	Funktion zum Rekonstruieren des Netzwerks	99

Anlagen

Algorithmus 1 Kürzester Pfad zu einer Pflanze

Eingabe: Startpunkt σ , Zielpunkt ρ

Ausgabe: Kürzester Pfad von σ zu ρ oder \emptyset falls kein Pfad existiert

```
1:  $q \leftarrow [\sigma]$                                 ▷ Initialisiere Warteschlange mit Startpunkt
2:  $\Psi(\sigma) \leftarrow 0$                             ▷ Distanz vom Startpunkt
3:  $\Pi(\sigma) \leftarrow \emptyset$                       ▷ Vorgänger des Startpunkts
4: while  $q \neq \emptyset$  do
5:    $\psi \leftarrow q.\text{dequeue}()$           ▷ Entferne vordersten Punkt aus der Warteschlange
6:   if  $\psi = \rho$  then
7:     return RekonstruierePfad( $\Pi, \rho$ ) ▷ Ziel erreicht, kürzester Pfad gefunden
8:   end if
9:    $\Gamma' \leftarrow \text{dynPrior}(\psi, \Gamma, \rho)$       ▷ Dynamische Priorisierung
10:  for  $\gamma \in \Gamma'$  do
11:     $\mu \leftarrow (x_\psi + \gamma_0, y_\psi + \gamma_1)$         ▷ Berechne neue Position
12:    if  $\mu \in G$  und  $\mu \notin \Psi$  then          ▷ Prüfe, ob gültig und nicht besucht
13:       $\Psi(\mu) \leftarrow \Psi(\psi) + 1$                 ▷ Aktualisiere Distanz
14:       $\Pi(\mu) \leftarrow \psi$                           ▷ Speichere Vorgänger
15:       $q.\text{enqueue}(\mu)$                          ▷ Füge neue Position zur Warteschlange hinzu
16:    end if
17:  end for
18: end while
19: return  $\emptyset$                                 ▷ Kein Pfad gefunden
```

Bereitstellung des Quelltextes:

Aufgrund des Umfangs des Quelltextes wird an dieser Stelle auf eine vollständige Auflistung verzichtet. Stattdessen kann der Quelltext unter den folgenden Links eingesehen werden:

GitHub: <https://github.com/lucas-diet/PlantProtectionSim>

GitLab: <https://git.uni-jena.de/xe43nok/PlantProtectionSim>



FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA Fakultät für Biowissenschaften

PD Dr.-Ing. habil. Thomas Hinze

Universität Jena · BioWi · 07737 Jena

Matthias-Schleiden-Institut
Lehrstuhl Bioinformatik
Ernst-Abbe-Platz 2
07743 Jena

Telefon: 0 172 79 79 603
Telefax: 0 36 41 9-464 52
E-Mail: thomas.hinze@uni-jena.de

Jena, den 20. August 2024

Aufgabenstellung für die Masterarbeit von Lucas Dietrich

Arbeitstitel

Ein Simulationssystem für die Wirkung von Schutzmechanismen von Pflanzen auf Fressfeinde – Visualisierter Wettbewerb zwischen konfigurierbaren Akteuren in kooperierenden und feindlichen Teams

Betreuende

Themenverantwortlicher: PD Dr. Thomas Hinze thomas.hinze@uni-jena.de
Zweitgutachter: Benjamin Förster, M.Sc. bfoerster@ihp-microelectronics.com

Hintergrund

Im Gegensatz zu Tieren können Pflanzen vor Fressfeinden nicht fliehen, sie sind an ihren Standort gebunden, haben aber Einfluss darauf, wo sich ihre Nachkommen in der Umgebung ansiedeln können. Betrachtet man großflächige natürlich gewachsene Habitate wie Urwälder, dann fällt auf, dass sich Pflanzen gleicher Art in gewissen Abständen zueinander in der Landschaft verteilen. Viele verschiedene Pflanzenarten leben in Nachbarschaft zueinander. Zwischen ihnen können symbiotische Beziehungen entstehen, indem zum Beispiel Nährstoffe ausgetauscht werden, aber es gibt auch Konkurrenz um Ressourcen wie Licht und Wasser. Komplizierte Netzwerke von Pflanzen verschiedener Arten sind auf der Erdoberfläche entstanden, hinter deren konkreter Ausprägung die Lösung komplexer Optimierungsprobleme steckt. Diese Netzwerke entwickeln sich über die Zeit immer weiter.

