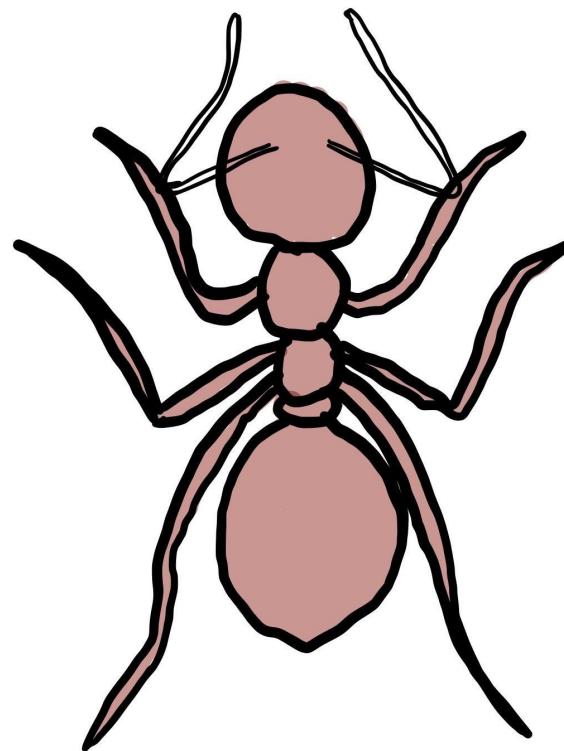


Simulation einer Ameisenkolonie

modelliert als Multiagentensystem (MAS)



Berg, Nickolay, Russo, Schopohl, Weber
Prof. Dr. habil. Zhong Li
14. Juni 2023

Gliederung

Gliederung.....	2
1. Einleitung.....	4
2. Definition MAS?.....	5
2.1. Was sind MAS?.....	5
2.2. Geschichte.....	5
2.3. Vorteile.....	5
2.4. Anwendungen.....	6
2.5. Bedeutung: Emergenz.....	6
3. Verhalten von Ameisen.....	7
3.1. Fakten über Ameisen.....	7
3.2. Einsatz von Pheromonen.....	7
3.3. Einengung der Eigenschaften für unsere Simulation.....	8
4. Simulation.....	9
4.1. Technische Rahmenbedingungen.....	9
4.2. Anfangsphase.....	9
4.3. Aufbau der Basis-Simulation.....	9
M1: Ameisen bewegen sich auf einer Ebene.....	10
M2: Futter wird eingeführt.....	10
M3: Ameisen legen Pheromonspuren.....	10
M4: Ameisen nehmen Pheromone wahr.....	10
M5: Effizienter Futterabbau.....	11
4.4. Zusätzliche Stufen.....	11
M1: Hindernisse – Wände.....	11
M2: Hindernisse – Das Doppelbrücken-Experiment.....	11
M3: Interaktion – Futter.....	12
M4: Interaktion – Hindernisse zeichnen.....	12
M5: Interaktion – Vergiften von Futterquellen.....	12
5. Implementierung.....	13
5.1. Implementierung der Basis-Simulation.....	13
Organisch wirkendes zufälliges Umherwandern.....	13
Berechnung des Bewegungswinkel durch Auswertung von Pheromonspuren.....	13
Darstellung von Pheromonen – Pheromon-Kacheln.....	14
Darstellung von Pheromonen – Pixelgenaue Implementierung mit numpy.....	14
Abschwächung der Pheromonspuren.....	17
Resultat.....	17
5.2. Implementierung der zusätzlichen Stufen.....	18
Kollisionerkennung zum Meiden von Wänden und Hindernissen.....	18
Hinzufügen von Futterquellen und Hindernissen zur Laufzeit.....	18
Vergiften von Futterquellen zur Laufzeit.....	18
5.3. Übersicht.....	19
5.4. UML Diagramm.....	19
5.5. Ablaufdiagramme.....	20
5.6. Verwaltung mit Git.....	20
6. Auswertung.....	21
6.1. Simulation Labyrinth.....	21
6.2. Simulation Doppelbrückenexperiment.....	22

Simulation mit gleicher Weglänge.....	23
Simulation mit unterschiedlicher Weglänge.....	24
7. Fazit.....	26
8. Anhang.....	27
8.1. Quellenverzeichnis.....	27
8.2. Config-Parameter.....	27
Excel Tabelle mit Simulationsergebnissen hier einfügen :):.....	27
Config für Map "Maze":.....	28

1. Einleitung

Ameisenkolonien haben eine beeindruckende Fähigkeit zur Zusammenarbeit und Koordination, die es ihnen ermöglicht, komplexe Aufgaben zu bewältigen. Die vorliegende Arbeit konzentriert sich auf die Schaffung eines Multiagentensystems, das als Ameisenkolonie fungiert und in der Lage ist, sich naturnah zu verhalten. Um ein realistisches Modell einer Ameisenkolonie zu entwickeln, wurden verschiedene Aspekte des Verhaltens von Ameisen untersucht und in die Simulation integriert. Hierzu zählen vornehmlich die Nahrungssuche und die Kommunikation über Pheromone. Basierend auf diesen Erkenntnissen wurden Agenten entworfen, die einzelne Ameisen repräsentieren und deren Verhalten durch geeignete Algorithmen gesteuert wird.

2. Definition MAS

2.1. Was sind MAS?

Bei einem Multiagentensystem handelt es sich um ein System, welches sich aus mehreren autonom handelnden intelligenten Agenten zusammensetzt, die miteinander kommunizieren und so zusammen ein Problem lösen. Ein intelligenter Agent wiederum ist ein zu intelligentem Handeln fähiger Akteur, der seine Umgebung wahrnimmt und eigenständig Entscheidungen trifft und Handlungen ausführt. Multiagentensysteme im allgemeinen Sinn spielen nicht nur in der Informatik eine Rolle. So ist ein Ameisenstaat zum Beispiel ein natürliches Multiagentensystem: Die einzelnen Ameisen entsprechen dabei den Akteuren, die mit ihrer Umwelt interagieren und durch Kommunikation miteinander Problem, wie etwa die möglichst effiziente Beschaffung von Futter, lösen. Ein wichtiger Aspekt bei der Funktionsweise eines Multiagentensystems ist hierbei, dass für die einzelnen intelligenten Agenten folgende Bedingungen gelten [1]:

- Reaktivität – sie können ihre Umgebung wahrnehmen und auf diese reagieren
- Proaktivität – sie sind zu zielgerichteten Handeln fähig und können die Initiative übernehmen
- Fähigkeit zum sozialen Handeln – sie sind in der Lage, miteinander zu interagieren

Somit handelt es sich bei einem MAS um ein dezentrales System, bei dem künstliche (zumindest in Bezug auf die Informatik) Intelligenz eine Rolle spielt.

2.2. Geschichte

Die Geschichte der MAS ist eng verbunden mit der des Themenfeldes der verteilten / dezentralen künstlichen Intelligenz, welches wiederum ein Teilstück der künstlichen Intelligenz ist.

Letztere trat als Forschungsgebiet erstmals in den 1950er Jahren in Erscheinung. Der sogenannte "Dartmouth Workshop" (vollständiger Name: "Dartmouth Summer Research Project on Artificial Intelligence"), an dem über ein Dutzend Wissenschaftler und Mathematiker sich zwei Monate lang über verschiedene mit künstlicher Intelligenz im Zusammenhang stehende Themen austauschen, wird oft als der Grundstein der KI-Forschung angesehen. Auch vor diesem Event gab es allerdings schon relevante Entwicklungen wie zum Beispiel die Entwicklung erster Schach- und Dame-Programme oder nicht zuletzt die Veröffentlichung des wegweisenden Papers "Computing Machinery and Intelligence" von Alan Turing, in dem dieser das erste Mal den später nach ihm benannten "Turing-Test" beschreibt – einen Test, der feststellen soll, ob eine Maschine über (eine von der menschlichen nicht unterscheidbaren) künstlichen Intelligenz verfügt.

Seit ca. 1975 hat sich das Gebiet der dezentralen künstlichen Intelligenz als Teilgebiet der KI-Forschung herauskristallisiert. Es setzt sich wiederum aus den Bereichen verteiltes Problemlösen und Multiagentensysteme zusammen, wobei letzterer seitdem zunehmend an Bedeutung gewann.

2.3. Vorteile

Da die einzelnen Agenten autonom und dezentral agieren, besteht ein großer Vorteil eines MAS darin, dass beim Ausfall bzw. der Fehlfunktion einzelner Akteure das Gesamtsystem kaum bis fast gar nicht in Mitleidenschaft gezogen wird. In diesem Sinne ist ein MAS, was den Aspekt der Robustheit betrifft, mit anderen dezentralen Systemen in der Informationstechnik, wie etwa dem Internet, vergleichbar.

Ein weiterer Vorteil, der sich aus dem selbstständigen Handeln der einzelnen Akteure ergibt, besteht darin, dass sich die für das MAS benötigte Rechenleistung dadurch relativ einfach auf verschiedene Systeme (wenn es sich bei dem MAS um ein verteiltes System handelt) bzw. via Threading auf verschiedene Prozessoren (wenn das MAS auf einem einzelnen System läuft) aufteilen lässt.

