# Der Schleimpilz

Modellierungsprojekt

Karl-Franzens-Universität Graz
Institut für Mathematik
Universitätsplatz 11
8010 Graz

by

Christian Kohlfürst & Magdalena Schreilechner

Vortagender: Dr. Stephen L. Keeling

Graz, the 10. Juli 2010

# Inhaltsverzeichnis

Die Grundlage für unser Modellierungsprojekt bildet der Schleimpilz *Physarum polycephalum*, er ist aus der Ordnung der *Physaida* und besiedelt verrottendes Holz und die Fruchtkörper fleischiger Pilze. Er dient in der Biologie als häufiger Modellorganismus zur Untersuchung der Zellmobilität und des Zellwachstums.

Als Anregung zu diesem Projekt dient ein Experiment japanischer Forscher, indem erkennbar wird, dass sich dieser Schleimpilz nicht nur bewegen, sondern auch „intelligente Entscheidungen“ treffen kann:

> Dazu nahm das Forscherteam ein 25x35 Zentimeter großes Labyrinth aus einer Kunststoffschablone und füllte die 'Gänge' mit einer Nährstofflösung. In der Mitte wurde der Pilz platziert, der sich ziemlich bald über die ganze Agar-Lösung ausbreitete. Dann legten die Forscher an jeweils beiden Enden des Labyrinths zwei Häufchen Haferflocken, wonach der Pilz begann, zu den Futterstellen zu wandern. Dabei hätte der Physarum polycephalum zwischen vier Gängen wählen können, von denen jeweils einer um 22% länger als der nächst beste war. In allen Versuchen nahm der Pilz immer den kürzeren Weg. Es scheint, so der Leiter des Forschungsteams, 'dass zelluläre Materie primitive Intelligenz aufweist.' [Quelle: http://www.science-at-home.de/referate/schleimpilz.php]

# 1. Das Projekt

Das Projekt „Der Schleimpilz“ hat mehrere Ziele, die es zu erreichen gilt. So sind wir zum einen bestrebt das Wachstum des Pilzes zu modellieren, um damit eine Energiebilanz zu erstellen und zum anderen wollen wir versuchen Vorraussagen über sein Verhalten zu treffen. Deshalb teilt sich diese Arbeit auch in zwei Unterabschnitte. Der erste Teil befasst sich vor allem mit der mathematischen Modellierung und der Beschreibung des Lebewesens mittels abstrakter Formeln und im zweiten Abschnitt wird, mittels eines Computerprogramms, versucht werden Aussagen über das Verhalten des Pilzes zu treffen.

Abbildung 1: Schleimpilz bei der Nahrungsaufnahme.

# 2. Das Wachstum

Wir werden das Wachstum des Schleimpilzes in zwei Schritten modellieren:

- Als ersten Schritt werden wir das Wachstum des Schleimpilzes aus gemessenen Daten bestimmen.
- Als zweiten Schritt werden wir das vermutete Modell für das Wachstum genauer überprüfen und verbessern.

## 2.1. Empirisches Wachstum

Für den ersten Schritt benutzen wir die Bilder der deutschen Presseagentur, die den Schleimpilz Physarum polycephalum im Labor zeigen, wie er sich auf einem eingeschränkten Feld mit eingeschränktem Vorhandensein von Nahrung ausbreitet. Dazu werden wir die Parameter folgender Funktionen mit der Methode der kleinsten Fehlerquadrate optimieren, um danach zu erkennen um welchen Wachstumstyp es sich handelt, wobei wir als Datenpunkte den Radius und die Fläche des Schleimpilzes zu den jeweilien Zeiten verwenden:

- Lineare Funktion:
$$y = kt + d \tag{1}$$
- exponentielle Funktion:
$$y = Ce^{kt} \tag{2}$$
- beschränkte Funktion:
$$y = S - ce^{-kt} \tag{3}$$
- logistische Funktion:
$$y = \frac{S}{1 + (\frac{S}{y_0} - 1)e^{-Skt}} \tag{4}$$

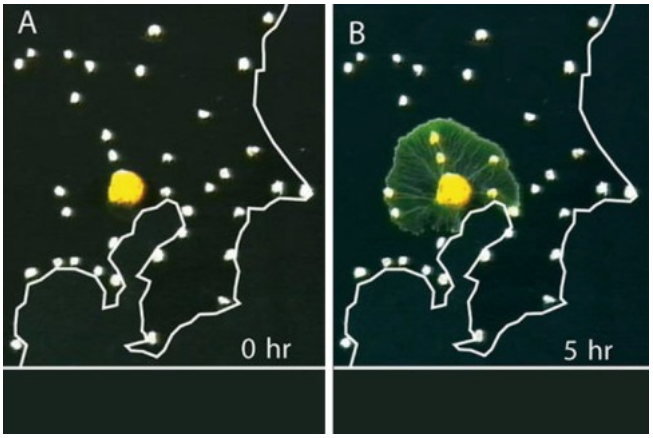



Abbildung 2: Der Radius des Schleimpilzes zu Beginn, d.h. t = 0 Stunden, beträgt 1.1 cm, bei einem Maßstab von 1 : 1.5 cm. Nach 5 Stunden beträgt der Radius 3.9 cm. Das entspricht einer Fläche von $1.21\pi\ cm^2$ für t = 0 h, und $15.21\pi\ cm^2$ für t = 5 h.

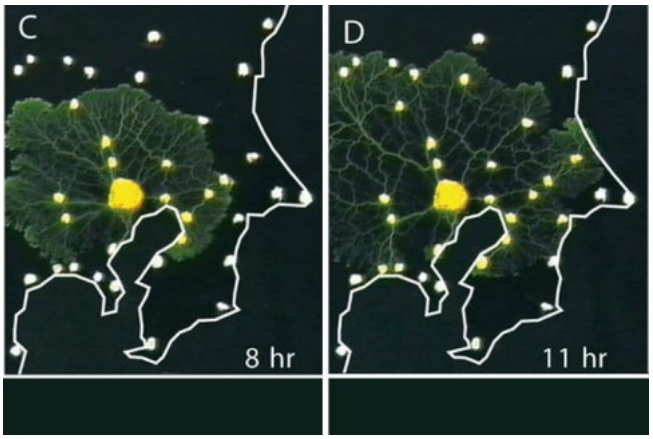



Abbildung 3: Nach 8 Stunden beträgt sein Radius 6.5 cm bzw. seine Fläche $42.25\pi\ cm^2$ , nach 11 Stunden beträgt der Radius 8.3 cm bzw.seine Fläche $68.89\pi\ cm^2$.

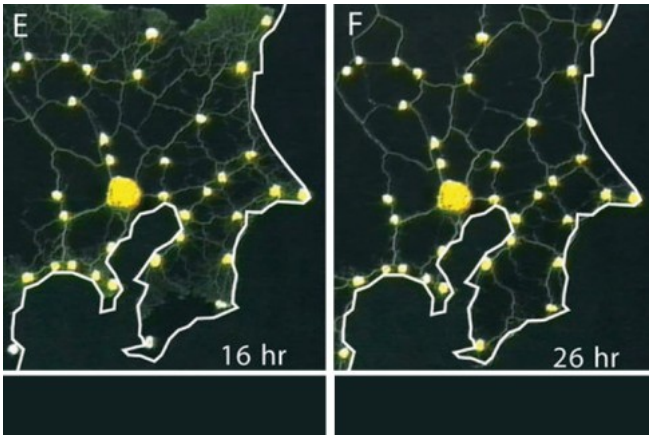



Abbildung 4: Nach 16 Stunden beträgt sein Radius 11.3 cm und seine Fläche $127.69\pi\ cm^2$. Nebenstehend der Pilz nach 26 Stunden.

Zur Übersicht:

| Zeit(h) | Radius(cm) | Fläche($cm^2$) |
|---|---|---|
| 0 | 1.1 | 3.8 |
| 5 | 3.9 | 47.78 |
| 8 | 6.5 | 132.73 |
| 11 | 8.3 | 216.42 |
| 16 | 11.3 | 401.15 |

### 2.1.1. Lineare Funktion

Da eine lineare Regression (1) mit den Daten des Radius ein gutes Ergenbnis liefert, kommt die lineare Funktion für die Daten der Fläche nicht mehr in Frage.

Minimum für die Funktion

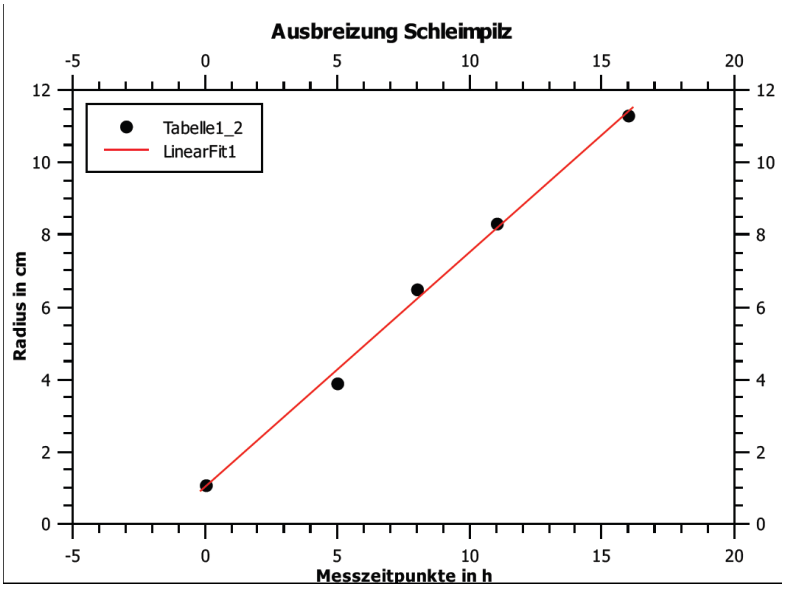
$$\mathrm{E}(k,d) = \sum_{n=1}^{5} [kt_n + d - y_n]^2 \tag{5}$$

liegt bei $k^* = \frac{\overline{ty} - \overline{t}\,\overline{y}}{\overline{x^2} - \overline{x}^2}$ und $d^* = \overline{y} - k^*\overline{t}$.