Pflanzen sind mannigfaltigen Fressfeinden ausgesetzt. Um ihr eigenes Überleben und das Überleben ihrer Art zu sichern, haben sie eine Vielzahl unterschiedlicher wirkungsvoller Schutzmechanismen hervorgebracht, mit denen sie Fressfeinde stoppen, eliminieren und auf andere Ziele lenken können.

Betrachtet man von außen in der Gesamtschau alle Akteure wie Pflanzen und Fressfeinde sowie die zwischen ihnen stattfindenden Wechselwirkungen, so entsteht ein System, in welchem über die Zeit zahlreiche Spieler mit- und gegeneinander agieren. Jeder von ihnen verfügt über eine Palette an Strategien und biologischer Ausstattung, die gezielt zum Einsatz gebracht werden. Das Bereithalten eines solchen Arsenals und seine jederzeitige Aktivierbarkeit kostet Energie, die die Pflanzen investieren müssen. Manche Schutzmechanismen lassen sich mit wenig Energieeinsatz ausbilden wie zum Beispiel scharf gezackte Blätter. Andere Schutzmechanismen wie spezielle, in den Blättern eingelagerte

Giftstoffe und das Warnen anderer artgleicher Pflanzen in der Umgebung über Signalstoffe konsumieren mehr Energie. Zudem benötigen Schutzmechanismen unterschiedlich viel Zeit, bis sie wirken und sich dieser Effekt in der Pflanze und ihrer Umgebung ausbreitet. Die Angreifer wiederum müssen sich auf Schutzmechanismen einstellen und zum Beispiel „gegen den Wind“ fressen, wie es bei Giraffen bereits beobachtet wurde.

Die Kommunikation zwischen Pflanzen, um sich gegenseitig zu warnen, erfolgt entweder mithilfe kleiner Signalmoleküle, die von Blättern freigesetzt werden und sich im Luftstrom (Wind) zu anderen Pflanzen bewegen. Ethylen (Ethen) ist hierfür ein häufig genutzter Botenstoff neben einigen anderen. Insbesondere bei Bäumen spielt aber auch ein ausgedehntes Wurzelgeflecht eine Rolle, das zudem mit dem Myzel symbiotischer Pilze im Erdboden verknüpft sein kann. Über dieses Wurzel- und Myzelgeflecht lassen sich Botenstoffe unabhängig von Wind und Windrichtung transportieren, wofür auch komplexere Signalmoleküle zum Einsatz kommen, die sich nicht ohne weiteres durch die Luft bewegen könnten.