2.4. Anwendungen

MAS finden in vielseitigen Themengebieten Verwendung, von denen im Folgenden einige aufgeführt werden:

- zur Verhaltenssimulation von Agenten verschiedenster Natur, wie z.B.:
 - Tiere:
 - Tierschwärme (wie es bei unserem Projekt der Fall ist)
 - Tierpopulationen
 - wirtschaftliche Akteure
 - Menschen:
 - Verhalten im Verkehr (unter anderem zum Testen von Software für selbstfahrende Autos nützlich)
 - Verhalten in der Gesellschaft
 - Verhalten bei Notsituationen (z.B. Evakuierungssimulationen)
- allgemein im Bereich der künstlichen Intelligenz und Schwarmintelligenz
- zur Produktionsplanung und -steuerung (PPS) in der Produktionstechnik
- zur Planung von Smart Grids
- zur Erstellung von 3D-Grafiken zur Nutzung in Computerspielen und Filmen
- zur Umsetzung von Webcrawlern

2.5. Emergenz

Unter dem Begriff "Emergenz" versteht man im Allgemeinen die Entstehung neuer Eigenschaften eines Systems, die durch das Zusammenwirken seiner einzelnen Elemente hervorgerufen werden. Wichtig ist hierbei, dass die einzelnen Elemente nicht bereits über die besagten emergenten Systemeigenschaften verfügen. Diese Definition ist auch für das Themengebiet der Informatik gültig, wenn man informationstechnische Systeme und deren Elemente betrachtet.

Die emergente Eigenschaft eines realen bzw. simulierten Ameisenstaates, dazu in der Lage zu sein, Ameisenstraßen als optimale Wege zur Nahrungsbeschaffung zu bilden, dient dabei als

anschauliches Beispiel aus der Biologie bzw. Informatik. Eine einzelne Ameise wäre dazu niemals in der Lage, nur das Gesamtsystem kann diese Aufgabe mittels Kommunikation der einzelnen Ameisen untereinander lösen.

3. Verhalten von Ameisen

3.1. Fakten über Ameisen

Ameisen sind erstaunlich gut darin, sich zu orientieren und effiziente Wege zu Nahrungsquellen zu finden. Dabei benutzen verschiedene Ameisenarten zum Teil ganz unterschiedliche Techniken.

Die meisten Ameisen verwenden Pheromone, um Spuren zu markieren und ihre Artgenossen zu leiten. Diese chemischen Duftspuren kennzeichnen den Weg zu Futterquellen oder dem Nest. Andere Ameisen folgen diesen Pheromonen und verstärken sie gegebenenfalls.

Manche Ameisen können auch die Position der Sonne nutzen, um ihre Richtung zu bestimmen. Sie besitzen eine interne biologische Uhr und eine Art Sonnenkompass, die es ihnen ermöglichen, die Position der Sonne während des Tages zu erkennen und als Orientierungshilfe zu nutzen.

Zudem können einige Ameisenarten auch das Magnetfeld der Erde wahrnehmen und es zur Navigation verwenden. Sie besitzen magnetische Partikel in ihrem Körper, die es ihnen ermöglichen, die Ausrichtung des Erdmagnetfeldes zu erkennen und als zusätzliche Orientierungshilfe zu nutzen.

Ameisen können auch ihre Schritte zählen, um sich zu orientieren. Sie messen die Anzahl der Schritte, die sie von ihrem Nest entfernt sind, und verwenden diese Informationen, um den Weg zurückzufinden.

Außerdem nutzen einige Ameisenarten auch visuelle Landmarken, um sich zu orientieren. Sie können sich an charakteristischen Merkmalen in ihrer Umgebung wie Bäumen, Steinen oder anderen markanten Objekten orientieren, um den Weg zu finden. [2] [3] [4]

3.2. Einsatz von Pheromonen

Zur Verdeutlichung der Komplexität von Pheromonspuren, sei hier ein kurzer (übersetzter) Ausschnitt aus der Einleitung der Arbeit "Trail Pheromones: An Integrative View of Their Role in Social Insect Colony Organization" [5] gegeben.

Seit über 200 Jahren ist bekannt, dass Ameisen bei der Nahrungssuche Duftstoffe - Pheromone - abgeben, genauso wie viele andere Insekten. Arbeiterinnen, die eine Futterstelle finden, können bei der Rückkehr zum Nest Trailpheromone ablegen. Die Pheromone wirken als positive Rückkopplung und lenken Nestgenossen zur Ressource. Diese klassische Vorstellung von Pheromonspuren bei sozialen Insekten war die Inspiration für die Ameisenkolonienoptimierung (ACO), eine Technik zur Berechnung von Lösungen für analytisch unlösbare Probleme (auf diese werden wir aber in unserer Arbeit nicht eingehen). Trailpheromone werden in vielen Zusammenhängen des kolonialen Lebens verwendet, abgesehen vom aufspüren von Nahrungsquellen. Sie werden bei der Suche nach einem neuen Nest verwendet, zur Rekrutierung, zum Kampf oder zur Flucht, oder zur Lenkung des Tunnelbaus.

Die Spuren können aus mehreren Pheromonen mit unterschiedlichen Eigenschaften bestehen, was es den Kolonien ermöglicht, externe Erinnerungen zu bilden oder sogar "die Vergangenheit zu riechen".

Dieser kurze Ausschnitt verdeutlicht, wie vielfältig alleine bei Ameisen der Einsatz von Pheromonen ist. Wir werden uns in unserer Arbeit größtenteils auf das Aufspüren von Futterquellen beschränken.

3.3. Einengung der Eigenschaften für unsere Simulation

In unserer Simulation werden wir nur einige stark vereinfachte Prinzipien der Ameisenorientierung einsetzen. Hauptsächlich wird sich unser Algorithmus auf das Legen und Erspüren von Pheromonen beschränken. Dabei werden einige Aspekte aus der obigen Arbeit berücksichtigt, z.B. dass die Ameisen Pheromonspuren, die in ihrer Laufrichtung liegen, höher gewichten (falls vorhanden) als Spuren, die zur Seite wegführen.

Außerdem verwenden wir eine äußerst reduzierte Art der Landmarken-Erkennung, in der Ameisen ihr Nest wahrnehmen, wenn sie sich diesem bis auf eine bestimmte Entfernung genähert haben – sie sind dann nicht mehr auf Pheromonspuren angewiesen.

4. Simulation

4.1. Technische Rahmenbedingungen

Wir haben uns dazu entschieden, unsere Simulation in Python zu schreiben und als Grafikbibliothek Pygame zu nutzen. Zum einen hat das die Arbeit in der Gruppe erleichtert, weil Python der größte gemeinsame Nenner war, zum anderen entfallen dadurch auch weitere Hürden durch unterschiedliche Betriebssysteme. Außerdem ist es so auch für Außenstehende leichter, unser Programm zu verstehen und auszuführen.

Natürlich gibt es dafür Einschränkungen in der Performanz unserer Simulation, die mit einer maschinennahen Sprache wie C weniger stark auftreten.

Unsere Simulation ist in diesem öffentlichen GitHub Repository zu finden:

<https://github.com/wschoopohl/fuh-ants>

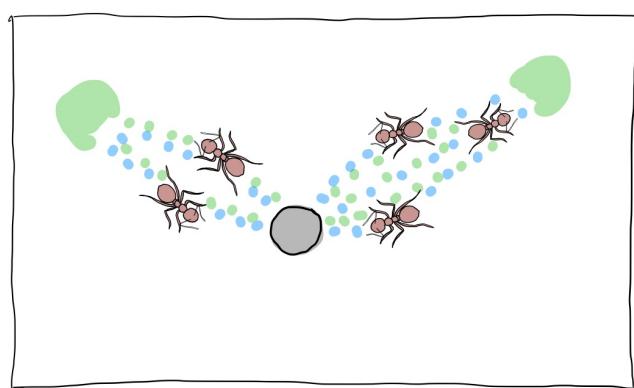
4.2. Anfangsphase

Um eine möglichst vielschichtige Blickweise auf das Thema zu ermöglichen, haben wir in einer Anfangsphase von etwa einem Monat unabhängig voneinander an vier verschiedenen Quellcode-Basen gearbeitet.

Dieser Ansatz hat uns geholfen, Gemeinsamkeiten in der Umsetzung festzustellen, als auch unterschiedliche Lösungen und deren Bewertung hinsichtlich Korrektheit, Effektivität und Darstellung zu ermöglichen.

Zu guter Letzt haben wir uns dann auf die Quellcode-Basis geeinigt, die uns die meisten Vorteile zu bieten schien und Stück für Stück Elemente aus anderen Codebasen darin eingearbeitet, wo es uns sinnvoll erschien.

4.3. Aufbau der Basis-Simulation

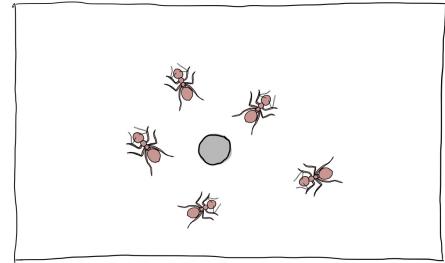


Unser erstes Ziel war das Erstellen einer Basis-Simulation. Sie sollte noch keine Wände enthalten, aber ansonsten voll funktionsfähig sein. Das bedeutet, Ameisen laufen zufallsgesteuert auf einer Ebene umher, nehmen Futter auf, wenn sie darauf stoßen und hinterlassen Pheromone für ihre Artgenossen. Des Weiteren sind sie in der Lage, Pheromone wahrzunehmen und ihre Laufrichtung entsprechend anzupassen.