Für den Radius folgt dann $k^* = 0.65$ und $d^* = 1.02$. Somit folgt für den Korrelationskoeffizienten

$$R = \frac{\frac{1}{5}\sum_{n=1}^{5}(t_n - \overline{t})(y_- \overline{y})}{\sqrt{\frac{1}{5}\sum_{n=1}^{5}(t_n - \overline{t})^2}\sqrt{\frac{1}{5}\sum_{n=1}^{5}(y_n - \overline{y})^2}} \tag{6}$$

$R = 0.9979$.



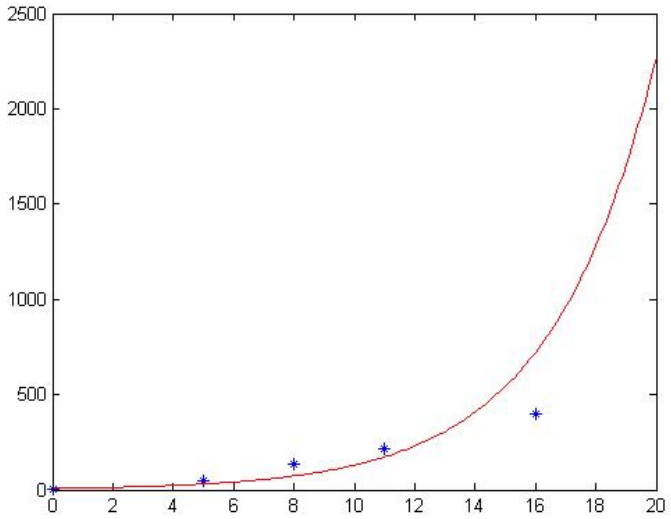

Lineare Regression für den Radius.

### 2.1.2. Exponentialfunktion

Für die Daten der Fläche des Schleimpilzes führen wir eine Regression für die Exponentialfunktion (2) durch. Durch Transformation der Daten können wir den Vorgang auf eine lineare Regression zurückführen. Durch Logarithmieren der Exponentialfunktion (2) erhalten wir:

$$\ln y_n = \ln C + kt_n \tag{7}$$

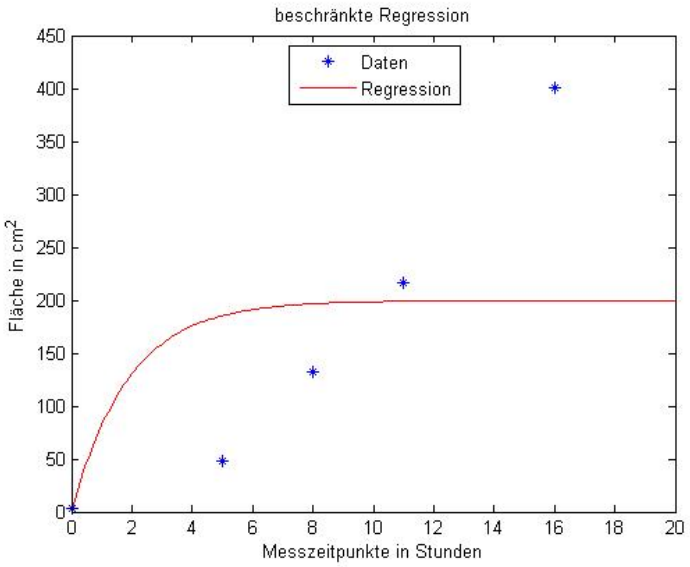
Aus (5) erhalten wir: $d^* = \ln C^* = 2.0015 \Rightarrow C^* = e^{2.0015} \approx 7.4$ und $k^* = 0.2863$. Aus (6) erhalten wir $R = 0.9467$.




Regression für die Exponentialfunktion.

### 2.1.3. Beschränktes Wachstum

Für eine Regression der beschränkten Wachstumsfunktion (3) und der logistischen Funktion (4) , haben wir Matlab und fminsearch verwendet. Als optimale Parameter für die beschränkte Funktion erhalten wir somit:
$S = 199.25$
$c = 198.15$ und
$k = 0.53$
Dass die Funktion mit diesen Parametern nicht optimal ist zeigt die Graphik:

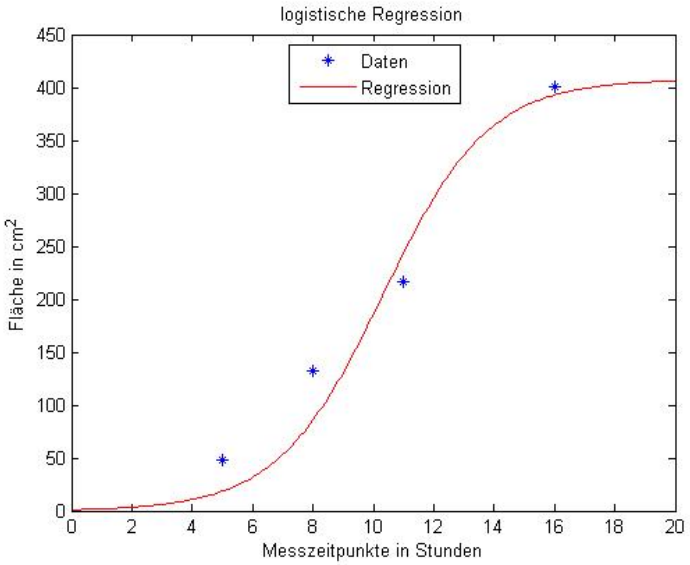



Regression für die beschränkte Funktion.

### 2.1.4. Logistisches Wachstum

Als optimale Parameter für die logistische Funktion erhalten wir:
$S = 407.6$
$k = 0.001408$

Die logistische Funktion liefert mit diesen Parametern eine gute Regression:




Regression für die logistische Funktion.

## 2.2. Modifiziertes Wachstum

Da wir für die Bestimmung des Wachstumsmodells nur sehr wenig Information zur Verfügung haben, werden wir ein paar Annahmen treffen, um das Wachstum des Schleimpilzes zu beschreiben:

- Der Schleimpilz hat nur eine beschränkte Anzahl von Futterquellen, die nur eine beschränkte Anzahl von Nahrungseinheiten besitzen, zur Verfügung.
- Der Schleimpilz breitet sich nur in 2 Dimensionen aus.

Aus Kapitel 2.1 erhalten wir, dass die logistische Funktion die Datenpunkte am besten beschreibt und darum wählen wir für das Wachstumsmodell des Schleimpilzes eine logistische Funktion:

$$M'(t) = kM(t)[S - M(t)] \tag{8}$$

Das bedeutet, dass die Änderung der $Masse = Fläche * Dichte$ proportional zur Masse, sowie proportional zur noch vorhandenen Kapazität $S$ ist. Allgemein wählen wir für die Anfangsbedingung $M(0) = m_0$, wobei gilt: $M \overset{t\to\infty}{\to} S$.

Als Lösung für diese Differentialgleichung erhalten wir die logistische Funktion

$$M(t) = \frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt}} \tag{9}$$

Die Anfangsbedingung $M(0) = m_0$ beschreibt dabei die Masse des Schleimpilzes zu Beginn des kontrollierten Wachstums. Aus der Differentialgleichung (8) erhalten wir:

$$M'(t) = k\frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt}}[S - \frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt}}] \quad = \frac{kS^2[\frac{S}{m_o} - 1]e^{-Skt}}{[1 + (\frac{S}{m_0} - 1)e^{-Skt}]^2}.$$

Um Extremstellen zu finden setzen wir die erste Ableitung 0.

$$M'(t) = \frac{kS^2[\frac{S}{m_o} - 1]e^{-Skt}}{[1 + (\frac{S}{m_0} - 1)e^{-Skt}]^2} = 0.$$

Der Nenner der Funktion ist stets $\neq 0$ (Wir betrachten hier nicht die Fälle $S = m_0$ sowie $S = 0$). Nullstellen des Zählers:

$$kS^2[\frac{S}{m_o} - 1]e^{-Skt} = 0 \text{ für } e^{-Skt} = 0 \text{ für } t \to \infty.$$

Für den Funktionswert $M(t)$ gilt dann:

$$\lim_{t\to\infty} \frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt}} \to S.$$

Extremstellen von $M(t)$ sind also die Randwerte $t = 0$ und $t = \infty$. Da für $t_1 < t_2$ gilt: $\frac{S}{1+(\frac{S}{m_0}-1)e^{-Skt_1}} < \frac{S}{1+(\frac{S}{m_0}-1)e^{-Skt_2}}$ ist $M(t)$ streng monoton wachsend $\Rightarrow t = 0$ ist ein globales Minmum, $t \to \infty$ ein globales Maximum für $M(t)$.

Der Fall $S = m_0$ bedeutet, dass kein Wachstum stattfindet, da der Schleimpilz seine Grenze bereits am Anfang erreicht hat. $S = 0$ bedeutet, dass der Pilz komplett verschwindet. Da diese Fälle trivial bzw. unnatürlich sind, wollen wir im Folgenden nicht mehr genauer darauf eingehen.

