Fachhochschule für
Wirtschaft Berlin
Berlin School of Economics

Modulnummer:	IT3191
Modul:	KI-Methoden
Modulverantwortlicher:	Dr. Dagmar Monett Díaz

„Rescue Team simulator“:
Entwicklung und Implementierung

Autor:	Paul Steven Robert Schalow

Aktuelle Version:	13.12.2012

I. Abstract

Dieses Paper spiegelt die Entwicklung und Implementierung in NetLogo eines Multi-Agenten-Systems speziell für das Beispiel des „Rescue Survivor“ Szenarios wieder.

Darin enthalten ist die Umsetzung des BDI Schemas auf verschiedene Arten von Rettungsagenten, deren Algorithmen in einer Art Wettkampf zueinander gesetzt werden. Dementsprechend wird auf unterschiedliche Filter bei der Zielfindung und auf einen Pathfinding Algorithmus zur Zielumsetzung eingegangen. Besonders interessant gestalten sich Probleme, die durch Filter entstehen, wie diese durch neue Filter gelöst werden und welche Phänomene durch Filter hervorgerufen werden. Als weiteren besonderen Kernpunkt wird die Kommunikation zwischen Agenten hervorgehoben.

Inhaltsverzeichnis

0 Vorbetrachtungen

Dieser Punkt der Arbeit ist bei allen Mitgliedern identisch, da die nachfolgenden zwei Punkte (0.1 und 0.2) für die Gruppe allgemeingültig sind.

0.1 Aufgabenstellung (Original)

“A group of rescue agents is modelled as a MAS whose goal is to find Survivors of a disaster as quickly as possible. Obstacles are positioned on the grid. A Survivor will deteriorate his/her health with probability lower than 0.6. Otherwise he/she will remain passive. The initial health is a global parameter to the system. A Survivor is found when a single rescue agent moves to the location of the Survivor. Doing so, the Survivor is rescued until the end of the current episode. An episode finishes when all Survivors have been rescued.

Rescue agents collisions are penalized by randomly repositioning the involved agents. Sensoring information about all other rescue agents is possible (global variables, table, etc.). Explicit coordination should occur when all rescue agents have completely surrounded the Survivors (one by one). They also have information about the intentions of other rescue agents on where to move.

Settings: nr. of rescue agents, nr. of Survivors, nr. of grid cells, nr. of episodes.

Analyze: types of rescue agents (moving random, avoiding collisions, with/without coordination.

Study:averaged rescue time, averaged rescued Survivors per rescue agent.”(Díaz)

1 Einleitung

Immer wieder berichten Medien von Szenarien mit verschütteten Opfern durch eingestürzte Gebäude. Dies zeigt auch ein aktuelles Beispiel vom 5. November 2007, bei dem mehr als 100 Häuser durch eine Schlammlawine begraben wurden.[1] Die Rettung der betroffenen Opfer gestaltet sich meist durch die Diskrepanz der instabilen Umgebung und dem dadurch entstehenden Risiko für die Retter äußerst schwierig. Durch nachrutschende Massen oder weiter zu zusammenstürzende drohende Bestandteile sind Helfer oftmals gefährdet. Durch diesen Umstand muss oft abgewogen und ein Risiko des Verlustes weiterer Menschenleben eingegangen werden.

Menschenleben sind nicht ersetzbar, „Maschinenleben“ schon.

Das Einsetzen intelligenter Maschinen und Ersetzen der menschlichen Helfer würde diesen Vorteil verfolgen. Intelligent bedeutet hierbei zum Beispiel eine Minimierung des Aufwandes zum Retten der Opfer sowie eine geschickte Verhaltensweise bei unbekanntem Territorium.

Der „Rescue Team simulator“ ist das Resultat einer Projektarbeit der drei Studenten Philipp Krüger, Steffen Müller und Paul Schalow im Studiengang Technische Informatik an der Fachhochschule für Wirtschaft Berlin Fachbereich Berufsakademie.

In dieser Multiagenten Simulation ist es möglich, verschiedene Such- und Rettungsalgorithmen in zufälligen Szenarien zu testen und miteinander zu vergleichen.