Wurden Pflanzen oder Bäume vor Fressfeinden gewarnt, steht ihnen ein Arsenal verschiedener Schutzmechanismen zur Verfügung. Je nach eingesetztem Botenstoff oder der Botenstoffkombination bei der Warnung kann ein spezifischer Schutzmechanismus aktiviert werden, sobald ein Schwellwert erreicht ist. Am häufigsten wird das schnelle Produzieren und Einlagern von Gift- oder Bitterstoffen in die Blätter genutzt. Oft reicht das schon, um Fressfeinde wie größere Tiere zu vertreiben. Manche Insekten oder schmarotzende Schädlinge sind jedoch hartnäckiger. Mitunter haben sie sogar schon Resistenzen gegen die eingesetzten Gift- oder Bitterstoffe entwickelt. Dann stehen weitere Sicherheitsmechanismen zur Verfügung wie die Verfärbung von Blättern. Hierbei werden das Chlorophyll und für die Fressfeinde attraktive Nährstoffe aus den Blättern herausgezogen und eine Zeitlang im Hartholz zwischengespeichert. Die Blätter verfärben sich dann und wirken teilweise wie abgestorben, so dass die Fressfeinde von der Pflanze ablassen. Ein besonders eleganter Schutzmechanismus setzt auf Kooperation betroffener Pflanzen mit anderen Tieren. Getreu der Erkenntnis „Die Feinde meiner Feinde sind meine Freunde“ locken sie gezielt mit Duftstoffen Feinde der Fressfeinde an. Dies funktioniert am besten, wenn Insekten die Fressfeinde sind. Deren Feinde (Vögel, andere größere Insekten, kleine Säugetiere) sind solchen Duftstoffen gegenüber sehr sensibel und werden mit leichter Beute, nämlich den eigentlichen Fressfeinden der Pflanze, belohnt.

Ziele

Ziel der Masterarbeit soll es sein, Szenarien aus Pflanzen, ihrer Verteilung in der Fläche, ihren Schutzmechanismen sowie deren Wirkung auf angreifende Fressfeinde abstrakt visuell zu konfigurieren, zu modellieren und schließlich das Verhalten des Gesamtsystems über die Zeit zu simulieren sowie diesen dynamischen Prozess grafisch darzustellen. Ergänzt wird die Simulation durch die Erfassung und Analyse von Begleitdaten, zum Beispiel den Energieverbrauch über die Zeit und die Populationsdynamik aller im Gesamtsystem vorhandenen Tier- und Pflanzenarten.

Der Nutzer kann sich mit dem Simulationssystem relativ frei Szenarien aus Pflanzen, ihrer Verteilung und Interaktion sowie verfügbaren Schutzmechanismen konfigurieren, ebenso die Fressfeinde mit ihren Fähigkeiten und Eigenschaften. In gewisser Weise ähnelt das Simulationssystem einem Computerspiel, bei dem zunächst eine Vielzahl von Akteuren spezifiziert und konfiguriert wird und dann all diese Spieler aufeinandertreffen und miteinander interagieren.

Mit dem Simulationssystem werden einige Fallstudien als Demonstrationsbeispiele durchgeführt und dokumentiert. Die Fallstudien widmen sich verschiedenen Optimierungsszenarien. Hier ist von besonderem Interesse, wie Pflanzen mit möglichst wenig aufgewandelter Energie einen möglichst starken

Angriff von Fressfeinden abwehren können. Ebenfalls von herausgehobenem Interesse sind Demonstrationsbeispiele, die sehr langlebige Systeme modellieren, die eine gute Regenerationsfähigkeit aufweisen und auf diese Weise Angriffswellen standhalten und sich an diese anpassen können.

Vorgeschlagene Festlegungen und Anforderungen

Das Simulationssystem soll eine grafische Oberfläche besitzen und möglichst intuitiv und leicht zu bedienen sein, indem die Nutzer durch gut strukturierte Dialogboxen geführt werden, um das Gesamtsystem schrittweise zu konfigurieren. Anschließend wird es visuell erlebbar in seinem Verhalten simuliert und danach erfolgt eine statistische Auswertung. Jedes definierte Gesamtsystem muss sich als Datei abspeichern und bei Bedarf wieder einlesen lassen einschließlich aller vorgenommenen Konfigurationen, numerischen Simulations- und statistischen Auswertungsdaten. Für die Implementierung wird die Programmiersprache *Python* favorisiert. Im Hinblick auf eine klare Trennung zwischen Datenverwaltung, Anwendungskern und Oberfläche soll die *Modell-View-Controller* Architektur verwendet werden, die klar definierte Schnittstellen zwischen der graphischen Oberfläche und dem Controller definiert, die dessen reibungslosen Austausch zulassen. Ergänzend soll die virtuelle Umgebung *Poetry* (<https://python-poetry.org/>) eingesetzt werden. Poetry ist ein modernes Werkzeug zur Verwaltung von Abhängigkeiten in Python. Es vereinfacht die Verwaltung von Projekten, Abhängigkeiten und virtuellen Umgebungen und verwendet eine deklarative Konfigurationsdatei. Darüber hinaus sollen die Quelltexte begleitend zur Implementierung im Repository *Git* bereithalten werden, so dass der Entwicklungsfortschritt dokumentiert ist, bei Bedarf zu vorherigen Versionen zurückgekehrt werden kann und externe Sicherungskopien der gesamten Implementierung vorliegen. Ein zweites, privat gehaltenes Git-Repository für die LaTeX-Quellen der Masterarbeit wird empfohlen.