Um den Wendepunkt zu finden betrachten wir die zweite Ableitung der Funktion:

$$\begin{aligned} M''(t) &= \frac{d}{dt} kM(t)[S - M(t)] \\ &= kM'(t)[S - M(t)] - kM(t_)M'(t) \\ &= kM'(t)[S - M(t) - M(t)] \\ &= kM'(t)[S - 2M(t)] \end{aligned}$$

Nullsetzen der zweiten Ableitung, $0 < t_0 < \infty$:

$$0 = M''(t_0) = kM'(t_0)[S - 2M(t_0)] \Rightarrow [S - 2M(t_0)] = 0 \Rightarrow M(t_0) = \frac{S}{2}.$$

Für den Wendepunkt $t_0$ gilt also:

$$M(t_0) = \frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt_0}} = \frac{S}{2} \Rightarrow$$

$$\begin{aligned} \frac{S}{1 + (\frac{S}{m_0} - 1)e^{-Skt_0}} &= \frac{S}{2} \\ \frac{1}{1 + (\frac{S}{m_0} - 1)e^{-Skt_0}} &= \frac{1}{2} \\ 1 + (\frac{S}{m_0} - 1)e^{-Skt_0} &= 2 \\ (\frac{S}{m_0} - 1)e^{-Skt_0} &= 1 \\ e^{Skt_0} &= \frac{1}{(\frac{S}{m_0} - 1)} \\ t_0 &= \frac{\ln \frac{1}{(\frac{S}{m_0}-1)}}{Sk} \end{aligned}$$

Damit haben wir einen eindeutigen Wendepunkt $t_0$ gefunden mit : $0 < t_0 < \infty$. In diesem Punkt hat der Schleimpilz seine maximale Wachstumsgeschwindigkeit:

$$M'(t_0) = k\frac{S}{2}[S - \frac{S}{2}] = \frac{kS^2}{4}$$

Um zu zeigen, dass $M'(t)$ in $t_0 = \frac{\ln \frac{1}{(\frac{S}{m_0}-1)}}{Sk}$ wirklich ein Maximum besitzt betrachten wir die dritte Ableitung von $M(t)$ :

$$\begin{aligned} M'''(t) &= kM'(t)[-2M'(t)] + kM''(t)[S - 2M(t)] \\ M'''(t_0) &= kM'(t_0)[-2M'(t_0)] + kM''(t_0)[S - 2M(t_0)] \\ &= k\frac{kS^2}{4}[-2\frac{kS^2}{4}] \\ &= -2k[\frac{kS^2}{4}]^2 \end{aligned}$$

$M'''(t_0)$ ist negativ $\Rightarrow M'(t)$ besitzt ein Maximum in $t_0$.

Da der Schleimpilz in der Lage ist die kürzesten Verbindungen zwischen Futterqellen zu finden, wird dem Schleimpilz „intelligentes Handeln“ zugesprochen. Dies geht in diesem Modell fogendermaßen ein: Zu Beginn der Futtersuche breitet sich der Schleimpilz über die gesamte ihm zur Verfügung stehende Fläche aus. Sobald er einige Futterquellen gefunden hat, beginnt er sich von den „leeren“ Feldern, so gut wie möglich zurückzuziehen. Dabei verliert er jedoch nicht an Masse, sondern baut die Zuleitungen zu den Futterquellen mit dem gewonnenen Material weiter aus, um mehr Energie daraus zu erlangen, d.h. er er gleicht die Abnahme der Fläche mit der Zunahme der Dichte aus. Dieses Verhalten ändert die Gesammtmasse also nicht, und darum ist dieses Modell sinnvoll. (Dieses Verhalten ist auf den Fotos in Kapitel 2.1 sehr gut zu erkennen).

Kommentar zu den Annahmen: Sollte der Schleimpilz keine konrollierte Umgebung und nicht nur einen beschränkten Vorrat an Futter zur Verfügung haben, dann würde er bis zu einer bestimmten Größe wachsen, wobei diese Größe eine instabile Form aufweist. So endet seine Ausbreitung irgendwann von selbst, oder mit einem Nahrungsmangel. Die zweite Annahme, dass sich der Schleimpilz nur in 2 Dimensionen ausbreitet, folgt daraus, dass wir hier nur das erste Stadium seines Wachstums betrachten. Wenn der Schleimpilz groß genug ist, beginnt er nämlich zusätzlich zu seiner flächendeckenden Ausbreitung auch kleine „Füßchen“ zu bilden und aus der Ebene hinaus zu wachsen.

# 3. Die Futterquellen

Für den Schleimpilz gibt es $N \in \mathbb{N}$ Futterquellen zu finden. Jede Futterquelle $F_1, \ldots, F_N$ besitzt eine unterschiedliche Menge an Futtereinheiten. $F_i(t)$ beschreibt dann die zur Zeit t vorhandene Menge an Nahung in der i-ten Nahrungsquelle, mit Anfangswerten $F_i(0) = f_i$ mit $f_i \in \mathbb{N}$. Der Schleimpilz baut die Nahrung mit einer konstanten Rate $r$ ab, wobei die unterschiedlichen Fundzeitpunkte $z_1, \ldots, z_N$ berücksichtigt werden müssen.

Es gilt also:

$$F_i'(t) = \begin{cases} -r, & \text{wenn } t \geq z_i \\ 0, & \text{wenn } t < z_i \end{cases}$$

Mit den Anfangsbedingungen $f_i$ erhalten wir, dass sich der Nahrungsvorrat pro Futterquelle folgendermaßen verhält:

$$F_i(t) = \begin{cases} f_i - rt, & \text{wenn } f_i \geq rt \text{ und } t \geq z_i \\ 0, & \text{sonst} \end{cases} \tag{10}$$

Also lautet die Differentialgleichung vollständig:

$$F_i'(t) = \begin{cases} -r, & \text{wenn } t \geq z_i \text{ und } f_i - rt > 0 \\ 0, & \text{sonst} \end{cases} \tag{11}$$

# 4. Energiebilanz

Wachstum kostet dem Schleimpilz Energie, Nahrung liefert ihm Energie.

$$E'(t) = -lM'(t) + rA(t) \tag{12}$$

Wobei r die konstante Futterabbaurate ist und $l > 0$ angibt, wieviel Energie Wachstum pro Zeit kostet.

$A(t)$ ist dabei die Anzahl der zur Zeit $t$ gefundenen und nicht leeren Futterquellen. Mit (8) folgt:

$$\begin{aligned} E'(t) &= -lM'(t) + rA(t) \\ &= -lkM(t)[S - M(t)] + rA(t) \end{aligned}$$

Als Randbedingungen für die Differentialgleichung (12) wählen wir:

$E(0) = e_0 > 0$.

Für $A(t)$ gilt:

$$A(t) = A_g(t) - A_{gl}(t) \tag{13}$$

Wobei $A_g(t)$ die Anzahl der gefundenen, und $A_{gl}(t)$ die Anzahl der gefundenen und leeren Futterquellen ist.

$$A_g(t) = \begin{cases} k, & \text{für } z_k \leq t < z_{k+1} \text{ für } k = 1, ...., N-1 \\ N, & \text{für } z_N \leq t \end{cases}$$

$$A_{gl}(t) = \sum_{i=1}^{N} \frac{1}{fi - r(t - z_i)} \min\{0, fi - r(t - z_i)\}$$

Wobei für $fi - r(t - z_i) = 0$ gilt: $\frac{1}{fi - r(t - z_i)} \min\{0, fi - r(t - z_i) = 1$

Es gibt also Zeitpunkte $t_i$ min $i \in \{1, ..., N\}$, sodass $A_{gl}$ konstant in $[t_i,\ t_{i+1}]$ ist. Diese $t_i$ errechnen sich folgenderweise:

$t_i = \frac{f_i + rz_i}{r}$

Solange $fi - r(t - z_i)$ positiv ist, liefert die Quelle $F_i$ keinen Beitrag zu $A$, sobald dieser Ausdruck jedoch negativ wird, wird $F_i$ von $A$ einmal abgezogen.

$A_g(t)$ und $A_{gl}$ sind Treppenfunktionen, also integrierbar. Durch Integration folgt dann für (12):

$$\begin{aligned} E(t) = \int E'(s)ds &= -l \int M'(s)ds + r \int A_g(s)ds - r \int A_{gl}(s)ds \\ &= -lM(t) + r[\sum_{l=0}^{k-1}(z_{l+1} - z_l)l + (t - z_k)k] - r \int A_{gl}(s)ds + C \end{aligned}$$

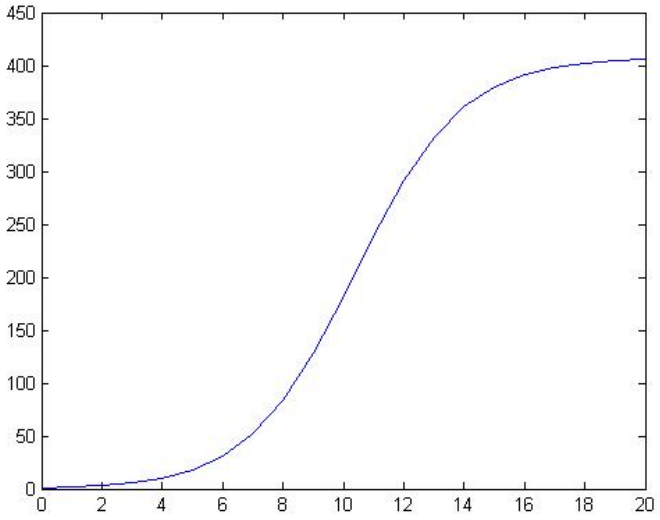
Aus der Anfangsbedingung $E(0) = e_0 > 0$ folgt $C = e_0 + lm_0$

$$E(t) = -lM(t) + r[\sum_{l=0}^{k-1}(z_{l+1} - z_l)l + (t - z_k)k] - r\int A_{gl}(s)ds + e_0 + lm_0 \qquad (14)$$

Wobei $z_0 = 0$ und $z_k \leq t < z_{k+1}$ mit $k \in \{0, ...N-1\}$.