Sämtliche folgende Darstellungen bauen auf Paper der Team-Mitglieder Steffen Müller und Philipp Krüger auf und setzen ein erlesenes Verständnis voraus.

[1] (ORF, 2007)

1.1 Überblick

Im Groben und Ganzen lässt sich die Entwicklung in drei Oberkategorien thematisieren. Da steht an erster Stelle das „Setup“. Dies beinhaltet die Erstellung von Episoden bzw. Experimenten. Dabei müssen z.B. die Welt erstellt, Agenten gesetzt oder Patches konfiguriert werden. Die Parameter werden prinzipiell nicht hart kodiert[2], das bedeutet für den Benutzer frei einstellbar. Die Einstellungen werden über die GUI getätigt. Zur Erstellung der GUI gehören in der zweiten Kategorie auch grafische Methoden für das flüssige Bewegen der Agenten, das Anzeigen des „Penalty-Moves“, Darstellungen des „View-Radius“ oder der internen Weltrepräsentation eines Rescue[3]-Agenten. Weiterhin ist diesem Bereich das Anzeigen der Plots, der Statistiken über eine aktuell laufende Simulation zuzuordnen.

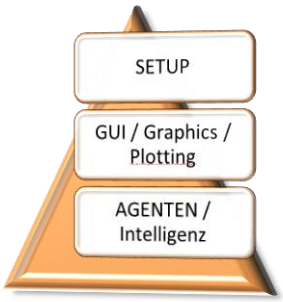


Abbildung 1: thematischer Inhalt der Entwicklung

Die GUI wird von einem anfänglichen vergleichbaren „Bildschirmschoner“, auf dem wahllos irgendwelche Agenten sich bewegen zu einem mittlerweile fast mächtigen Simulationstool für das Rettungsszenario entwickelt.

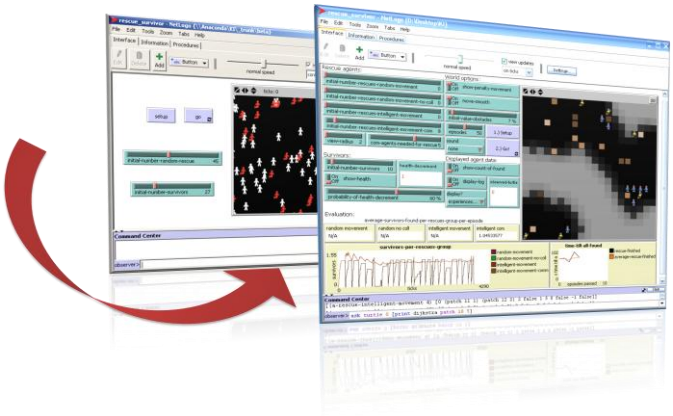


Abbildung 2: Fortschritt der GUI

Der weitere Verlauf dieses Dokuments befasst sich jedoch vielmehr mit der dritten Oberkategorie, mit dem komplexeren und interessanteren Bereich der Agenten und konzeptionierten künstlichen Intelligenz, die in diesem Projekt eingesetzt werden. Diese Thematik wird für NetLogo in der Version 4.0 entwickelt und getestet.[4]

[2] im Quelltext konstant festgelegt
[3] engl. für Rettung
[4] nähere Informationen dem Paper von Philipp Krüger entnehmen

2 Allgemeiner Simulationsablauf

Der gesamte Ablauf einer Simulation bzw. eines Experiments wird von einer globalen Methode `go` periodisch gesteuert. Da in NetLogo parallele Berechnungen äußerst diffizil zu realisieren sind, ist die Abarbeitung der Agenten rundenbasiert gestaltet. Das bedeutet, jeder Agent ist genau einmal pro Periode explizit in der Lage eine Aktion durchzuführen. Eine Periode bedeutet in diesem Fall ein `tick`[5].