Im Folgenden werden die Schritte zur Erreichung dieses Basisziels erklärt.

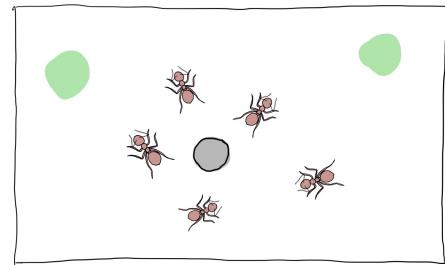
M1: Ameisen bewegen sich auf einer Ebene

Zuerst sollen sich die Ameisen zufallsgesteuert auf einer Ebene bewegen. Die Ameisen ändern dafür ihre Ausrichtung zufällig um einige Grad und gehen dann ein paar Schritte in diese Richtung. Danach wiederholt sich der Prozess.



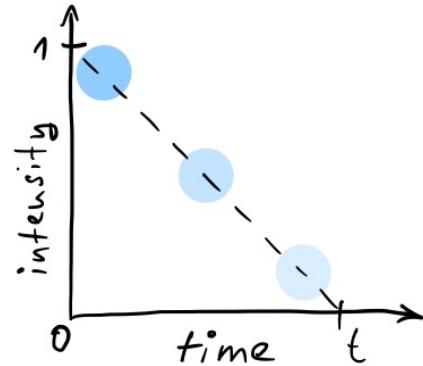
M2: Futter wird eingeführt

Im nächsten Schritt werden Nahrungshaufen eingeführt. Jeder Haufen enthält eine begrenzte Menge an Nahrungsteilchen. Wenn eine Ameise einen Haufen erreicht, nimmt sie ein Nahrungsteilchen auf und wenn sie mit einem Nahrungsteilchen beladen auf das Nest stößt, liefert sie es dort ab.



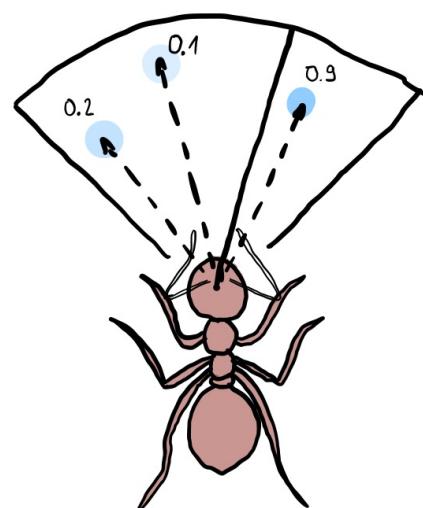
M3: Ameisen legen Pheromonspuren

Im Anschluss werden Pheromone eingeführt. Die Ameisen hinterlassen in bestimmten Abständen zwei Arten von Pheromonen. Wenn sie keine Nahrung tragen, hinterlassen sie eine "Zurück zum Nest" Pheromonspur und wenn sie Nahrung tragen, hinterlassen sie eine "Zur Nahrung" Pheromonspur. Die Intensität der sich in der Simulationswelt befindlichen Pheromonspuren nimmt über die Zeit linear ab.



M4: Ameisen nehmen Pheromone wahr

Im letzten Schritt soll die Fähigkeit implementiert werden, Pheromone wahrzunehmen. Die Ameisen haben ein Sichtfeld, in dem sie Pheromone erkennen, wobei die verschiedenen (von der Ameise zu den jeweiligen Pheromonmarkern verlaufenden) Vektoren durch die Intensität und ihre Winkelabweichung von der Ausrichtung der Ameise gewichtet werden und aus ihnen dann ein neuer Bewegungswinkel berechnet wird.



M5: Effizienter Futterabbau

Die in M1 bis M4 umgesetzten Eigenschaften und Verhaltensweisen der Ameisen sollen nach Bestimmung passender Parameter letztendlich wie erwünscht zur Folge haben, dass die Ameisen zusammen dazu in der Lage sind, effizienten Futterabbau mithilfe von dazu optimierten Ameisenstraßen zu bewältigen.

4.4. Zusätzliche Stufen

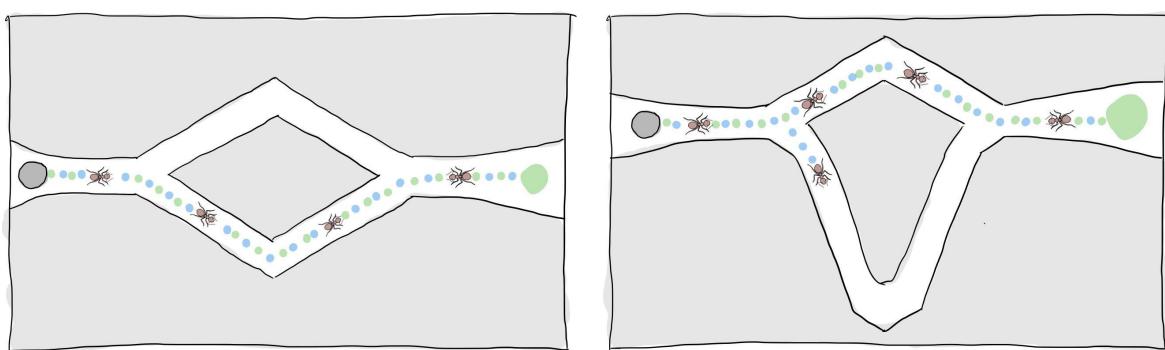
M1: Hindernisse – Wände

In einer zusätzlichen Stufe werden Wände eingeführt, um Hindernisse zu schaffen. Hierzu wird eine Kollisionserkennung zwischen Ameisen und Wänden implementiert. Dadurch werden die Simulationen noch interessanter, da die Ameisen nun ihre Fähigkeiten zeigen können, komplexere Probleme zu lösen. Zum Testen der Implementierung dieser Stufe wird eine Labyrinth-Welt erstellt.

M2: Hindernisse – Das Doppelbrücken-Experiment

Wir sind daran interessiert, die in M1 umgesetzte Funktionalität zu verwenden, um das Verhalten unserer Ameisen in einem Doppelbrücken-Experiment zu beobachten [6]. Kurz gesagt werden die Ameisen mit zwei Situationen konfrontiert, die wir als separate "Welten" für die Simulation anlegen:

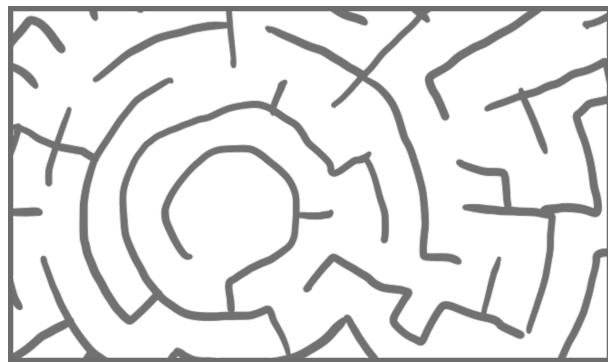
Im ersten Szenario, mit zwei Pfaden gleicher Länge, wird erwartet, dass die Ameisen im Laufe der Zeit zufällig einen der beiden Pfade wählen. Bei wiederholter Durchführung dieses Experiments sollte die Wahrscheinlichkeit für beide Pfade gegen 50% konvergieren.



Im zweiten Szenario ist einer der Pfade deutlich kürzer als der andere. Natürlich wird erwartet, dass die Ameisen im Laufe der Zeit den kürzeren Pfad wählen.

M3: Interaktion – Futter

In dieser Stufe wird zusätzlich die Möglichkeit eingebaut, während der laufenden Simulation per Mausklick Futterquellen variabler Größe hinzuzufügen. Somit kann zum Beispiel beobachtet werden, wie zügig die Ameisen dazu in der Lage sind, eine neue Futterquelle auszunutzen, die sich näher an ihrem Nest befindet als die bisherigen Optionen.



M4: Interaktion – Hindernisse zeichnen

Als weitere Interaktionsmöglichkeit wird hier realisiert, dass man zur Laufzeit Hindernisse einzeichnen kann, etwa um einen optimalen Weg zu einer Futterquelle zu blockieren und die Ameisen dazu zu zwingen, einen neuen kürzesten Weg zu finden.

M5: Interaktion – Vergiften von Futterquellen

Es soll nun zusätzlich die Möglichkeit geben, existierende Futterquellen zur Laufzeit zu "vergiften". Ameisen können dabei vergiftetes Futter nicht auf direkte Weise als solches erkennen, bevor sie es aufnehmen. Bereits vergiftete Ameisen können jedoch eine neue Art von "Gift-Pheromonen" platzieren, um ihre Artgenossen so vor der Gefahr zu warnen.