## 4.1. Einfache Simulation

Wir verwenden die Daten die wir für die logistische Regression erhalten haben:
$S = 407$
$k = 0.0014$
$m_0 = 1.1$
$l = 0.1$
$r = 4$
$0 \leq t \leq 20$
$z = [1, 2, 4, 6, 9, 11, 12, 16, 18]$ Fundzeitpunkte
$f = [11, 10, 4, 13, 6, 7, 9, 6, 2]$ Futterquelleninhalt




Wachstum M

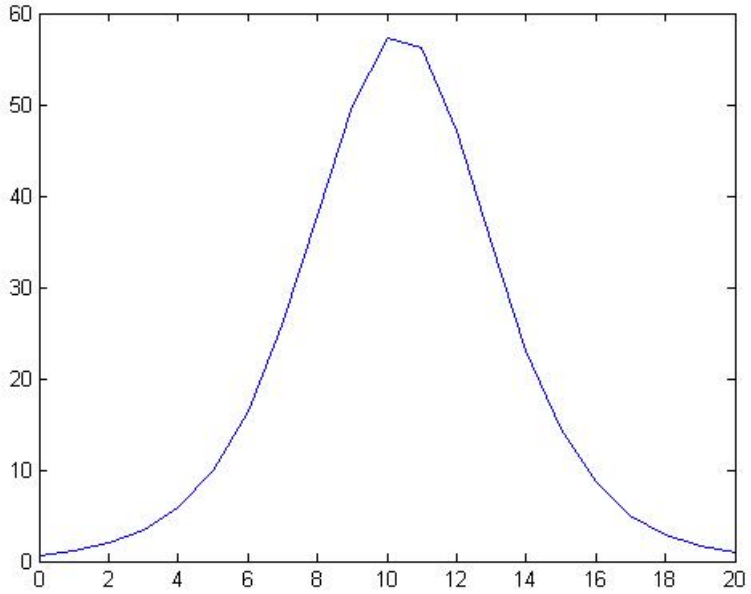



Abbildung 5: Wachstumsgeschwindigkeit M'

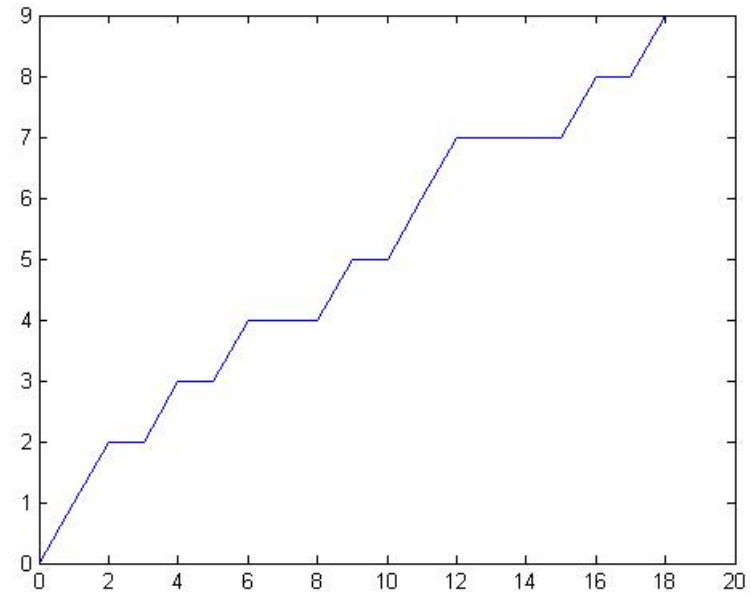



Abbildung 6: Gefundene Futterquellen $A_g$

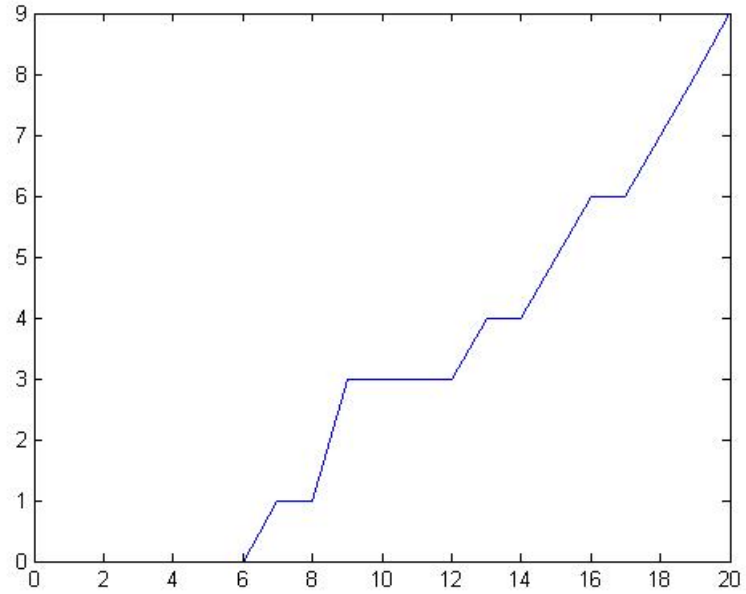



Abbildung 7: Gefundene und leere Futterquellen $A_{gl}$

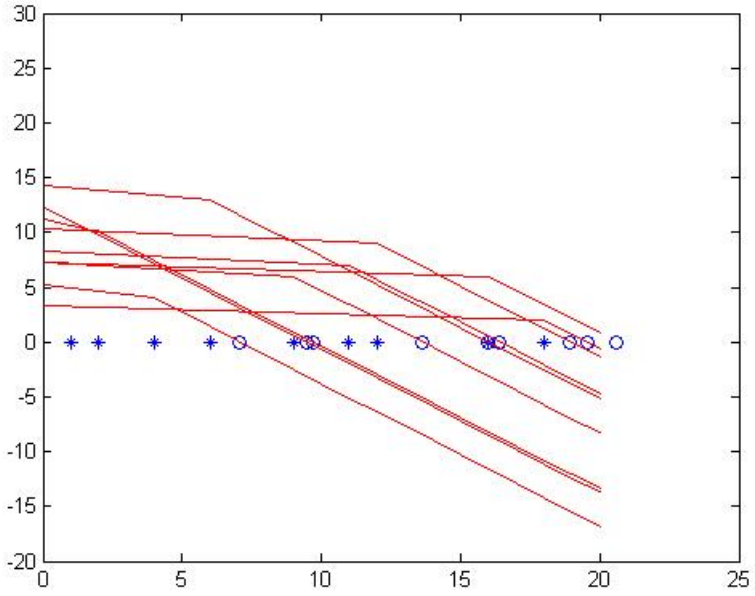



Abbildung 8: Futterabbau; *:Fundzeitpunkte; o:Zeitpunkte des Leerwerdens

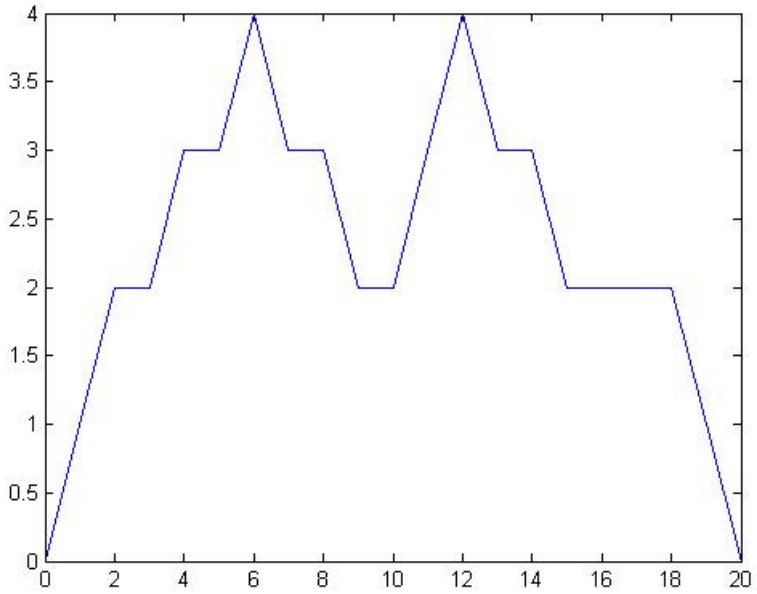



Abbildung 9: Gefundene nicht leere Futterquellen $A_g - A_{gl}$

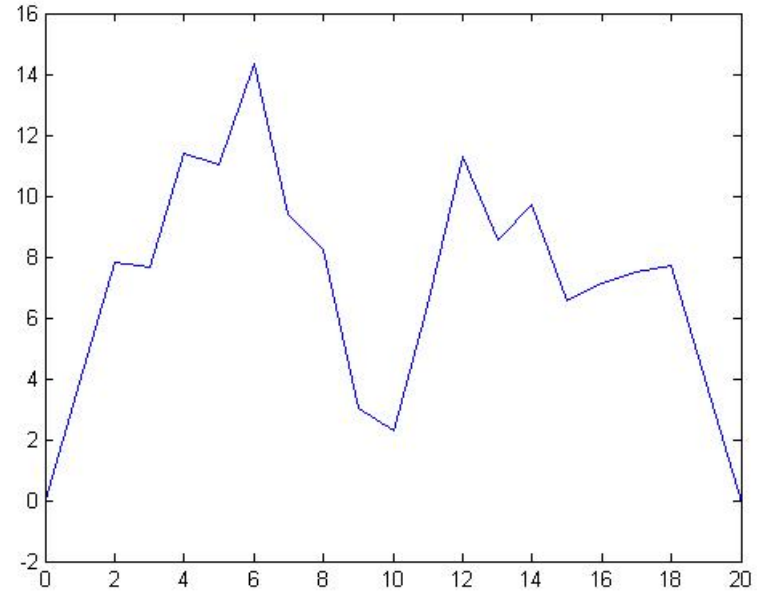



Abbildung 10: Energieänderung E'

# 5. Programm

## 5.1. Erste Überlegungen

Die Simulation des Pilzes wird in C++ durchgeführt. Dabei wird das Wachstum des Pilzes auf einem $nxn$ -Gitter durchgeführt und die Zeit in diskreten Schritten erhöht. Die grafische Auswertung der Simulation erfolgt mit MATLAB, wobei vor allem die einzelnen Verbindungen zwischen zwei Gitterpunkten dargestellt werden. Durch die Diskretisierung von Raum und Zeit können effiziente Unterprogramme zur Wegfindung und zum Aufbau der Verbindungen zwischen zwei Energiequellen geschrieben werden. Wichtig dabei ist, dass nicht nur die einzelnen Gitterpunkte protokolliert werden sondern dass vor allem die Verbindungen zwischen den Punkten gespeichert werden.