Abbildung 3: rundenbasierter Ablauf einer Simulation

Jedem Episodenanfang steht ein Setup bevor, bei dem die Episode vorbereitet wird. Anschließend wird den Agenten die Möglichkeit gegeben eine Aktion durchzuführen, zum Beispiel sich ein Patch weiter zu bewegen. Jeder `breed`[6] ist eine spezifische `action-`*`breed`* Methode zugeordnet. In diesem Schritt wird jeder Agent der Gesamtheit aller Agenten zufällig einmal abgefragt und diese Operation durchgeführt. Nachdem die Aktion festgelegt ist, führt die `move-agents` Methode diese für jeden Agenten aus. Hierbei wird außerdem durch `check-collision` überprüft, ob eine Kollision durch einen Agenten mit einem anderen stattgefunden hat und gegebenenfalls ein Penalty durchgeführt. Zusätzlich erhöht die `check-Survivors-here` Operation die Statistik der gefundenen Survivors[7] eines Rescue-Agenten, wenn dieser einen Patch mit einem Survivor betreten hat. Dieser Survivor wird gleichzeitig aus der Welt genommen. Im nächsten Schritt stellt die `do-plot-Survivors-rescued` Methode die gesammelten Statistiken grafisch dar. Eine Periode wird mit Überprüfung des Episoden-[8] bzw. des Simulationsendes[9] abgeschlossen. Je nach eintretendem Fall muss am Beginn der nächsten Periode das jeweilige Setup durchgeführt bzw. die Simulation beendet werden.

[5] Zeiteinheit, in der sich jeder Agent exakt einmal bewegen darf
[6] Rasse
[7] engl. „Überlebende“
[8] wenn alle Survivors gerettet wurden bzw. verstorben sind
[9] wenn die festgelegte Anzahl der durchzuführenden Episoden erreicht wurde

3 Auswirkungen auf und durch das BDI Konzept

Im folgendem wird dargestellt, wo und wie sich Bestandsteile des festgelegten BDI[10,11] Konzepts zur Aktionsfindung der Agenten im Simulationsablauf wiederspiegeln. Im späteren Verlauf wird noch einmal genauer erläutert wodurch diese Spezifikation zustande kam.

3.1 Übersicht Reflektion von BDI auf Umsetzung

Im nächsten Unterkapitel wird die interne Weltrepräsentation eines Agenten näher erläutert. Die Initialisierung dieser erfolgt in den Setup Methoden. Die agentenartspezifischen `action-breed` Operationen sind wiederum für das Updaten dieser verantwortlich und nutzen dadurch gesammelte Erfahrungen über die Umwelt, um die Deliberation, das Festlegen, welches Ziel erreicht werden soll, durchzuführen. Hierzu ist vor allem das Festlegen der desires sowie der beliefs, daraus resultierende Optionen und intentions zu zählen. Alle umgesetzten Agenten glauben (belief), dass unerreichte Patches frei sind, das heißt nicht mit einem Hindernis oder ähnlichem belegt sind, maximal einen Survivor „beherbergen“. Demzufolge sind unerreichte Patches, genau wie Patches auf denen sich Survivors befinden, erstrebenswerte Zustände (desires), da beide die Chance auf die Rettung von Survivor bieten. Aus sämtlichen Optionen wird mittels verschiedener Filter, die im Folgenden näher erläutert werden, die momentan wahrscheinlich optimale intention bestimmt. Weiterhin wird durch diese Operation das Means-Ends-Reasoning, das Erreichen des Plans, durchgeführt. Hierbei wird der nächste Schritt zum Erreichen der aktuellen intention, berechnet. Dieser erste Schritt wird, nachdem bei allen Rescue-Agenten die beschriebene Prozedur durchgeführt wurde, durch die diversen `move` Methoden ausgeführt und ist damit als Aktion der Agenten zu betrachten. Die Schleife des BDI Pseudocodes wird erneut begonnen, bis sämtliche beliefs und desires überprüft bzw. durchgeführt wurden.