5. Implementierung

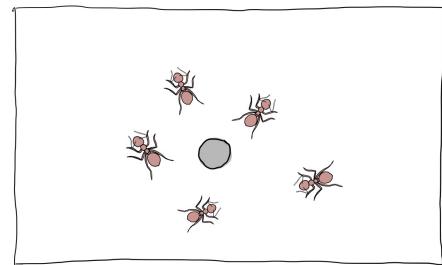
5.1. Implementierung der Basis-Simulation

In diesem und dem nächsten Abschnitt wird genauer ausgeführt, welche Probleme und Schwierigkeiten bei der konkreten Umsetzung der in 4.3 und 4.4 aufgetreten sind und wie wir diese bewältigt haben.

Organisch wirkendes zufälliges Umherwandern

Das erwünschte zufällige Umherwandern der Ameisen gelang nicht auf Anhieb so, wie wir es uns vorgestellt hatten: Der erste Ansatz bestand darin, dass jede Ameise bei jedem Berechnungsschritt bzw. Richtungsupdate einen zufälligen positiven oder negativen Winkel mit festgelegtem Maximalbetrag bestimmt, den sie dann zu ihrer gegenwärtigen Bewegungsrichtung addiert. Da diese Richtungsupdates allerdings viele Male pro Sekunde erfolgen müssen (zumindest müssen sie das im späteren Verlauf der Implementierung, sobald die Orientierung an Pheromonen umgesetzt ist) und eine Kursänderung nach links genauso wahrscheinlich wie eine Kursänderung nach rechts ist, laufen die Ameisen keine schönen Kurven nach links oder rechts, sondern zappeln stattdessen im Sekundenbruchteil-Takt hin und her und bewegen sich dabei im Mittel trotzdem meist mehr oder weniger geradeaus. Die Lösung für dieses Problem bestand darin, den zufälligen Winkel, den die Ameisen zur Kursänderung verwenden, nur deutlich seltener (und nicht bei jedem Richtungsupdate) neu zu bestimmen – eine zufällig gewählte Zeitspanne zwischen 0,5 und 1,5 Sekunden lieferte zufriedenstellende Ergebnisse. Die besagte Zeitspanne wird jedes Mal neu zufällig gewählt, damit nicht alle Ameisen gleichzeitig ihren Kurs ändern, was unnatürlich wirken würde.

Im Bezug auf organisch wirkende Bewegungen haben wir auch mit einer Variante herumexperimentiert, in der Ameisen beschleunigen und langsamer werden können, je nachdem ob sie praktisch geradeaus laufen oder gerade eine enge Kurve beschreiben, statt sich immer mit einer gleichbleibenden Geschwindigkeit fortzubewegen. Da diese Variante die Simulation nicht realistischer erscheinen ließ, den Code für das Folgen von Pheromonspuren und das Vermeiden von Wänden / Hindernissen aber unnötig verkomplizierte, haben wir sie im späteren Verlauf verworfen.



Berechnung des Bewegungswinkels durch Auswertung von Pheromonspuren

Die Berechnung des Bewegungswinkels haben wir letztendlich so umgesetzt: Für jeden sich im Sichtfeld befindlichen Pheromon-Marker (diese sind entweder pixelgenau oder als "Pheromon-Kacheln" realisiert, mehr dazu später) mit Pheromon-Intensität > 0 wird der Vektor von der Ameise zu dieser Zelle bestimmt und aus allen dabei erhaltenen Vektoren wird ein Gesamtvektor als gewichteter Mittelwert gebildet. Der Winkel dieses Gesamtvektors ist dann der

neue Bewegungswinkel der Ameise. Was die Gewichtung der einzelnen Vektoren angeht, haben wir uns, nachdem wir verschiedene Möglichkeiten getestet haben, für folgende Variante entschieden: Die Länge des Vektors (also die Entfernung der Ameise zum Pheromon-Marker) spielt keine Rolle, die Richtung des Vektors hingegen schon - die Gewichtung des Einzelvektors ist maximal, wenn seine Richtung mit der Blickrichtung der Ameise übereinstimmt und damit im Zentrum des Sichtfeldes liegt und minimal, wenn er gerade so am Rand des Sichtfeldes noch enthalten ist.

Darstellung von Pheromonen – Pheromon-Kacheln

Wir mussten feststellen, dass die von uns zunächst umgesetzte pixelgenaue Darstellung von Pheromonspuren sich beim Implementieren der Auswertung von Pheromonen durch die Ameisen als nicht performant genug erwies: Wählt man etwa für das Pheromon-“Sichtfeld” der Ameisen den Winkel $\beta = 120^\circ$ und die Sichtweite $d = 100$ Pixel, so enthält der auszuwertende Kreissektor bereits $(\beta/360^\circ) * \pi d^2 \approx 10.472$ Pixel, die es bei jedem Richtungsupdate, also einige Male pro Sekunde, auszuwerten gilt.

Zur Lösung dieses Problems wurde eine räumliche Hashing-Struktur eingeführt, welche die Gesamtfläche der Simulation in einzelne Kacheln zerteilt und die enthaltenen Pheromone verwaltet. Pro Kachel ist nur noch ein Pheromon jeden Typs erlaubt; beim Ablegen eines neuen Pheromons auf dieser Kachel wird lediglich die Intensität des bereits enthaltenen Pheromons verstärkt.

Will eine Ameise ihre Richtung anpassen, erhält sie durch das räumliche Hashing in Kombination mit ihrer aktuellen Position nur eine Teilmenge von Pheromonen geliefert, die in ihrem Sichtfeld liegen könnten. Pheromone, die (weit) außerhalb ihres Sichtfeldes liegen, werden von vornherein ignoriert.

Durch diese Maßnahmen kann die Simulation bedeutend effizienter ausgeführt werden, ohne dass sich das Verhalten der Ameisen dadurch ändert – zumindest solange die Kacheln klein genug sind, um den Ameisen genug Bezugspunkte für präzise Richtungsupdates zu liefern. Als konkreten Wert für die Seitenlänge der Kacheln, `PheromoneMapTileSize`, haben wir uns nach einem Testen auf $s = 15$ Pixel festgelegt. Die Ameisen müssen nun bei jedem Richtungsupdate bei den vorher angenommenen Werten $\beta = 120^\circ$ und $d = 100$ Pixel nur noch $(\beta/360^\circ) * \pi(d/s)^2 \approx 47$ Kacheln statt über 10.000 Pixel auswerten, was die für die Simulation benötigte Rechenleistung natürlich erheblich reduziert.

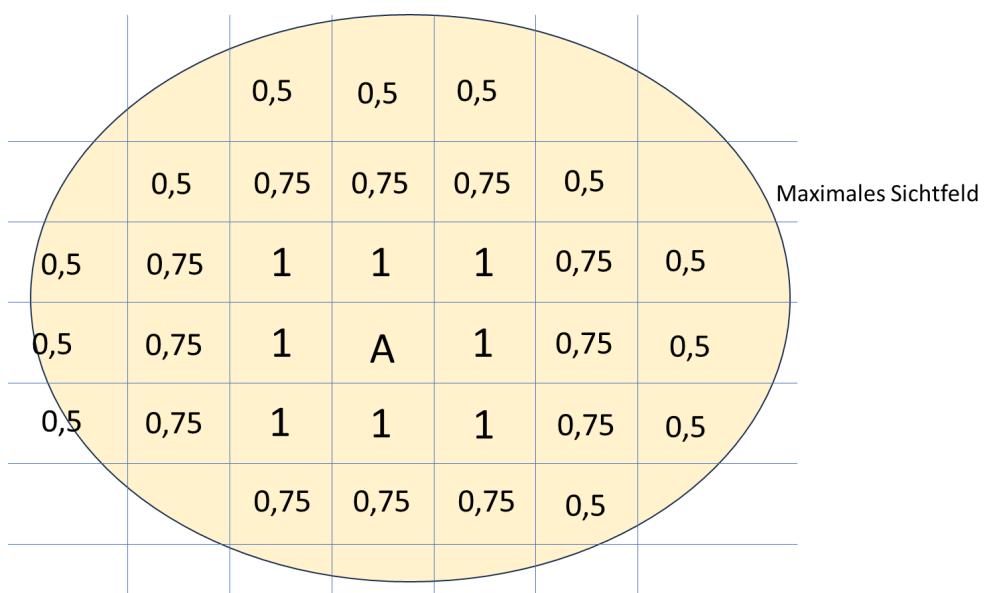
Darstellung von Pheromonen – Pixelgenaue Implementierung mit numpy

Parallel dazu wurde noch eine verbesserte, pixelgenaue Implementierung realisiert, welche sich auf Matrizenoperationen der `numpy` Bibliothek stützt, um ein schnelleres Auswerten der Pheromone zu ermöglichen. Dafür wurden mehrere Matrizen definiert, die folgend beschrieben werden.

Die Intensitätsmatrix der Pheromone besitzt die Größe der Simulation/Spielfeldes und enthält als Einträge die Mächtigkeit der Pheromone.