### 5.1.1. Wachstum

Anfangs war am wichtigsten, dass sich der Schleimpilz, wie in der Natur, ausbreitet so weit er kann. Das wurde in der ersten Version realisiert. Dabei wurde ein Punkt am Gitter als Startpunkt deklariert und der Pilz breitet sich dann von einem bereits besetzten Gitterpunkt in alle möglichen Richtungen aus. Da die Zeit in diskreten Schritten implementiert wurde, musste das Wachstum pro Zeiteinheit begrenzt werden um unkontrolliertes Anwachsen des Pilzes zu verhindern.

### 5.1.2. Nahrung

Die nächste Überlegung betraf den Einbau von Energiequellen. In einer natürlichen Umgebung benötigen Schleimpilze Nahrung, in welcher Form auch immer, um überleben zu können. Dabei erweitert der Pilz die Verbindungen vom Zentrum zur gefundenen Quelle.

Da in dieser frühen Version der Pilz keine Hindernisse außer dem Ende des Gitters kennt wurde der Weg Nahrung-Zentrum per Skript vorgegeben. Sobald der Pilz also Nahrung findet, d.h. wird ein Gitterpunkt auf dem eine Nahrungsquelle liegt vom Pilz besetzt, so wird ein Zick-Zack-Weg von der Nahrung zum Zentrum gebaut und alle benötigten Verbindungen dabei erweitert.

Für einen zweiten Nahrungspunkt gibt es nun mehrere Möglichkeit eine Verbindung herzustellen. Entweder wird eine Verbindung von der Nahrungsquelle zum Zentrum hergestellt oder der Pilz stellt eine Verbindung von der Nahrung zu einem bereits ausgebauten Weg her.

## 5.2. Verbesserungen

Das Problem an der obigen Version ist einerseits, dass sie keine Hindernisse etc. berücksichtigt und andererseits die optimalen Verbindungen nicht vom Programm gesucht sondern durch menschliche Überlegungen eingebaut werden. Deshalb ist als erstes ein Algorithmus zur Wegsuche zu implementieren. Dieser sollte berücksichtigen, dass der Schleimpilz kein Gedächtnis hat sondern nur reflexartig reagieren kann und trotzdem optimale Wege zwischen zwei Punkten finden, auch wenn der direkte Weg blockiert ist.

### 5.2.1. Wegfindung

Der Ansatz für die Wegfindung sieht ab dieser Version folgendermaßen aus: Der Pilz findet reflexartig, d.h. ohne kompliziertes Wissen einen Weg zum Ziel, dazu 'merkt' er sich einen Zufallsweg von der Nahrungsquelle zum Zentrum. Der Zufallsweg sieht so aus, dass sich der Pilz ausgehend von der Nahrung an jedem Gitterpunkt in eine der möglichen Richtungen bewegt bis er sein Ziel erreicht hat. Um Optimalität zu gewährleisten wird dieses System zum Finden eines Weges mehrfach angewendet, so werden im allgemeinen sehr schlechte Wege vermieden.

### 5.2.2. Hindernisse und mehrere Nahrungsquellen

Zum einen werden nun Hindernisse eingebaut, das sind Gitterplätze die der Pilz nicht belegen kann und zum anderen wieder beliebig viele Nahrungsquellen berücksichtigt. Dabei ist es von Vorteil, dass der Wegfindungsalgorithmus leicht erweitert werden kann. So wird die Wahrscheinlichkeit dass der Zufallsweg in ein Hinderniss geht auf null gesetzt. Die weiteren Nahrungsquellen bewirken nur, dass eine verstärkte Verbindung nun ebenfalls als Ziel des Zufallsweges berücksichtigt wird.

## 5.3. Energie

Eine Schwachstelle der bisherigen Versionen der Simulation war, dass sich der Pilz immer ungehindert ausbreiten konnte; Energieerhaltung oder Vorteile des Findens von Nahrung blieben unberücksichtigt. Deshalb wurde ein 'Energiekonto' eingeführt: Der Pilz startet mit einer gewissen Menge an Energie. Jeder besetzte Gitterpunkt bzw. jeder Ausbau einer Verbindung kostet Energie während jede Nahrungsquelle pro Zeiteinheit Energie liefert. Mit diesem Ansatz ist es nun möglich das Wachstum des Pilzes besser zu simulieren und sogar zu beschränken.

# 6. Ergebnisse der Computersimulation

Hier werden die Ergebnisse der Simulation dargestellt. Als wichtigstes Merkmal kann die Wegfindung genannt werden, weshalb als erstes eine Simulation in einem Labyrinth durchgeführt wurde.

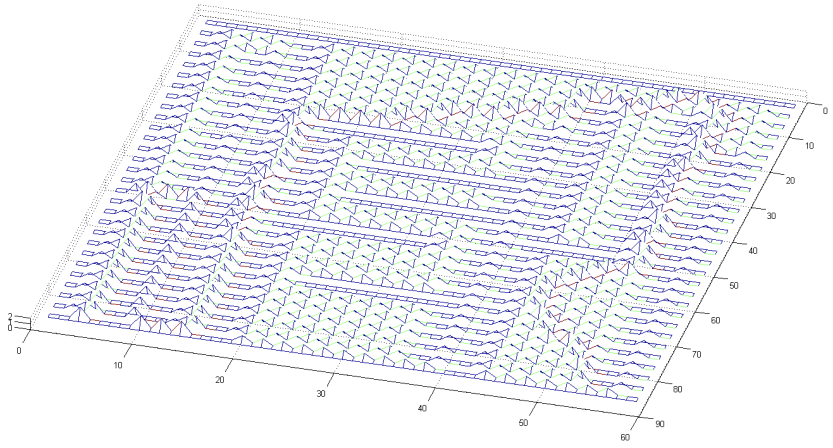


Abbildung 11: Der Pilz sucht sich den Weg durchs Labyrinth selbstständig.

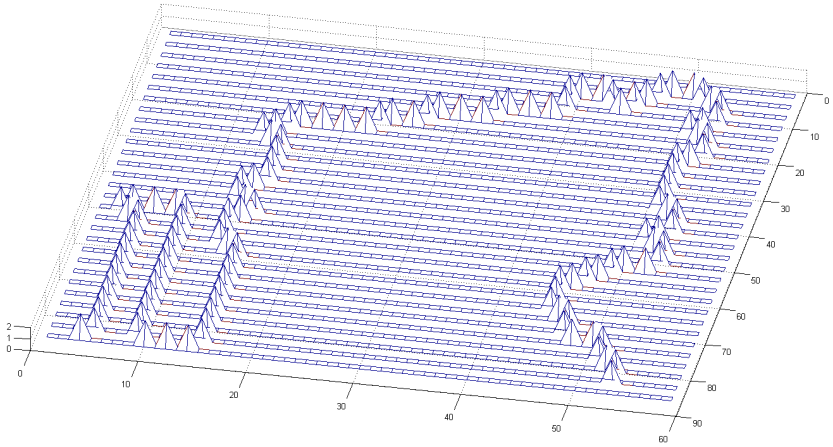


Abbildung 12: Nur der Weg nochmal herausgezeichnet.

Anschließend wurde eine Karte Japans nachgebaut um einen Vergleich mit der Natur zu haben(siehe Fotostrecke bei Kapitel 2.1).

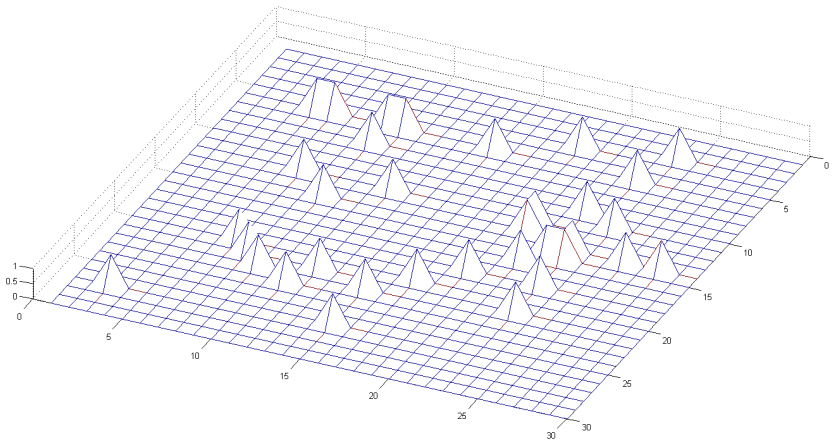


Abbildung 13: Nahrungsverteilung am Gitter.

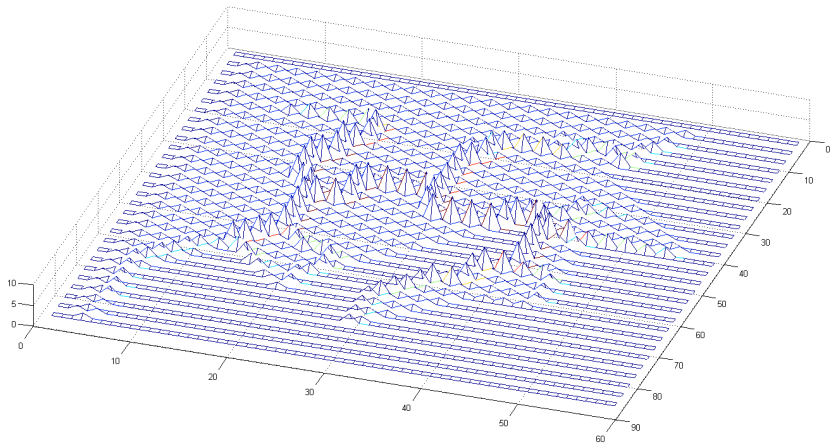


Abbildung 14: Ein vollständiges Bild der Simulation.

Schließlich wurde begonnen Simulationen auf einem freien Gitter durchzuführen um vor allem die Energiekurven des Modells zu betrachten. Ausgehend von einer zufälligen Nahrungsverteilung mit 30 Nahrungsquellen wurden so nachfolgendes Modell berechnet.