[10] Abk. BDI ≙ beliefs, desires, intentions = engl. für Annahmen über die Umwelt, erstrebenswerte Zustände, mögliche Zielstellungen
[11] nähere Informationen dem Paper von Steffen Müller entnehmen

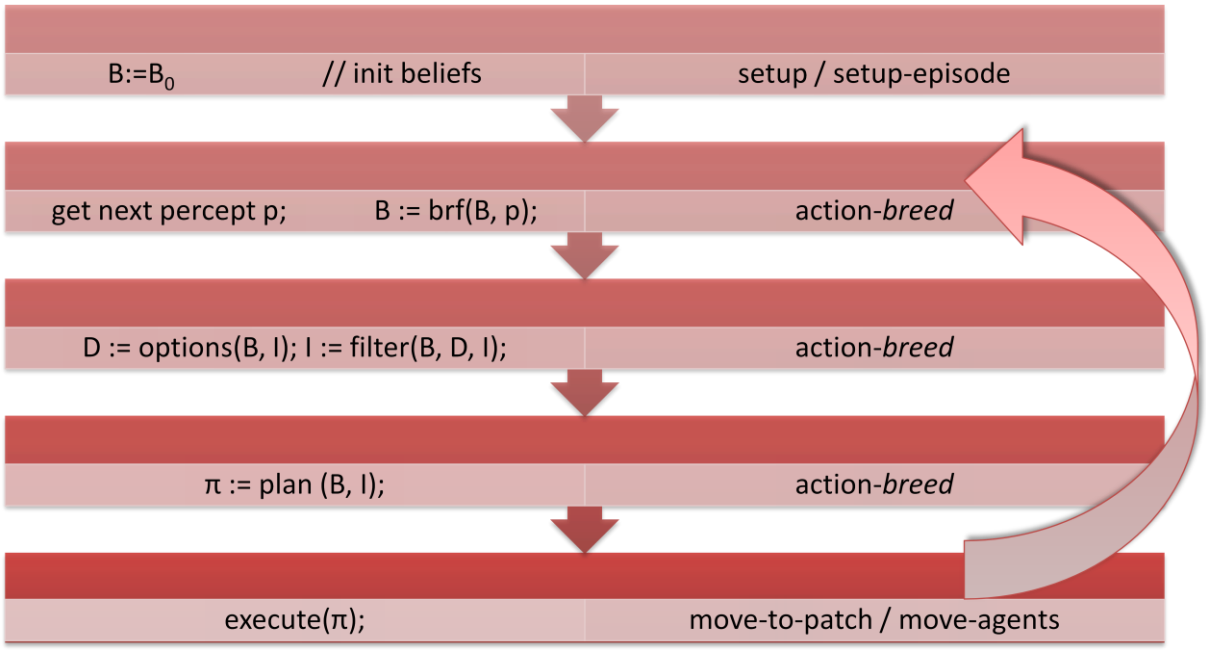


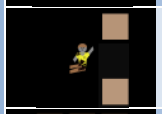
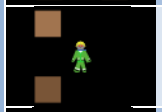
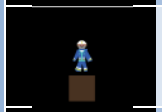
Abbildung 4: Übersicht – Reflektion von BDI auf Umsetzung

4 interne Weltrepräsentation

Jeder Rescue-Agent kapselt für sich eine interne Repräsentation der Welt. Diese Repräsentation ist gleichzusetzen mit einem Gedächtnis, da es fortlaufend, abhängig von der unmittelbaren Umwelt eines Agenten, aktualisiert wird. Da es in diesem Sinne die gesamte Welt repräsentieren kann, muss es dementsprechend auch eine Größe annehmen, die so mächtig ist, sämtliche mögliche Zustände der Welt zu speichern. Die Welt wird durch ein zweidimensionales Raster dargestellt. Es wird eine `turtle` Variable `experience-array` vom Typ *`Array of numbers`* realisiert. Arrays sind in NetLogo nur mittels einer extension möglich, da normalerweise in Logo hauptsächlich mit Listen gearbeitet wird. Für den Fall des wahlfreien Zugriffs eignet sich in Hinsicht auf die Zugriffsgeschwindigkeit eher der Typ Array. Weiterhin besitzt NetLogo das Manko, nur eindimensionale Array abbilden zu können. In diesem Fall werden einige Zugriffsfunktionen benötigt die wiederum zu Lasten der Performanz gehen. Ein Versuch des Erstellens eines zweidimensionalen Arrays, indem jedem Element des eindimensionalen Array ein weiteres eindimensionales Array zugeordnet wird, bringt trotz hoher Erwartung an gewonnener Performanz eher einen Geschwindigkeitsnachteil.