Alle konfigurierbaren Systembestandteile werden abstrakt durch ihre Attribute erfasst. Als ebene Landschaft dient ein rechteckiger *Grid* aus $m \times n$ Einheitsquadraten, nachfolgend Gridfelder genannt (Richtwerte: $m, n \leq 80$).

Jedes Feld kann durch höchstens eine *Pflanze* belegt werden. Der Nutzer wählt, wieviele Pflanzenarten (mindestens eine, höchstens 16) unterschieden werden sollen. Für jede Pflanzenart wird eine Farbe festgelegt. Eine Pflanze wird auf ein Feld gesetzt, indem der Nutzer dieses Feld mit der entsprechenden Farbe markiert.

Jeder einzelnen *Pflanze* lässt sich eine anfängliche Anzahl Energiedinheiten zuweisen, über die sie verfügt. Bei Pflanzen gleicher Art darf dieser initiale Wert variieren, weil sie zum Beispiel unterschiedlich hoch oder unterschiedlich alt sind und sich ihre Gesamtblattfläche unterscheidet. Darüber hinaus wird konfiguriert, um wieviele Prozent der bereits vorhandenen Energie die Energie einer Pflanze pro diskretem Simulationsschritt zunehmen soll. Zudem gibt es für jede individuelle Pflanze eine Nutzervorgabe, welche Mindestanzahl Energiedinheiten zum Überleben notwendig ist. Unterschreitet die Anzahl Energiedinheiten einer Pflanze während der Simulation diesen Wert, so wird die betreffende Pflanze vom Grid gelöscht. Pflanzen können sich mit der Zeit vermehren. Hierzu lässt sich vorgeben, ob und nach welcher Zeit (Anzahl Simulationsschritte) eine individuelle Pflanze wieviele Nachkommen gleicher Art hervorbringt. Jeder Nachkomme verfügt wiederum über eine vorgebbare anfängliche Anzahl Energiedinheiten und wird per Zufall in einem freien Gridfeld in einer zufällig gewählten Entfernung (unter Beachtung einer Min-Max-Vorgabe) und Richtung zur Mutterpflanze platziert. Ist kein freies Gridfeld unter den eingestellten Bedingungen verfügbar, so wird der Nachkomme nicht erzeugt. Während der Simulation wird auf jedem Gridfeld, das durch eine Pflanze belegt ist, als Zahl oder in anderer geeigneter Weise angezeigt, wieviel Prozent Energie die Pflanze in Bezug auf ihre anfängliche Anzahl Energiedinheiten aktuell hat. Dieser Wert kann sowohl über als auch unter 100% liegen. Pflanzen, die sich auf unmittelbar benachbarten Gridfeldern befinden, können sich individuell miteinander

symbiotisch verbinden. Dies ist auch dann möglich, wenn es sich um verschiedene Arten handelt. Der Nutzer klickt dazu auf die Begrenzungslinie der beiden Gridfelder, was grafisch durch eine farblich erkennbare Verbindung zwischen diesen Gridfeldern angezeigt wird. Über diese Verbindungen können sich während der Simulation Signalstoffe verbreiten.