0	0	1	2	3	4	0
0	3	2	2	2	2	0
2	0	0	1	1	1	2
4	1	1	0	1	3	5
3	5	1	0	2	5	6
0	0	0	0	0	1	0

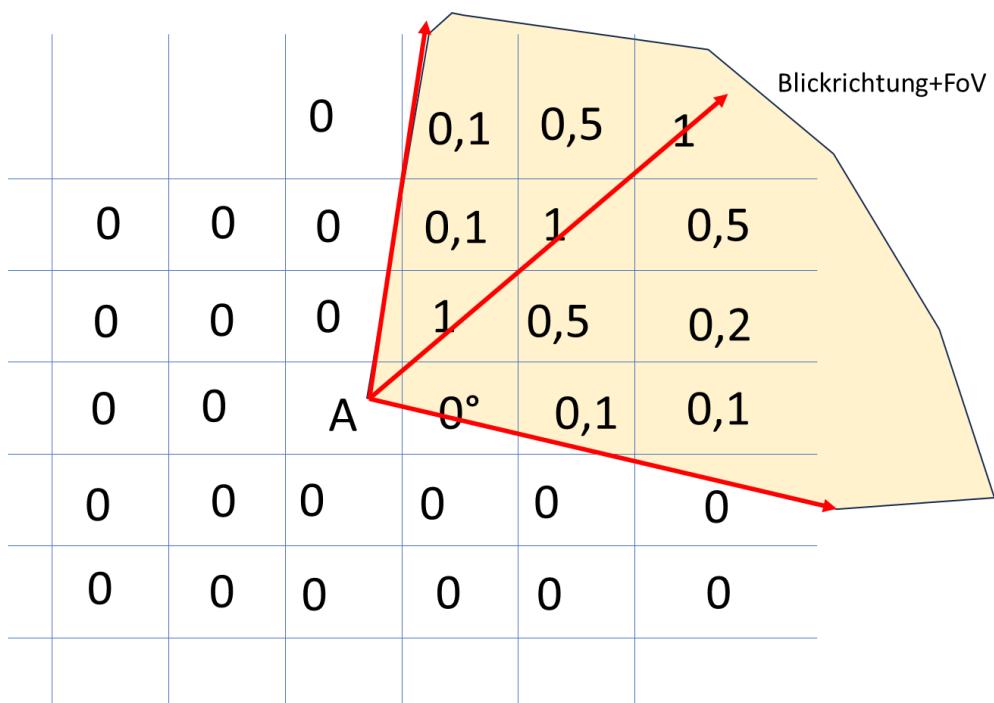
Die Distanzmatrix hat die Form $2 \cdot \text{Sichtweite} + 1$, mit den Einträgen eins Minus eins durch Distanz zur Mitte der Matrize geteilt durch die Sichtweite. Die Matrize wird nur einmal berechnet, was während der Initialisierung des Programms erfolgt.



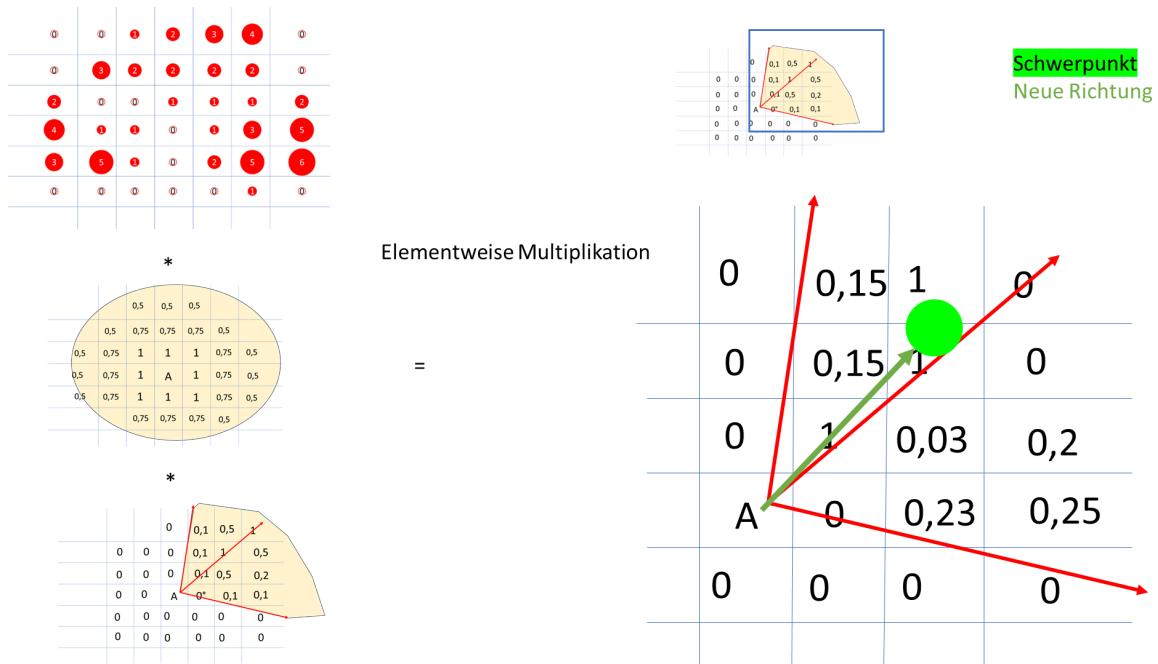
Die Winkelmatrix hat die Form $2 \times \text{Sichtweite} + 1$, mit den Einträgen des Winkels bezüglich der Mitte der Matrix und der x-Achse. Initialisierung und Berechnung erfolgt ebenfalls einmalig zum Programmstart.

		90°			45°
135°		90°		45°	
	135°	90°	45°	22,5°	
180°	180°	A	0°	0°	0°
	-135°	-90°	-45°	-22,5°	
-135°		-90°		-45°	

Pro Ameisenschritt wird die Winkelmatrix an die aktuelle Blickrichtung angepasst (bzw. eine Kopie von ihr). Die Einträge werden um den Blickwinkel subtrahiert. Alle Einträge kleiner als die Blickweite (außerhalb des Sichtfeldes) werden auf Null gesetzt. Alle Werte innerhalb des Sichtfeldes werden bezüglich ihrer Abweichung von der aktuellen Blickrichtung verändert, wobei die aktuelle Blickrichtung den Wert 1 erhält und zum Rand des Sichtfeldes kleiner wird.



Die Intensität, die Distanz und die Blickmatrix werden elementweise miteinander multipliziert. Alle Werte außerhalb des Sichtkegels werden dadurch Null, alle anderen Werte werden bezüglich ihrer relativen Position zur Ameise gewichtet. Aus diesen Werten wird der Schwerpunkt ermittelt. Der Richtungsvektor von Ameise zum Schwerpunkt ist die neue Bewegungsrichtung.



Abschwächung der Pheromonspuren

Wir mussten zudem feststellen, dass unsere Ameisen dazu neigen, anderen ziellos umherwandernden Artgenossen zu folgen, die schon vor langer Zeit vom Nest losgezogen oder auf Futter gestoßen sind. Um diesen Effekt einzudämmen, haben wir zusätzlich zur für M3 bereits eingeführten zeitlichen Abnahme der Stärke von Pheromonmarkern einen weiteren die Intensität

von Pheromonen betreffenden Mechanismus eingeführt: Ameisen sollten schwächere Pheromonspuren hinterlassen, je länger sie in einem Zustand (als Zustand gilt hier: "mit Futter" / "ohne Futter") unterwegs sind. Jedesmal wenn eine Ameise auf das Nest oder eine Futterquelle stößt, wird dieser Effekt zurückgesetzt und die Ameise hinterlässt wieder eine Spur mit maximaler Stärke.

Resultat

Abschließend wurde der bisherige Stand in einer Vielzahl von Testläufen beobachtet und bewertet. Die für die Stufen M1 bis M4 umgesetzten Eigenschaften und Verhaltensweisen der Ameisen hatten nach dem Lösen obiger Probleme und der Bestimmung passender Parameter letztendlich wie erwünscht zur Folge, dass die Ameisen zusammen dazu in der Lage sind, effizienten Futterabbau mithilfe von zu diesem Zweck optimierten Ameisenstraßen zu bewältigen.

5.2. Implementierung der zusätzlichen Stufen

Kollisionserkennung zum Meiden von Wänden und Hindernissen

Die Erkennung und Vermeidung von Wänden durch die Ameisen wird algorithmisch durch Abfrage einer Bitmaske erreicht, die für jeden Punkt der Karte in "Wand" / "Keine Wand" einordnet. Bei Erkennung einer Kollision wird durch iteratives Anpassen der Bewegungsrichtung die betragsmäßig kleinste Drehung gesucht, bei der keine Kollision auftritt. Dieses Verfahren vermeidet das Abprallen der Ameisen an der Wand mit dem damit verbunden Ping-Pong Effekt, führt aber zu einer gewissen Wandfixierung der Ameisen, da diese sich nun teilweise sehr deutlich entlang der Wände bewegen.

Die Auswertung des sich der hier umgesetzten Kollisionserkennung bedienenden Labyrinth-Welt-Tests und des Doppelbrücken-Experiments wird im Auswertungs-Kapitel ausführlich erläutert.

Hinzufügen von Futterquellen und Hindernissen zur Laufzeit

Bei der Umsetzung der interaktiven zusätzlichen Stufen M3 bis M5 haben wir uns das in Pygame bereits enthaltene Event-Management zunutze gemacht, mit dem sich Maus- und Tastatur-Events erkennen und zuordnen lassen: Im Haupt-Renderloop muss dazu bei jedem Durchlauf geprüft werden, ob etwa eine Maustaste gedrückt oder losgelassen wurde oder eine Tastatureingabe erfolgt ist.