In diesem experience-array finden sich beliefs und desires eines jeden Agenten wieder. Jedes Element reflektiert einen Patch der entdeckten Welt und kann die folgenden aufgeschlüsselten Werte annehmen. Diesbezüglich werden spezifische Konstanten eingeführt.

Tabelle 1: mögliche Werte im experience-array

	Darstellung	Konstante	Wert
Welt		UNDISCOVERED	0
		FREE	1
		OBSTACLE	2
Survivor		SURVIVOR	3
		SURVIVOR_NOT_RESCUEABLE	4
rescue agents		RESCUE-RANDOM-MOVEMENT	5
		RESCUE-RANDOM-MOVEMENT-NO-COLL	6
		RESCUE-INTELLIGENT-MOVEMENT	7
		RESCUE-INTELLIGENT-MOVEMENT-COMM	8
		RESCUE-INTELLIGENT-MOVEMENT-COMM-NO-MOVE	9

Zu Beginn einer Episode werden die internen Weltrepräsentationen sämtlicher Agenten mit `UNDISCOVERED` initialisiert. Danach nehmen die Elemente des Array entweder Werte über Aussagen von freien oder mit Hindernissen belegten Patches an. Es ist auch möglich, dass ein Survivor entdeckt wird, sowie dass dieser eventuell so eingeschlossen ist, dass er überhaupt nicht gerettet werden kann und demzufolge auch nicht weiter verfolgt werden muss. Diese Entscheidung übernimmt die Methode

`interpretate-experience`, welche im Folgenden noch beschrieben wird. Neben Survivor-Agenten können aber auch weitere Rescue-Agenten erkannt werden. Hierbei besteht die Besonderheit in kommunikativen Agenten mit intelligenter Bewegung die in einem Momentzustand direkt an einen Survivor angrenzen. Der beobachtende Rescue-Agent kann in diesem Fall mit einer gewissen Wahrscheinlichkeit damit rechnen, dass dieser weitere Rescue-Agent sich erstmal nicht weiter bewegen wird, da er eventuell auf weitere „zur Hilfe eilende“ kommunikative Agenten warten muss. Demzufolge ist dieser Agent für einen gewissen Zeitraum als Hindernis anzusehen.

4.1 Beeinflussung

Wie genau wird eine interne Weltrepräsentation eines Agenten beeinflusst?

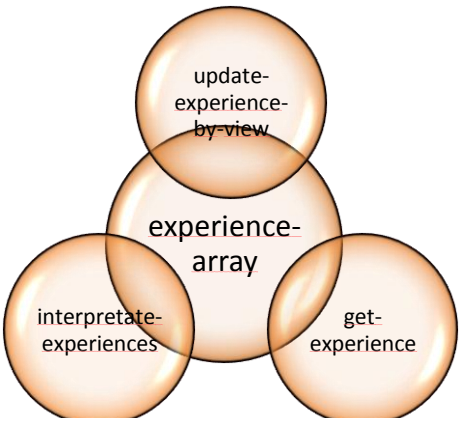


Abbildung 5: Beeinflussung des experience-array

Das experience-array bzw. die interne Weltrepräsentation reflektiert die Annahmen eines Agenten über seine Umwelt. Demzufolge muss er in der Lage sein, diese Umwelt beobachten zu können. Dafür verantwortlich ist die Methode `update-experience-by-view`, die in jeder Runde für jeden Agenten explizit einmal ausgeführt wird, bevor dieser seine Aktionen planen und durchführen kann. Dabei wird in einem von dem Benutzer spezifizierten radialen Abstand zum jeweiligen Agenten sämtliche Patches beobachtet und damit die interne Weltrepräsentation angepasst. Weiterhin tauschen kommunikative Agenten über die `get-experience` Methode neu gewonnene Erfahrungen aus. Wie im letzteren Kapitel beschrieben, können einige Zustände auch interpretiert werden zu besonderen Umständen, die es bei der Planung zu beachten gilt. Dafür verantwortlich ist die `interpretate-experiences` Operation, die als letzter Schritt vor der Planung ausgeführt wird.