Die Konfiguration der *Fressfeinde* beginnt ebenfalls mit der Festlegung, wieviele Arten es davon geben soll (0 bis 15). Fressfeinde können in Schwärmen oder einzeln agieren. Um dies abzubilden, führen wir den Begriff „Fressfeindcluster“ ein. Ein Fressfeindcluster besteht aus einer initial frei konfigurierbaren Anzahl von Individuen (mindestens 1). Alle Individuen in einem solchen Cluster gehören der gleichen Fressfeindart an und befinden sich zu jeder Zeit gemeinsam im gleichen Gridfeld. Jedes definierte Fressfeindcluster kann vom Nutzer frei auf ein initiales Gridfeld gesetzt werden, unabhängig davon, ob dort eine Pflanze platziert ist und welcher Art sie angehört. Visuell wird dies durch ein Symbol je nach Fressfeindart (Kreis, Dreieck, ... in verschiedenen, nicht für Pflanzen genutzten Farben) markiert und die Anzahl Individuen als Zahlenwert oder anderweitig im Symbol dargestellt. Es ist zulässig, dass sich auf demselben Gridfeld mehrere Fressfeindcluster befinden. Fressfeindcluster auf Gridfeldern, auf denen sich keine Pflanze befindet, bewegen sich in Richtung der nächstgelegenen Pflanze. Gibt es mehrere Pflanzen, die gleich nah entfernt sind, wird eine davon per Zufall ausgewählt. Der Nutzer gibt für jedes Fressfeindcluster vor, wieviele Zeitschritte der Simulation vergehen sollen, bis das Fressfeindcluster ein benachbartes Gridfeld erreicht hat. Die Bewegungen von Fressfeindclustern erfolgen entlang gedachter Geraden über den Grid und werden in der zeitdiskreten Darstellung auf die jeweils von der Gerade erfassten Gridfelder heruntergebrochen.

Als weiteres Element des Gesamtsystems erfolgt die Konfiguration von *Substanzen*, also Signal- bzw. Giftstoffen. Es können zwischen 0 und 15 verschiedene Substanzen definiert werden. Für jede Substanz kann ein individueller Name vergeben werden, und es muss vom Nutzer zugeordnet werden, ob es ein Signal- oder ein Giftstoff ist.

Nun wird die Konfiguration der einzelnen Wechselwirkungen zwischen Pflanzen, Fressfeindclustern und Substanzen vorgenommen. Hierfür finden *Interaktionsmatrizen* Verwendung, die der Nutzer im Vorfeld der Simulation ausfüllt und die sich vorteilhaft durch geometrische Graphen abbilden lassen.

Beginnen wir mit der Interaktionsmatrix der Signalstoffe. Für jeden Signalstoff muss angebar sein, welche Pflanzenarten ihn emittieren und welche Pflanzenarten für ihn empfänglich sind. Zusätzlich muss festgelegt werden, welche Kombinationen von Fressfeindarten das Freisetzen des Signalstoffs auslösen und wieviele Individuen die einbezogenen Cluster dazu mindestens enthalten müssen. Für jeden Signalstoff lässt sich einstellen, ob er sich entweder durch die Luft oder über Gridfeldverbindungen ausbreitet. Die Ausbreitung durch die Luft geschieht in immer größer werdenden Kreisen um die auslösende Pflanze herum. Als Parameter für die Ausbreitungsgeschwindigkeit wird festgelegt, nach wievielen Zeitschritten der Radius dieses Kreises um eine weitere Gridfeldlänge zugenommen hat. Signalstoffe, die sich über Gridfeldverbindungen ausbreiten, benötigen ebenfalls einen Parameter, der darüber Auskunft gibt, nach wievielen Zeitschritten die jeweils benachbarten Gridfelder erreicht sind. Die Produktion von Signalstoffen kostet Energie, die von der Pflanze aufgewendet werden muss. Folglich gibt es für jeden Signalstoff einen konfigurierbaren Wert, der aussagt, wieviele Energienheiten pro Zeitschritt vom Energielevel der Pflanze dafür abgezogen werden. Ein und dieselbe Pflanze kann gleichzeitig verschiedene Signalstoffe freisetzen. Das Vorhandensein von Signalstoffen wird durch verschiedene Musterungen und ihre Überlagerung in den betroffenen Gridfeldern dargestellt. Die Produktion eines Signalstoffes endet, sobald die betreffende Pflanze nicht mehr von Fressfeindarten attackiert wird, die zu seiner Auslösung notwendig sind. Für jeden Signalstoff kann definiert werden, wieviele Zeitschritte er noch nachwirkt. Nach Ablauf dieser Zeitspanne wird der gesamte bisher ent-