Futterquellen werden in unserer Umsetzung mittels Linksklick hinzugefügt. Damit sie nicht innerhalb von Wänden platziert werden können, wird beim Drücken der linken Maustaste zunächst die Mausposition mit einer Maske der vorhandenen Wände und Hindernisse abgeglichen. Die Größe der hinzugefügten Futterquelle wird dadurch bestimmt, wie lange man die linke Maustaste gedrückt hält: Je länger, desto größer die resultierende Futterquelle. Wir haben zudem eine

Maximalgröße für Futterquellen eingeführt, damit diese nicht durch langes Gedrückthalten extrem groß werden können.

Das Hinzufügen von Hindernissen ist mittels Rechtsklick möglich. Durch Gedrückthalten lassen sich damit neue Wände einziehen. Die neu generierten Hindernisse werden dabei der entsprechenden Maske hinzugefügt, damit sie von den Ameisen auch berücksichtigt werden.

Vergiften von Futterquellen zur Laufzeit

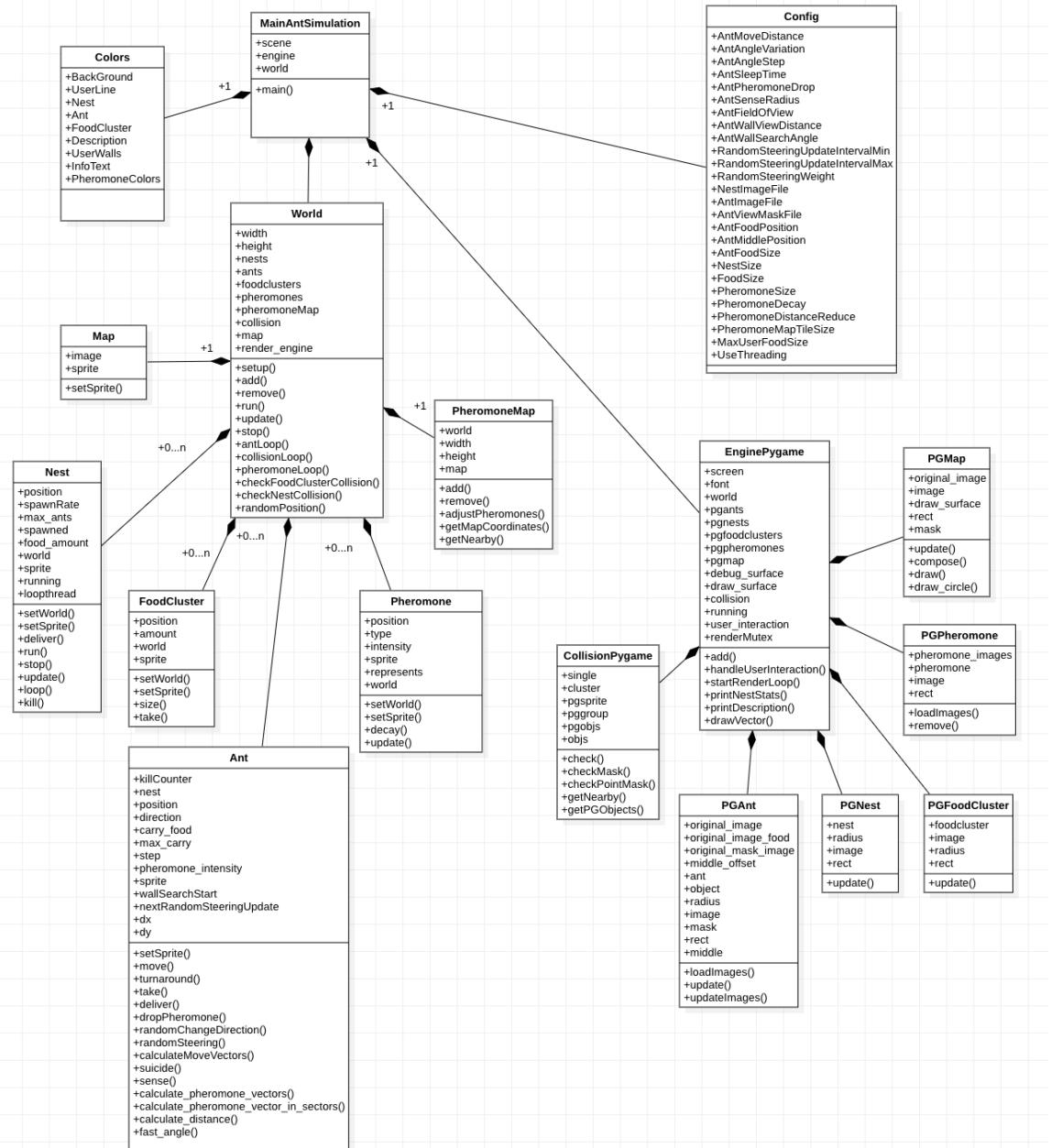
Bei der Umsetzung dieser Funktionalität haben wir uns für die folgenden Implementationsdetails entschieden: Die Ameisen können die vergiftete Futterquelle an sich vor der Nahrungsaufnahme nicht als solche erkennen. Sie sind allerdings dazu in der Lage, festzustellen, dass sie vergiftet worden sind und warnen dann ihre Stammmitglieder durch das Platzieren der für diese Funktionalität implementierten "Giftpheromone". Insgesamt sorgt das Gift für die folgenden Verhaltensänderungen bei den Ameisen:

- Vergiftete Ameisen sterben nach einigen Sekunden.
- Vergiftete Ameisen sind nicht mehr in der Lage, Pheromonspuren zu folgen und wandern somit ziellos umher.
- Vergiftete Ameisen platzieren statt Pheromonen zum Signalisieren von Futter nun solche zum Signalisieren von Gift.
- Andere Ameisen meiden die vergiftete Futterquelle, indem sie sich von den Giftpheromonen weg bewegen.

Das letztgenannte Verhalten wird konkret so umgesetzt: Zusätzlich zu den Pheromon-Layern, die zum Finden von Nahrung bzw. des Heimwegs verwendet werden, gibt es nun ein drittes Layer für die Giftmarker. Die Giftpheromone werden genau wie anderen Pheromone auch von einer Ameise in einem kreissektorförmigen Bereich vor ihr wahrgenommen und auf die gleiche Weise zu einem gewichteten Gesamtvektor verrechnet. Die Maximaldistanz – also der Radius des Kreissektors – lässt sich dabei zudem getrennt von der für die anderen Pheromone geltenden Distanz anpassen und wurde niedriger gewählt. Nimmt eine Ameise nun die Präsenz von Giftpheromonen wahr, so ignoriert sie alle anderen (Futter/Nest)-Pheromoninformationen und steuert so weit wie möglich in die der Richtung des berechneten Giftmarker-Vektors entgegengesetzte Richtung, also entweder so scharf nach links oder nach rechts, wie es ihre Drehgeschwindigkeit erlaubt. Diese "extreme" Reaktion der Ameisen auf Giftpheromone hat aber auch ihre Nachteile: Befindet sich etwa eine vergiftete Futterquelle in der Nähe des kürzesten Pfades zu einer anderen Futterquelle, so sind die Ameisen nicht in der Lage, diesen optimalen Pfad zu nehmen – sie müssen stattdessen einen Umweg auf sich nehmen, da sie die Umgebung der vergifteten Quelle meiden.

5.3 UML Diagramm

ANPASSEN WENN FINALE VERSION BEKANNT



5.3. Ablaufdiagramme

TO-DO

5.4. Verwaltung mit Git

TO-DO

6. Auswertung

6.1. Simulation Labyrinth

Bei der richtigen Wahl der Parameter können Ameisen auch ein Szenario namens "Labyrinth" lösen. Dabei finden die Ameisen den Weg von der Futterquelle bis zum Nest und bestimmen dabei den optimalen Weg durch das Labyrinth.

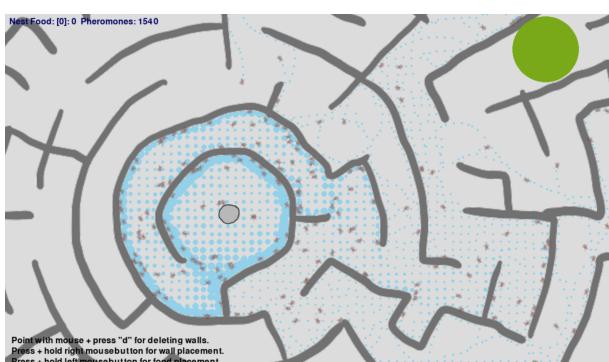
Das Verhalten von Ameisen in Bezug auf die Navigation und das Lösen von Problemen ist ein faszinierendes Phänomen, das als Inspiration für verschiedene Optimierungsansätze verwendet wurde. Ein solcher Ansatz ist der sogenannte Ameisenalgorithmus oder auch ACO (Ant Colony Optimization). Dieser Algorithmus basiert auf dem Schwarmverhalten von Ameisen und wird oft zur Lösung von Optimierungsproblemen eingesetzt.

Im Falle eines Labyrinths können Ameisen mithilfe des Ameisenalgorithmus den kürzesten Weg von der Futterquelle zum Nest finden.