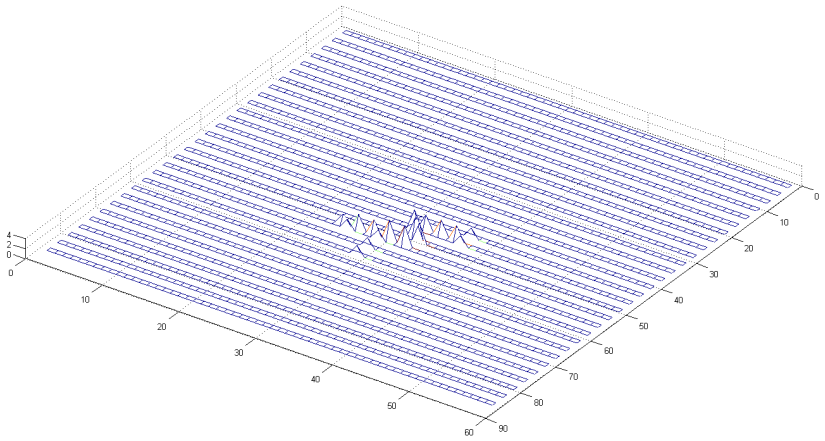



Abbildung 15: Ein vollständiges Bild der Simulation.

Außerdem war es nun möglich die Gesamtenergie, die Anzahl der bereits erschlossenen Quelle sowie die noch ergiebigen Quellen aufzuzeichnen. Zu beachten ist dabei, dass die Simulation abbricht falls die Energie des Pilzes negativ wird.

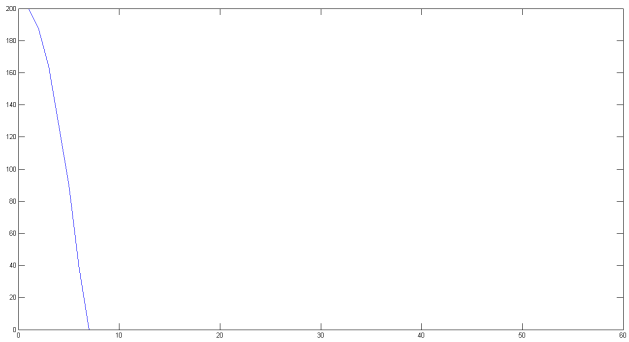


Abbildung 16: Aufgezeichnete Energien.

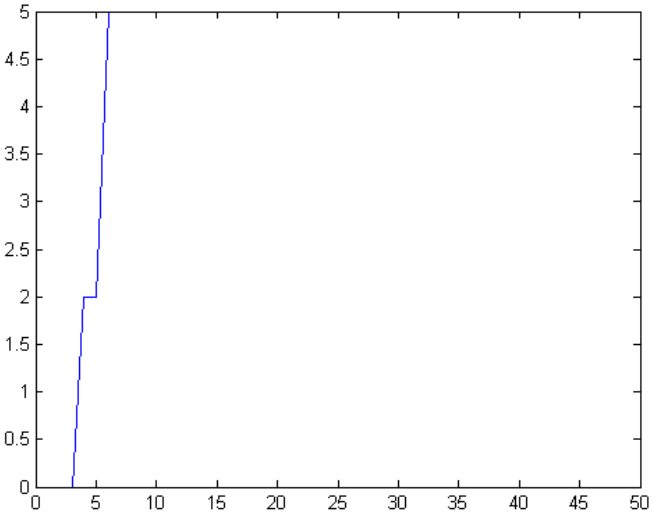



Abbildung 17: Erschlossene Quellen pro Zeiteinheit.

Hier nun die Simulation mit 80 Nahrungsquellen. Man sieht sofort, dass die nun gefundene Energie ausreicht um den Pilz über ein beträchtlich größeres Areal wachsen zu lassen.

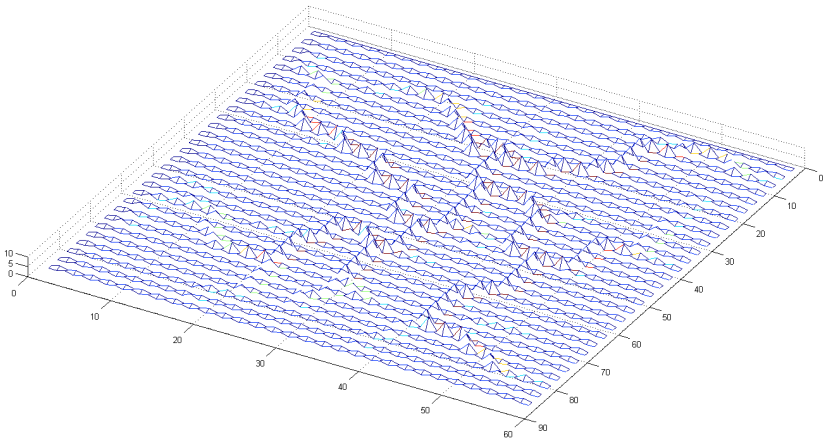



Abbildung 18: Die Simulation auf dem Gitter.

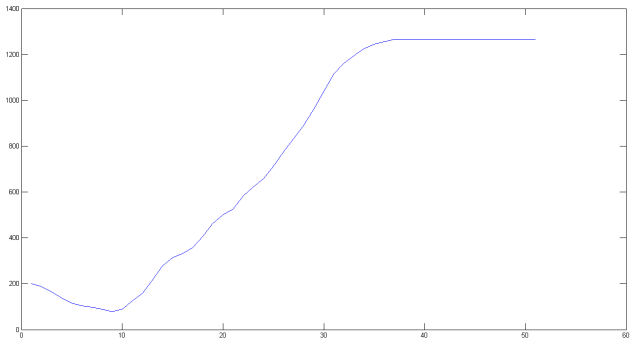



Abbildung 19: Aufgezeichnete Energien.

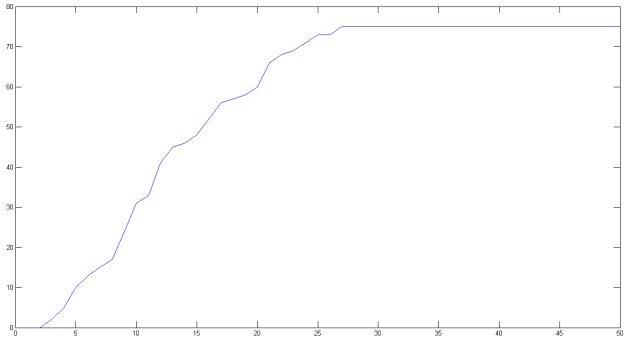



Abbildung 20: Die vom Pilz gefundenen Quellen.

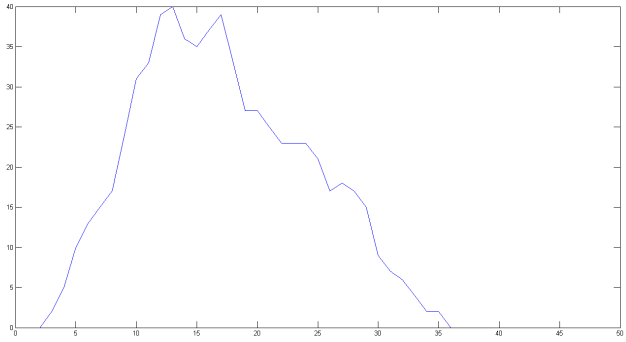



Abbildung 21: Die erschlossenen und noch ergiebigen Quellen.