standene Wirkkreis bzw. Wirkbereich des Signalstoffes, der von der auslösenden Pflanze ausging, entfernt. Möglicherweise überlappende Wirkkreise bzw. Wirkbereiche des gleichen Signalstoffes, den andere Pflanzen freigesetzt haben, bleiben davon unberührt.

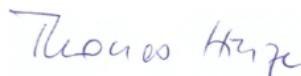
In ähnlicher Weise werden nun die Eigenschaften und Interaktionen der Giftstoffe festgelegt. Für jeden Giftstoff erfolgt wiederum die Auswahl, welche Pflanzenarten ihn produzieren und wieviel Energie die jeweilige Pflanze dafür pro Zeitschritt aufwenden muss. Für jeden Giftstoff wird individuell in einer Matrix festgelegt, welche Kombination aus Signalstoffen und/oder Fressfeindarten mit welcher jeweiligen Mindestfressfeindclusterstärke (Anzahl Individuen) seine Produktion auslösen und wieviele Zeitschritte diese Produktion erfordert, bis der Giftstoff seine Wirkung an der Pflanze entfaltet. Wir nehmen an, dass das Vorhandensein eines Giftstoffes definierbar entweder Individuen von Fressfeinden abtötet oder das Fressfeindcluster von sich weglenkt. Der Nutzer kann für jeden Giftstoff konfigurieren, welche Fressfeindarten vom Giftstoff attackiert werden und wieviele Individuen eines Clusters pro Zeitschritt eliminiert werden bei abtötender Wirkung. Bei weglenkender Wirkung wird die betreffende Pflanze aus der Wahrnehmung der attackierenden und dafür empfänglichen Fressfeindcluster entfernt, so dass sich diese zur nächstgelegenen Pflanze weiterbewegen, ohne dass Individuen entfernt werden. Das Vorhandensein eines Giftstoffs wird auf dem Gridfeld der produzierenden Pflanze visualisiert. Verschiedene Giftstoffe können wiederum durch individuelle Farbmuster dargestellt werden. Die Produktion eines Giftstoffes endet, sobald die betreffende Pflanze nicht mehr durch Fressfeinde angegriffen wird, die auf den betrachteten Giftstoff ansprechen. Wir nehmen vereinfachend an, dass Giftstoffe nicht nachwirken. Endet ihre Produktion, endet unmittelbar auch ihre Wirkung, und die entsprechende Pflanze ist dann frei davon, aber auch wieder mögliches Ziel zuvor weggelenkter Fressfeindcluster.

Zur vollständigen Konfiguration der Interaktionen gehören nun noch die „Fresserfolge“ der Fressfeindcluster. Für jede Fressfeindart lässt sich festlegen, wieviele aus verzehrten Pflanzen aufgenommene Energieeinheiten nötig sind, um ein neues, zusätzliches Individuum im jeweiligen Cluster hervorzu bringen.