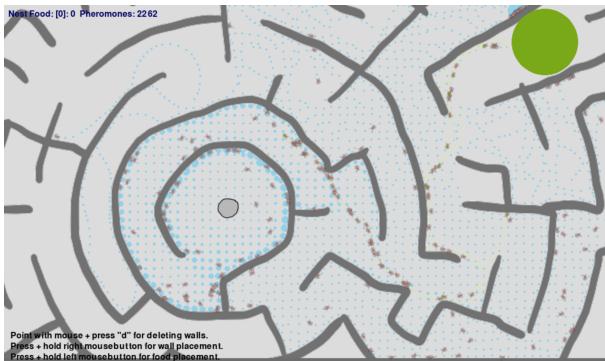
Während sich die Ameisen durch das Labyrinth bewegen, verstärken sie die Duftspuren auf den kürzesten Wegen, da sie schneller stärker werden (vgl. das Doppelbrücken-Experiment in Kapitel 6.2). Dadurch entsteht im Laufe der Zeit eine immer stärker werdende Duftspur, die den anderen Ameisen als Hinweis für den kürzesten Weg dient. Auf diese Weise können die Ameisen gemeinsam den optimalen Pfad zum Nest finden.

Die Wahl der Parameter im Algorithmus ist entscheidend, um das Labyrinth erfolgreich zu lösen. Dazu gehören beispielsweise die Menge der freigesetzten Pheromone, die Verdunstungsgeschwindigkeit (PheromoneDecay) der Duftspuren, die Anzahl der Ameisen und die Sichtweite der Ameisen. Durch die sorgfältige Anpassung dieser Parameter kann der Algorithmus so konfiguriert werden, dass er das Labyrinth effektiv löst. Die gewählten Parameter, die in der vorliegenden Simulation zum Erfolg führen, befinden sich im Anhang.

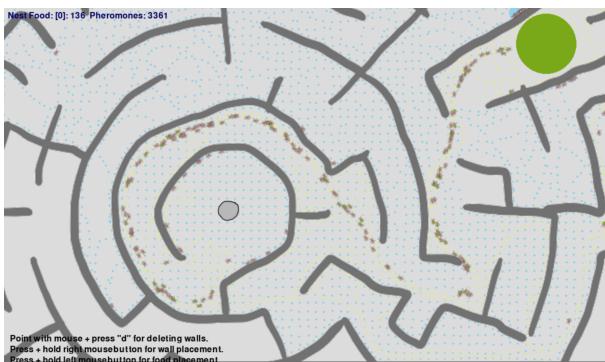
Es ist wichtig zu beachten, dass der Ameisenalgorithmus ein vereinfachtes Modell des Verhaltens von Ameisen darstellt und nicht immer eine optimale Lösung für komplexe Probleme bietet. Dennoch hat er in vielen Anwendungen, einschließlich der Lösung von Labyrinthen, beeindruckende Ergebnisse erzielt.



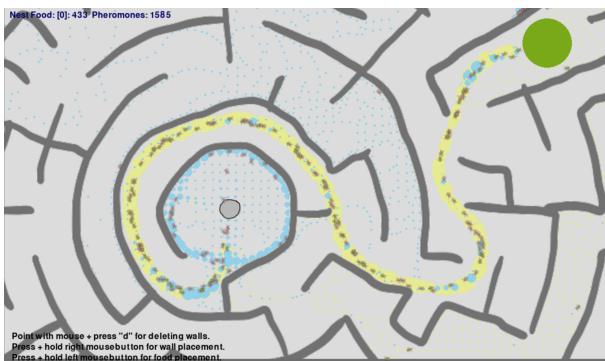
Beginn der Simulation: Die Ameisen schwärmen aus dem Nest aus und begeben sich auf die Suche nach Futter. Dabei hinterlassen sie blaue "zum Nest"-Pheromone. Dickere Punkte deuten auf eine höhere Pheromonkonzentration hin.



Anfangsphase: Erste Ameisen erreichen zufällig die Futterquelle. Wenn sie Futter gefunden haben, hinterlassen sie gelbe "zum Futter" Pheromone.



Weiterer Verlauf: Mehr Ameisen folgen den gelben Pheromonspuren. Sie legen dabei blaue Spuren, damit die Ameisen mit dem Futter den Weg zurückfinden. Die Pheromonspuren werden immer stärker und es folgen immer mehr Ameisen den Spuren, was den Effekt wiederum verstärkt.



Ergebnis: Es hat sich ein Gleichgewicht eingestellt und die Ameisen finden die Futterquelle zuverlässig. Das Futter wird nach und nach in das Nest gebracht.

6.2. Simulation Doppelbrückenexperiment

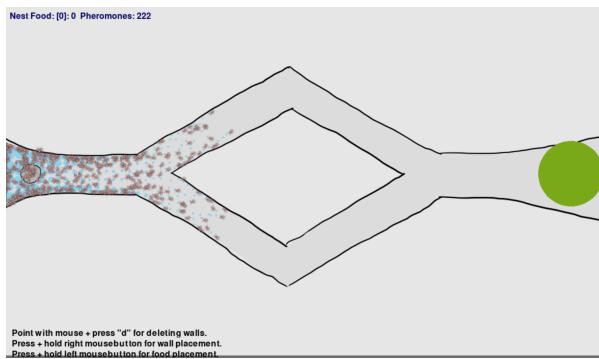
Ein prominentes Experiment, das zur Untersuchung der kollektiven Intelligenz und der emergenten Verhaltensweisen von Ameisenpopulationen verwendet wird, ist das Doppelbrückenexperiment [6]. Im Folgenden werden verschiedene Szenarien dieses Experiments betrachtet und die Lösungen analysiert, die von Ameisenkolonien entwickelt werden.

Das Doppelbrückenexperiment besteht aus zwei separaten Wegen, die von der Ameisenkolonie zu einer Futterquelle führen und miteinander verbunden sind. Die Ameisenkolonie wird vor die Herausforderung gestellt, ihre Ressourcen effizient zwischen den beiden Wegen aufzuteilen, um die optimale Nahrungsversorgung sicherzustellen. Es gibt zwei unterschiedliche Szenarien dieses Experiments, welche im Folgenden betrachtet werden.

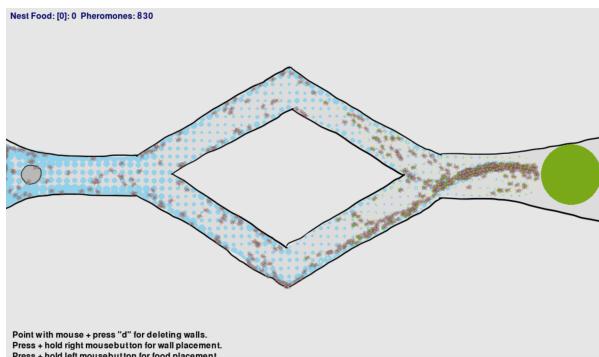
Simulation mit gleicher Weglänge

Wenn beide Wege die gleiche Länge haben, ist die Herausforderung für die Ameisenkolonie, ein Gleichgewicht zu finden, um die Ressourcen gleichmäßig zu verteilen. Studien haben gezeigt, dass die Ameisen in der Lage sind, eine pheromon- basierte Kommunikation einzusetzen, um den Weg mit mehr Ameisen Attraktivität zu markieren. Dies führt dazu, dass die Kolonie nach und nach mehr Ameisen auf diesem Weg rekrutiert, wodurch die Ressourcenverteilung ausgeglichen wird.

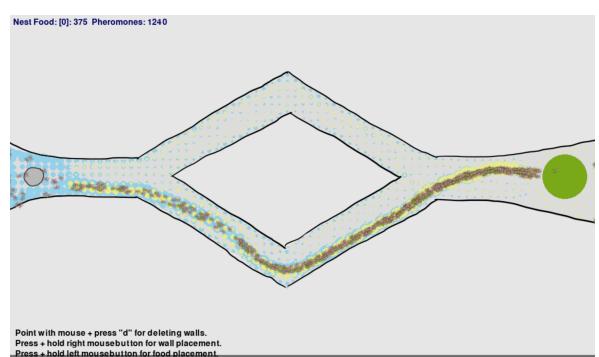
Die Ameisen entscheiden sich für einen der beiden Wege und bevorzugen diesen, bis die Futterquelle aufgebraucht ist. Die Wahrscheinlichkeit, mit der die Ameisen sich für den oberen oder den unteren Weg entscheiden, liegt in der Theorie bei 50 %. [1]



Beginn der Simulation: Die Ameisen entscheiden sich mit einer Wahrscheinlichkeit von 50% in Bezug auf den Weg, den sie wählen.



Anfangsphase: Es nehmen zufälligerweise mehr Ameisen den Weg entlang der unteren Brücke als entlang der oberen Brücke.



Weiterer Verlauf: Mehr Ameisen nehmen den unteren Weg. Die Pheromone verstärken sich dort, wodurch wiederum noch mehr Ameisen diesen Weg nehmen.