# A. C++ Code

```
1   // Modellierung1.cpp : Der Schleimpilz
2   //
3
4   #include "stdafx.h"
5   #include <iostream>
6   #include <fstream>
7   #include <cmath>
8   #include <vector>
9
10  using namespace std;
11
12  const int n = 30;
13  const int start = 465;
14  const int t = 50;
15
16  void initneib(int neib[n*n][4]);
17  void initlattice(int site[], int line_l[], int line_q[], bool site_food[], bool site_food_found[], int site_food_res[], bool block[],
18  void dynamic1(int site[], int line_l[], int line_q[], int neib[n*n][4], bool block[], int & konto);
19  void nahrung1(int site[], int line_l[], int line_q[], int neib[n*n][4], bool site_food[], bool site_food_found[], int & konto);
20  void dynamic_way2(int site[], int neib[n*n][4], int line_l[], int line_q[], vector<int> & gitter, int & konto);
21  void energie(int site[], int line_l[], int line_q[], bool site_food_found[], int site_food_res[], int & konto, bool & ende);
22
23  int _tmain(int argc, _TCHAR* argv[])
24  {
25          cout << "Modellierung:␣Schleimpilz" << endl;
26          //Gitter initialisieren
27          //Gitter 1 speichert ob ein Gitterpunkt besetzt ist
28          int site[n*n];
29          //Gitter 2, 3 speichert die Liniendicke zwischen den Gitterpunkten
30          int line_l[n*(n-1)];
31          int line_q[n*(n-1)];
32          //Nachbarn werden extra gespeichert
33          int neib[n*n][4];
34
35          //Nahrungsgitter
36          bool site_food[n*n];
37          bool site_food_found[n*n];
38          int site_food_res[n*n];
39
40          //Hindernisse als Gitter speichern
41          bool block[n*n];
42
43          //Einführung eines Punktekontos
44          int konto[t+1];
45          bool ende = false;
46
47          int protokoll[t];
48          for(int i = 0; i < t;i++)
49                  protokoll[i] = 0;
50          int help;
51
52          initlattice(site, line_l, line_q, site_food, site_food_found, site_food_res, block, konto);
53          initneib(neib);
54
55          //Die Schleife für die Zeit
56          for(int i = 0; i < t; i++) {
57                  //Konto aktualisieren
58                  konto[i+1] = konto[i];
59                  //1.Versuch: Ausbreitung wenn möglich
60                  if(ende == false) dynamic1(site, line_l, line_q, neib, block, konto[i+1]);
61                  //Nahrung mal mit Zufallsweg
62                  if(ende == false) nahrung1(site, line_l, line_q, neib, site_food, site_food_found, konto[i+1]);
63                  if(ende == false) energie(site, line_l, line_q, site_food_found, site_food_res, konto[i+1], ende);
64                  for(int j = 0; j < n*n; j++) {
65                          if(site_food_found[j] == true && site_food_res[j] > 0) help = 1;
66                          else help = 0;
67                          protokoll[i] = protokoll[i] + help;
68                  }
69          }
70
71          //Ausgabe der Daten
72          ofstream file_line_q;
73          file_line_q.imbue( locale("german") );
74          file_line_q.open("line_q.data");
75
76          for(int i = 0; i < n-1; i++){
77                  for(int j = 0; j < n; j++) {
78                          file_line_q << line_q[j+i*n] << "␣";
79                  }
80                  file_line_q << endl;
81          }
```

```
82          file_line_q.close();
83
84          ofstream file_line_l;
85          file_line_l.imbue( locale("german") );
86          file_line_l.open("line_l.data");
87
88          for(int i = 0; i < n; i++){
89                  for(int j = 0; j < n-1; j++) {
90                          file_line_l << line_l[j+i*(n-1)] << "␣";
91                  }
92                  file_line_l << endl;
93          }
94          file_line_l.close();
95
96          ofstream file_site;
97          file_site.imbue( locale("german") );
98          file_site.open("site.data");
99
100         for(int i = 0; i < n; i++){
101                 for(int j = 0; j < n; j++) {
102                         file_site << site[j+i*n] << "␣";
103                 }
104                 file_site << endl;
105         }
106         file_site.close();
107
108         ofstream file_vis;
109         file_vis.imbue( locale("german") );
110         file_vis.open("vis.data");
111
112         for(int i = 0; i < n; i++){
113                 file_vis << "0␣";
114                 for(int j = 0; j < n-1; j++) {
115                         file_vis << min(6, line_l[j+i*(n-1)]) << "␣0␣";
116                 }
117                 file_vis << endl;
118                 for(int j = 0; j < n; j++){
119                         if(i<n-1) file_vis << min(6, line_q[j+i*n]) << "␣0␣";
120                 }
121                 file_vis << endl;
122         }
123         file_vis.close();
124
125         ofstream file_energie;
126         file_energie.open("energie.data");
127
128         for(int i = 0; i < t; i++){
129                 file_energie << protokoll[i] << endl;
130         }
131         file_energie.close();
132
133         ofstream file_food;
134         file_food.imbue( locale("german") );
135         file_food.open("food.data");
136
137         for(int i = 0; i < n; i++){
138                 for(int j = 0; j < n; j++) {
139                         file_food << site_food[j+i*n] << "␣";
140                 }
141                 file_food << endl;
142         }
143         file_food.close();
144
145         //Programmende
146         cin >> site[0];
147         return 0;
148 }
149
150 //Die Nachbarn für jeden Punkt speichern >> erspart später das Rechnen
151 void initneib(int neib[][4]) {
152
153         int n_neigh[4];
154
155         for(int i = 0; i < n; i++) {
156                 for(int j = 0; j < n; j++) {
157
158                         n_neigh[0] = j+1;
159                         if(n_neigh[0] == n) n_neigh[0] = 0;
160
161                         n_neigh[1] = j-1;
162                         if(n_neigh[1] == -1) n_neigh[1] = n-1;
163
164                         n_neigh[2] = i+1;
165                         if(n_neigh[2] == n) n_neigh[2] = 0;
```

```
166
167                         n_neigh[3] = i-1;
168                         if(n_neigh[3] == -1) n_neigh[3] = n-1;
169
170                         neib[j+i*n][0] = j + n*n_neigh[2];
171                         neib[j+i*n][1] = n_neigh[0] + n*i;
172                         neib[j+i*n][2] = j + n*n_neigh[3];
173                         neib[j+i*n][3] = n_neigh[1] + n*i;
174                 }
175         }
176 }
177
178 //Die Arrays auf 0 setzen und Anfangskonfigurationen setzen
179 void initlattice(int site[], int line_1[], int line_q[], bool site_food[], bool site_food_found[], int site_food_res[], bool block[],
180
181         for(int i = 0; i < n*n; i++) {
182                 site_food[i] = false;
183                 site_food_found[i] = false;
184                 site[i] = 0;
185                 site_food_res[i] = 0;
186                 block[i] = false;
187         }
188         site[start] = 1;
189
190         //Zufallsnahrungsverteilung: Start:465
191         int random;
192         for(int i = 0; i < 30; i++) {
193                 random = rand();
194                 random = random % (n*n-2*n-2) + n;
195                 if(random % n == 0) i--;
196                 else if(random % n == n-1) i--;
197                 else {
198                         site_food[random] = true;
199                         site_food_res[random] = 50;
200                 }
201         }
202
203         /*//Das Labyrinth: Start 841
204         site_food[866] = true;
205
206         for(int i = 0; i < 30; i++) {
207                 block[i] = true;
208                 block[870+i] = true;
209                 block[i*30] = true;
210                 block[i*30+29] = true;
211         }
212         for(int i = 0; i < 13; i++) {
213                 block[i*30+513] = true;
214                 block[i*30+93] = true;
215                 block[i*30+114] = true;
216         }
217         for(int i = 0; i < 5; i++)
218                 block[504-i] = true;
219
220         for(int i = 8; i < 30; i++) {
221                 block[i*30+8] = true;
222                 block[i*30-33] = true;
223                 block[i*30-42] = true;
224                 block[i*30-130] = true;
225         }
226         for(int i = 0; i < 5; i++)
227                 block[i*30+740]=true;
228
229         for(int i  = 0; i < 28;i++)
230                 block[i*30+5] = true;
231
232         for(int i = 0; i < 7; i++) {
233                 block[249+i] = true;
234                 block[347-i] = true;
235                 block[437-i] = true;
236                 block[519+i] = true;
237                 block[677-i] = true;
238         }*/
239
240         /*//Japan: Start 406
241         for(int i = 0; i < 30; i++) {
242                 block[i] = true;
243                 block[870+i] = true;
244                 block[i*30] = true;
245                 block[i*30+29] = true;
246         }
247         for(int i = 0; i < 7; i++) {
248                 block[i*30+25-i/2] = true;
249                 block[i*30+202+i] = true;
```

```
250           }
251           block[418] = true;
252           for(int i = 0; i < 6; i++)
253                   block[448-i] = true;
254           for(int i = 0; i < 8; i++)
255                   block[i*30+443] = true;
256           for(int i = 0; i < 4; i++)
257                   block[653-i] = true;
258           for(int i = 0; i < 7; i++)
259                   block[i*30+649-i] = true;
260           for(int i = 0; i < 7; i++)
261                   block[-i*30+823] = true;
262           for(int i = 0; i < 7; i++) {
263                   block[-i*30+643+i] = true;
264                   block[-i*30+583+i] = true;
265           }
266           block[439] = true;
267           block[612] = true;
268           block[641] = true;
269           for(int i = 0; i < 3; i++)
270                   block[i*30+641] = true;
271           block[700] = true;
272           for(int i = 0; i < 4; i++)
273                   block[-i*30+699-i] = true;
274           for(int i = 0; i < 4; i++)
275                   block[i*30+606-i] = true;
276           for(int i = 0; i < 6; i++)
277                   block[i*30+693] = true;
278
279           site_food[83]  = true;
280           site_food[172] = true;
281           site_food[417] = true;
282           site_food[415] = true;
283           site_food[323] = true;
284           site_food[442] = true;
285           site_food[532] = true;
286           site_food[622] = true;
287           site_food[764] = true;
288           site_food[586] = true;
289           site_food[528] = true;
290           site_food[470] = true;
291           site_food[379] = true;
292           site_food[611] = true;
293           site_food[670] = true;
294           site_food[638] = true;
295           site_food[607] = true;
296           site_food[576] = true;
297           site_food[644] = true;
298           site_food[782] = true;
299           site_food[441] = true;
300           site_food[412] = true;
301           site_food[349] = true;
302           site_food[291] = true;
303           site_food[341] = true;
304           site_food[336] = true;
305           site_food[398] = true;
306           site_food[218] = true;
307           site_food[159] = true;
308           site_food[164] = true;
309           site_food[158] = true;
310           site_food[155] = true;
311           site_food[154] = true;
312           site_food[108] = true;
313
314           for(int i = 0; i < n*n; i++)
315                   if(site_food[i]==true) site_food_res[i] = 50;*/
316
317           for(int i = 0; i < t+1; i++)
318                   konto[i] = 0;
319           konto[0] = 200;
320
321           for(int i = 0; i < n*(n-1); i++) {
322                           line_l[i] = 0;
323                           line_q[i] = 0;
324           }
325   }
326
327   //Die Ausbreitung des Pilzes
328   //Dynamik1: Pilz breitet sich in alle möglichen Richtungen aus
329   void dynamic1(int site[], int line_l[], int line_q[], int neib[n*n][4], bool block[], int & konto) {
330
331           int delay[n*n];
332
333           for(int i = 0; i < n*n; i++)
```

```
334                         delay[i] = 0;
335
336         //Zuerst die neuen gitterpunkte besetzen;  && konto > 2 rausgenommen
337         for(int i = 1; i < n-1; i++) {
338                 for(int j = 1; j < n-1; j++) {
339                         if(delay[j+i*n] == 0 && site[j+i*n] == 1) {
340                                 if(site[neib[j+i*n][0]] == 0 && block[neib[j+i*n][0]] == false) {
341                                         delay[neib[j+i*n][0]] = 1;
342                                         site[neib[j+i*n][0]] = 1;
343                                         konto = konto - 3;
344                                 }
345                                 if(site[neib[j+i*n][1]] == 0 && block[neib[j+i*n][1]] == false) {
346                                         delay[neib[j+i*n][1]] = 1;