Für den Simulationslauf kann die Zeitspanne für einen Zeitschritt eingestellt werden. Ferner lässt sich eine Auswahl aus mehreren Optionen vornehmen, um die Dauer bzw. das Ende des Simulationslaufs zu spezifizieren. Hier kann man entweder eine feste Anzahl Zeitschritte angeben oder eine der folgenden Bedingungen: (1) Aussterben einer bestimmten, wählbaren Pflanzenart (2) Aussterben aller Pflanzenarten (3) Aussterben einer bestimmten, wählbaren Fressfeindart (4) Aussterben aller Fressfeindarten (5) Erreichen einer Grenze für die Gesamtenergie aller Pflanzen (6) Erreichen einer Grenze für die Gesamtzahl Individuen aller Fressfeinde.

Die Auswertung soll in der Granularität Zeitschritt während der Simulation mitprotokollieren, wieviele Pflanzen welcher Arten mit wieviel Gesamtenergie, wieviele Fressfeindcluster welcher Arten mit wievielen Individuen sich im System befinden und in einem oder mehreren Diagrammen in geeigneter Weise darstellen.

Mit freundlichem Gruß



PD Dr.-Ing. habil. Thomas Hinze



Eigenständigkeitserklärung

1. Hiermit versichere ich, dass ich die vorliegende Arbeit - bei einer Gruppenarbeit die von mir zu verantwortenden und entsprechend gekennzeichneten Teile - selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich trage die Verantwortung für die Qualität des Textes sowie die Auswahl aller Inhalte und habe sichergestellt, dass Informationen und Argumente mit geeigneten wissenschaftlichen Quellen belegt bzw. gestützt werden. Die aus fremden oder auch eigenen, älteren Quellen wörtlich oder sinngemäß übernommenen Textstellen, Gedankengänge, Konzepte, Grafiken etc. in meinen Ausführungen habe ich als solche eindeutig gekennzeichnet und mit vollständigen Verweisen auf die jeweilige Quelle versehen. Alle weiteren Inhalte dieser Arbeit ohne entsprechende Verweise stammen im urheberrechtlichen Sinn von mir.
2. Ich weiß, dass meine Eigenständigkeitserklärung sich auch auf nicht zitierfähige, generierende KI-Anwendungen (nachfolgend „generierende KI“) bezieht.
Mir ist bewusst, dass die Verwendung von generierender KI unzulässig ist, sofern nicht deren Nutzung von der prüfenden Person ausdrücklich freigegeben wurde (Freigabeerklärung). Sofern eine Zulassung als Hilfsmittel erfolgt ist, versichere ich, dass ich mich generierender KI lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss deutlich überwiegt. Ich verantworte die Übernahme der von mir verwendeten maschinell generierten Passagen in meiner Arbeit vollumfänglich selbst.
Für den Fall der Freigabe der Verwendung von generierender KI für die Erstellung der vorliegenden Arbeit wird eine Verwendung in einem gesonderten Anhang meiner Arbeit kenntlich gemacht.
Dieser Anhang enthält eine Angabe oder eine detaillierte Dokumentation über die Verwendung generierender KI gemäß den Vorgaben in der Freigabeerklärung der prüfenden Person.
Die Details zum Gebrauch generierender KI bei der Erstellung der vorliegenden Arbeit inklusive Art, Ziel und Umfang der Verwendung sowie die Art der Nachweispflicht habe ich der Freigabeerklärung der prüfenden Person entnommen.
3. Ich versichere des Weiteren, dass die vorliegende Arbeit bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt wurde oder in deutscher oder einer anderen Sprache als Veröffentlichung erschienen ist.
4. Mir ist bekannt, dass ein Verstoß gegen die vorbenannten Punkte prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass meine Prüfungsleistung als Täuschung und damit als mit „nicht bestanden“ bewertet werden kann. Bei mehrfachem oder schwerwiegendem Täuschungsversuch kann ich befristet oder sogar dauerhaft von der Erbringung weiterer Prüfungsleistungen in meinem Studiengang ausgeschlossen werden.