Um das Verhalten des simulierten Multiagentensystems zu testen, wurden hundert Testläufe gestartet. Basierend auf den Ergebnissen des Doppelbrückenexperiments lässt sich folgendes deuten:

- Von den 100 Läufen sind 40 Läufe (40 %) dokumentiert, in denen die Ameisen den oberen Weg genommen haben.
- In 42% der Läufe haben die Ameisen den unteren Weg gewählt.
- In 15% der Läufe haben die Ameisen während des Experiments gewechselt, indem sie entweder vom oberen zum unteren Weg oder umgekehrt gewechselt sind. Dies deutet darauf hin, dass die Entscheidung der Ameisen, welchen Weg sie wählen, nicht immer konstant ist.

Der Faktor `PheromoneDecay`, also die Verdunstungsrate der Pheromone, kann dazu führen, dass die Ameisen ihre Präferenz ändern. Um dies zu untersuchen, wurde die Verdunstungsrate der Pheromone von zuvor 0,0005 auf 0,001 angehoben. Das heißt, dass die Pheromone doppelt so schnell verdunsten. Das Ergebnis:

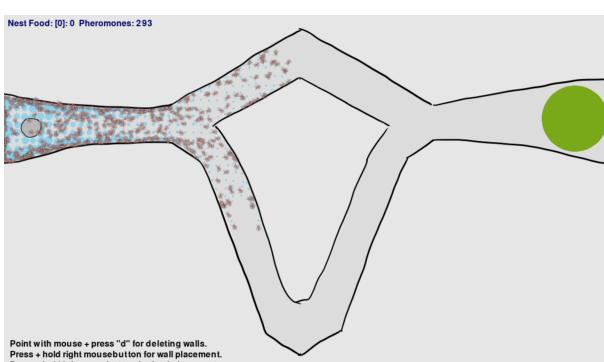
- Von den 100 Läufen haben sich die Ameisen in 49 Läufen (49 %) für den oberen Weg entschieden.
- In ebenfalls 49% der Läufe haben die Ameisen den unteren Weg gewählt.
- In nur zwei Fällen (2%) haben die Ameisen während des Experiments gewechselt, indem sie entweder vom oberen zum unteren Weg oder umgekehrt gewechselt sind.

Die Verdunstungsrate (PheromoneDecay) hat also einen entscheidenden Einfluss auf die Beständigkeit bei der Wegfindung.

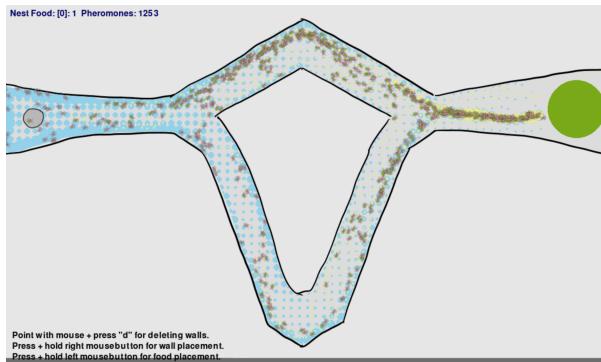
Simulation mit unterschiedlicher Weglänge

Wenn die Wege unterschiedliche Längen haben, müssen die Ameisen eine effiziente Strategie entwickeln, um die kürzere Route zu bevorzugen. Studien haben gezeigt, dass die Pheromonspuren auf dem längeren Weg aufgrund von Verdunstungseffekten schwächer sind als auf dem kürzeren Weg. Dies führt zu einer erhöhten Attraktivität des oberen Weges für die Ameisen. [1]

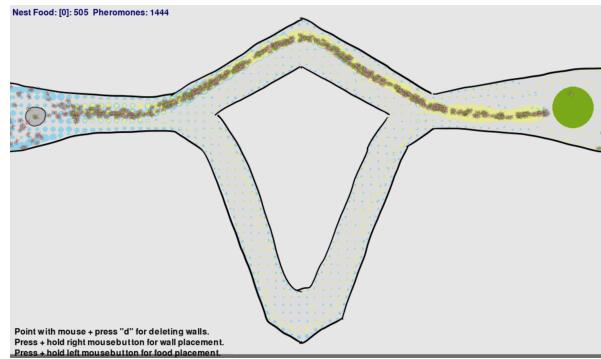
Indem die Ameisen verstärkt den kürzeren Weg entlanglaufen, hinterlassen sie mehr Pheromone auf diesem Pfad. Dieser positive Rückkopplungsmechanismus verstärkt die Attraktivität des kürzeren Weges weiter, während die Pheromonspuren auf dem längeren Weg allmählich verschwinden. Letztendlich führt dies dazu, dass immer mehr Ameisen den kürzeren Weg wählen, da er durch die intensivere Pheromon-Konzentration attraktiver geworden ist. Dies ermöglicht der Ameisenkolonie, die Ressourcen effizienter zu verteilen und eine optimale Nahrungsversorgung sicherzustellen.



Beginn der Simulation: Die Ameisen laufen sowohl oben (kurzer Weg) als auch unten (langer Weg) entlang.



Anfangsphase: Durch Verdunstungseffekte sind die Pheromonspuren auf dem kürzeren Weg stärker. Es laufen mehr Ameisen den kürzeren Weg.



Weiterer Verlauf: Der Effekt verstärkt sich dadurch, dass immer mehr Ameisen den kürzeren Weg nehmen. Es bildet sich eine stabile Route.

Für dieses Szenario wurden insgesamt 25 Testläufe gestartet. Die Ameisen haben sich in diesem Szenario zu 100 % (25 von 25) für den kürzeren oberen Weg entschieden.

7. Fazit

Abschließend lässt sich sagen, dass die von uns umgesetzte Simulation einer Ameisenkolonie veranschaulicht, wie relativ simple einzelne Akteure in einem MAS dazu in der Lage sind, zusammen komplexe Probleme zu lösen.

Bei der Implementierung wurde uns bewusst, wie wichtig bei einer solchen Simulation das Finetuning der verschiedenen Parameter und deren Abgestimmtheit aufeinander sind, um zu einem zufriedenstellenden Ergebnis zu gelangen. Kleine Änderungen können hier beträchtliche Auswirkungen zur Folge haben.

8. Anhang

8.1. Quellenverzeichnis

- [1] nach Wooldridge, Michael (2002). An Introduction to MultiAgent Systems. John Wiley & Sons. p. 366. ISBN 978-0-471-49691-5
- [2] <https://www.mpg.de/research/ants-orientation>
- [3] <https://www.sciencedaily.com/releases/2018/04/180426130001.htm>
- [4] <https://www.livescience.com/871-ants-marching-count-steps.html>
- [5] <https://www.annualreviews.org/doi/10.1146/annurev-ento-010814-020627>
- [6] http://www.scholarpedia.org/article/Ant_colony_optimization#The_double-bridge_experiment

8.2. Config-Parameter

M1

- AntAngleVariation – Maximale Änderung des Winkels
- AntAngleStep – Laufweite nach jeder Winkeländerung
- evtl. zusätzliche Parameter wie z.B. maximale Beschleunigung, für den Fall, dass die Fortbewegungsgeschwindigkeit der Ameisen nicht konstant bleiben soll

M2

- AntFoodSize – (minimale) Größe der Nahrungsquellen
- AntSenseRadius – Maximaldistanz, aus der Ameisen Futterquellen (und Pheromone) erkennen können

M3

- AntPheromoneDrop – Schrittweite zwischen Pheromonen
- PheromoneDecay – Verringerung der Intensität pro Simulationsschritt
- PheromoneMapTileSize – Seitenlänge der Kacheln für die "Pheromonkarte"
- PheromoneDistanceReduce – Abschwächung der hinterlassenen Pheromone pro Simulationsschritt ohne Zustandsänderung

M4

- AntSenseRadius – Maximaldistanz, aus der Ameisen Pheromone (und Futterquellen) erkennen können
- AntFieldOfView – Blickwinkel, in dem Pheromone wahrgenommen werden

Excel Tabelle mit Simulationsergebnissen hier einfügen :)

Config für Map “Maze”:

```
AntMoveDistance = 1
AntAngleVariation = 10
AntAngleStep = 10
AntSleepTime = 0.001
AntPheromoneDrop = 5
AntSenseRadius = 50
AntSenseRadiusPoisonPheromones = 50
AntFieldOfView = 70
AntWallViewDistance = AntMoveDistance + AntAngleStep + 5
AntWallSearchAngle = 15

RandomSteeringUpdateIntervalMin = 4 # lowest possible number of direction
updates where the random steering angle stays the same
RandomSteeringUpdateIntervalMax = 12 # highest possible number of direction
updates where the random steering angle stays the same
RandomSteeringWeight = 1 # applied to reduce the strength of random steering if
there's a trail to follow (was 0.2 in Wieland's version)

# graphical configs
NestImageFile = "assets/nest.png"
AntImageFile = "assets/ant_small.png"
AntViewMaskFile = "assets/ant_small_view_mask.png"
#AntFoodPosition = (10,10)
AntFoodPosition = (12,9)
AntMiddlePosition = (9,9)
AntFoodSize = 2.5

NestSize = 60
FoodSize = 0.03

PheromoneSize = 2
PheromoneDecay = 0.0001
PheromoneDistanceReduce = 0.1
PheromoneMapTileSize = 15
# The maximum food size that can be placed per mouse click (interaction) is
defined as follows
MaxUserFoodSize = 400
UseThreading = False
UseNumpy = False

eraserLimit = 50
```