347                                         site[neib[j+i*n][1]] = 1;
348                                         konto = konto - 3;
349                                 }
350                                 if(site[neib[j+i*n][2]] == 0 && block[neib[j+i*n][2]] == false) {
351                                         delay[neib[j+i*n][2]] = 1;
352                                         site[neib[j+i*n][2]] = 1;
353                                         konto = konto - 3;
354                                 }
355                                 if(site[neib[j+i*n][3]] == 0 && block[neib[j+i*n][3]] == false) {
356                                         delay[neib[j+i*n][3]] = 1;
357                                         site[neib[j+i*n][3]] = 1;
358                                         konto = konto - 3;
359                                 }
360                         }
361                 }
362         }
363         //Jetzt durchs Gitter durchgehen und die Verbindungen setzen
364         for(int i = 1; i < n-1; i++) {
365                 for(int j = 1; j < n-1;j++) {
366                         if(site[j+i*n] == 1) {
367                                 if(site[neib[j+i*n][0]] == 1) line_q[j+i*n] = max(1, line_q[j+i*n]);
368                                 if(site[neib[j+i*n][1]] == 1) line_l[j+i*(n-1)] = max(1, line_l[j+i*(n-1)]);
369                                 if(site[neib[j+i*n][2]] == 1) line_q[j+(i-1)*n] = max(1, line_q[j+(i-1)*n]);
370                                 if(site[neib[j+i*n][3]] == 1) line_l[j-1+i*(n-1)] = max(1, line_l[j-1+i*(n-1)]);
371                         }
372                 }
373         }
374 }
375
376
377 void nahrung1(int site[], int line_l[], int line_q[], int neib[][4], bool site_food[], bool site_food_found[], int & konto) {
378
379         vector<int> gitter, gitter_opt;
380         bool end = false;
381         double random = 0.;
382         int site_now = 0, j= 0, gittersize = 0;
383
384         //Schleife über das ganze Gitter
385         for(int i = 0; i < n*n; i++) {
386                 //Nahrung bei Position i
387                 if(site[i] == 1 && site_food[i] == true && site_food_found[i] == false) {
388                         site_food_found[i] = true;
389                         //Den besten von x Wegen auswählen
390                         for(int l = 0; l < 10; l++) {
391                                 //Startpunkt speichern
392                                 gitter.push_back(i);
393                                 site_now = i;
394                                 end = false;
395                                 //Zufallsweg über das Gitter bis ein richtiger Gitterplatz gefunden
396                                 while(end == false) {
397                                         random = rand();
398                                         random = random /  (RAND_MAX + 1);
399                                         //Durchs Gitter gehen
400                                         if(random < 0.25 && site_now%n > 1 && site[neib[site_now][3]] == 1) {
401                                                 site_now = site_now - 1;
402                                                 gitter.push_back(site_now);
403                                         }
404                                         if(random >= 0.25 && random < 0.50 && site_now/n > 1 && site[neib[site_now][2]] == 1) {
405                                                 site_now = site_now - n;
406                                                 gitter.push_back(site_now);
407                                         }
408                                         if(random >= 0.50 && random < 0.75 && site_now%n < n-2 && site[neib[site_now][1]] == 1) {
409                                                 site_now = site_now + 1;
410                                                 gitter.push_back(site_now);
411                                         }
412                                         if(random >= 0.75 && site_now/n < n-2 && site[neib[site_now][0]] == 1) {
413                                                 site_now = site_now + n;
414                                                 gitter.push_back(site_now);
415                                         }
416                                         //Zurückgehen verhindern
417                                         j = 0;
```

```
418                                     gittersize = gitter.size();
419                                     while(j < gittersize-1) {
420                                             if(gitter[j] == site_now) {
421                                                     gittersize = j+1;
422                                                     gitter.erase(gitter.begin()+j, gitter.end()-1);
423                                                     j = -1;
424                                             }
425                                     j++;
426                                     }
427                                     //Falls beim Start, aufhören und Weg vergleichen
428                                     if(site_now == start) {
429                                             if(gitter.size() < gitter_opt.size() || gitter_opt.size() == 0) {
430                                                     gitter_opt.erase(gitter_opt.begin(), gitter_opt.end());
431                                                     for(int k = 0; k < gitter.size(); k++)
432                                                             gitter_opt.push_back(gitter[k]);
433                                             }
434                                             end = true;
435                                             gitter.erase(gitter.begin(), gitter.end());
436                                     }
437                                     else {
438                                             //Falls bei Gitterplatz mit dickerem Weg >> dorthin bauen
439                                             if(line_l[site_now - site_now/n] > 1 || line_l[site_now-1 - site_now/n] > 1 || line_q
440                                                     if(gitter.size() < gitter_opt.size() || gitter_opt.size() == 0) {
441                                                             gitter_opt.erase(gitter_opt.begin(), gitter_opt.end());
442                                                             for(int k = 0; k < gitter.size(); k++)
443                                                                     gitter_opt.push_back(gitter[k]);
444                                                     }
445                                                     end = true;
446                                                     gitter.erase(gitter.begin(), gitter.end());
447                                             }
448                                     }
449                             }
450                     }
451                     dynamic_way2(site, neib, line_l, line_q, gitter_opt, konto);
452                     gitter_opt.erase(gitter_opt.begin(), gitter_opt.end());
453             }
454     }
455 }
456
457 void dynamic_way2(int site[], int neib[n*n][4], int line_l[], int line_q[], vector<int> & gitter, int & konto) {
458
459         vector<int> gitter_new;
460         bool end = false;
461         double random = 0.;
462         int  j= 0, gittersize = gitter.size(), site_now = gitter[gittersize-1], gittersize_delete = 0;
463
464         //Weg durchs Gitter ausgehend vom vector gitter berechnen
465         for(int i = 0; i < gittersize-1; i++) {
466                 if(gitter[i] - gitter[i+1] == -1) line_l[gitter[i] - gitter[i]/n]++;
467                 if(gitter[i] - gitter[i+1] == 1) line_l[gitter[i]-1 - gitter[i]/n]++;
468                 if(gitter[i] - gitter[i+1] == n) line_q[gitter[i]-n]++;
469                 if(gitter[i] - gitter[i+1] == -n) line_q[gitter[i]]++;
470         }
471         //Schon beim Start?
472         if(gitter[gittersize-1] != start) {
473                 //Wenn noch nicht beim Start, Weg zum Start suchen
474                 //Startpunkt speichern
475                 gitter_new.push_back(site_now);
476
477                 while(end == false) {
478                         random = rand();
479                         random = random /  (RAND_MAX + 1);
480                         //Durchs Gitter gehen
481                         if(random < 0.25 && site_now%n > 0 && line_l[site_now-1 - site_now/n] > 1) {
482                                 site_now = site_now - 1;
483                                 gitter_new.push_back(site_now);
484                         }
485                         if(random >= 0.25 && random < 0.50 && site_now/n > 0 && line_q[site_now - n] > 1) {
486                                 site_now = site_now - n;
487                                 gitter_new.push_back(site_now);
488                         }
489                         if(random >= 0.50 && random < 0.75 && site_now%n < n-1 && line_l[site_now - site_now/n] > 1) {
490                                 site_now = site_now + 1;
491                                 gitter_new.push_back(site_now);
492                         }
493                         if(random >= 0.75 && site_now/n < n-1 && line_q[site_now] > 1) {
494                                 site_now = site_now + n;
495                                 gitter_new.push_back(site_now);
496                                 }
497                         //Zurückgehen verhindern
498                         j = 0;
499                         gittersize_delete = gitter_new.size();
500                         while(j < gittersize_delete-1) {
501                                 if(gitter_new[j] == site_now) {
```

```
502                                             gittersize_delete = j+1;
503                                             gitter_new.erase(gitter_new.begin()+j, gitter_new.end()-1);
504                                             j = -1;
505                                     }
506                                     j++;
507                             }
508                             if(site_now == start) end = true;
509                     }
510                     //Weg durchs Gitter ausgehend vom vector gitter_new berechnen
511                     for(int i = 0; i < gitter_new.size()-1; i++) {
512                             if(gitter_new[i] - gitter_new[i+1] == -1) line_l[gitter_new[i] - gitter_new[i]/n]++;
513                             if(gitter_new[i] - gitter_new[i+1] == 1) line_l[gitter_new[i]-1 - gitter_new[i]/n]++;
514                             if(gitter_new[i] - gitter_new[i+1] == n) line_q[gitter_new[i]-n]++;
515                             if(gitter_new[i] - gitter_new[i+1] == -n) line_q[gitter_new[i]]++;
516                     }
517             }
518     }
519
520     void energie(int site[], int line_l[], int line_q[], bool site_food_found[], int site_food_res[], int & konto, bool & ende) {
521
522             //Energieverbrauch im Ruhezustand
523             /*for(int i = 0; i < n; i++)
524                     for(int j = 0; j < n; j++)
525                             if(site[j+i*n] == 1) konto = konto - 1;
526
527             for(int i = 0; i < n-1; i++)
528                     for(int j = 0; j < n; j++)
529                             if(line_q[j+i*n] > 1) konto = konto - 1;
530
531             for(int i = 0; i < n; i++)
532                     for(int j = 0; j < n-1; j++)
533                             if(line_l[j+i*(n-1)] > 1) konto = konto - 1;*/
534
535             for(int i = 0; i < n; i++)
536                     for(int j = 0; j < n; j++)
537                             if(site_food_found[j+i*n] == true && site_food_res[j+i*n] > 0) {
538                                     konto = konto + min(5, site_food_res[j+i*n]);
539                                     site_food_res[j+i*n] -= min(5, site_food_res[j+i*n]);
540                             }
541
542             //Keine Energie >> Zurückziehen
543             if(konto <= 0) {
544                     ende = true;
545                     konto = 0;
546                     for(int i = 0; i < n-1; i++)
547                             for(int j = 0; j < n; j++)
548                                     if(line_q[j+i*n] == 1)  line_q[j+i*n] = 0;
549
550                     for(int i = 0; i < n; i++)
551                             for(int j = 0; j < n-1; j++)
552                                     if(line_l[j+i*(n-1)] == 1) line_l[j+i*(n-1)] = 0;
553
554                     for(int i = 0; i < n; i++)
555                             for(int j = 0; j < n; j++)
556                                     if(site[j+i*n] == 1 && (line_l[j+i*n - (j+i*n)/n] == 0 && line_l[j+i*n-1 - (j+i*n)/n] == 0 && line_q[
557                                             //konto = konto + 1;
558                                             site[j+i*n] = 0;
559                                     }
560             }
561     }
```