Jena, 10.03.2025

Ort und Datum

L. Dietrich

Unterschrift



Freigabeerklärungen

Von der prüfenden Person konkret festzulegender Geltungsbereich
(z.B. Thema, Veranstaltung, Zeitraum):

Masterarbeit Bioinformatik für Lucas Dietrich

Die folgenden Tools müssen nicht als Hilfsmittel deklariert werden und dürfen zur Erstellung von Seminar- und Abschlussarbeiten genutzt werden, auch wenn sie KI-gestützt sind:

- Textverarbeitungsprogramme, z. B. Word oder OpenOffice Writer
- Tabellenkalkulation, z. B. Excel oder LibreOffice Calc
- Rechtschreib- und Grammatikprüfung sowie -korrektur inkl. Werkzeugen in Textverarbeitungsprogrammen, z. B. DeepKomma
- Suchmaschinen
- Digitale Wörterbücher und Thesaurus
- Mindmap-Tools
- Recherchetools, z. B. wissenschaftliche Literatursuche via PubMed
- Recherchetools, die keine Ideen generieren, z. B. wissenschaftliche Literatursuche via Google Scholar
- eigene Erweiterungen...

Seminar- und Abschlussarbeiten sollen zuvorderst die Gedanken, Ideen und Erkenntnisse der Verfasserin bzw. des Verfassers beinhalten. Es muss klar erkennbar sein, ob und an welchen Stellen diese durch generierende KI ergänzt wurden. Im Folgenden finden Sie eine Auflistung der für die betroffene Prüfung erlaubten KI-Werkzeuge und wie deren Einsatz zu kennzeichnen ist. Die Dokumentation soll in einem separaten KI-Quellenverzeichnis erfolgen.

Erlaubte Werkzeuge:

- Textgenerierende** KI-Werkzeuge: Die wörtliche oder inhaltliche Übernahme aus KI-generierten Textquellen (einschließlich Quellcodes, mathematischen Ausdrücken etc.) ist erlaubt.
- Bildgenerierende** KI-Werkzeuge: Die direkte Übernahme aus KI-generierten Bildquellen ist erlaubt.
- Bildverarbeitende** KI-Werkzeuge: Die direkte Übernahme aus Bildquellen, die mittels KI-Werkzeug weiterverarbeitet werden, ist erlaubt.
- Übersetzung** durch KI-Werkzeuge: Die wörtliche Übernahme aus KI-generierten Übersetzungen ist erlaubt.
- eigene Erweiterungen...

Zur Dokumentation des KI-Einsatzes ist mindestens die Bezeichnung des verwendeten Werkzeugs und die Version erforderlich. Darüber hinaus sind die folgenden Angaben zu machen:

- Angabe der direkt übernommenen Generate des KI-Werkzeugs.
- Zeit und Datum der Nutzung des Werkzeugs.
- Die vollständige Eingabe in das Werkzeug, z. B. durch Prompts.
- Die vollständige Ausgabe des Werkzeugs.
- Falls vorhanden: Die Internetadresse, unter der das Werkzeug aufgerufen wurde.
- eigene Erweiterungen...

Bei der mehrstufigen Nutzung von KI-Werkzeugen in Form einer Sequenz von Überarbeitungen eines Inhalts sind alle Zwischenschritte einzeln zu dokumentieren. Dies, sowie Ein- und Ausgabe, können ggf. mit Links zu unveränderlichen Chatprotokollen sichergestellt werden.

21.08.2024

Datum

Thomas Hug

Unterschrift der prüfenden Person

Dokumentation entsprechend der Freigabeerklärung

Gemäß der Freigabeerklärung auf Seite 137 folgt hier eine kurze Dokumentation zur Verwendung von KIs. Zur Korrektur und Verbesserung des Ausdrucks wurde das Modell GPT-4o mini von ChatGPT verwendet.