5 weitere Instanzvariablen

Aus der Aufgabenstellung und Spezifikation gehen weitere wichtige Variablen hervor. Beispielsweise kapseln Patches in `has-obstacle`, ob ein Hindernis an dieser Stelle besteht. Survivors müssen ihren Gesundheitszustand mitteilen können. Dieser wird in einer Instanzvariablen `health` gekapselt. Um noch neben vielen weiteren zwei wichtige zu nennen, ist an dieser Stelle `count-survivors-rescued` zu erwähnen. Diese Variable wird den Rescue-Agenten zugeordnet und dient der Statistikermittlung, indem sie für jeden Rescue-Agenten die Anzahl der bereits geretteten Survivors speichert. Die wohl bedeutendste ist jedoch `next-patch-to`. `next-patch-to` reflektiert den aktuellen ersten Schritt (*sub goal*) in einem Plan zu einer intention (*goal*). Jede intention kann als Patch angesehen werden, da ein Ziel eines jeden Rescue-Agenten darin besteht, entweder auf ein Feld eines Survivors zu treten und ihn somit zu retten oder in die Nähe eines unentdeckten Patches zu gelangen um somit neue Erfahrungen zu erlangen.

6 Erreichen von Zielen

Im Folgenden wird beschrieben, wie diese spezielle Instanzvariable `next-patch-to` beeinflusst wird, wie also Schritt für Schritt ein Ziel erreicht werden kann.

6.1 Die besondere action-*breed* Methode

Wie bereits erwähnt existieren für jede `breed` spezifische `action-breed` Methoden, die unterschiedliche Ziele verfolgen oder unterschiedliche Wege zu gleichen Zielen darbieten. In den folgenden Unterkapiteln werden diese spezifischen Methoden näher erläutert.

6.1.1 action-survivor

Survivor verfolgen das einzige Ziel des Überlebens. Allerdings können sie das erfolgreiche Abschließen nicht selbst beeinflussen. Dementsprechend gestaltet sich ihre `action` Methode trivial. Hierbei wird mit einer vom Benutzer spezifizierten Wahrscheinlichkeit ihr `health` Zustand dekrementiert. Sobald dieser den Wert 0

(null) erreicht, stirbt, im virtuellen Sinne, der Survivor und wird aus der Welt genommen.

6.1.2 action-rescues-random-movement (-no-coll)

Auch die `action` Methoden der Random-Agenten gestalten sich noch sehr trivial, da sie nur eine begrenzte Anzahl von Optionen haben und die Planung sehr einfach ist. So legt in der `action-rescues-random-movement` Methode die Funktion `get-random-neighbor`, bei den Random-Agenten mit Kollision, benachbarte Patches ohne Hindernis als Alternativen fest. Der Filter gestaltet sich als einfache zufällige Rückgabe eines dieser Alternativen. Der Unterschied der `get-random-neighbor-no-coll`, der bei der `action-rescues-random-movement-no-coll` Methode Verwendung findet, definiert sich dadurch, dass bei den Alternativen, auch auf ein „frei sein“ im Sinne von nicht vorhandenen Rescue-Agenten geachtet wird. Dadurch werden Kollisionen vermieden.

6.1.3 action-rescue-intelligent-movement (-comm)

Die Agenten, bei denen intelligente Algorithmen ins Spiel kommen sollen, verwenden in ihren `action` Methoden die spezielle Operation `determine-patch-to`, die sich hierbei viel stärker am BDI Konzept orientieren muss, als das bei den action Methoden der Random-Agenten der Fall ist.

Hierbei definiert sich eine intention nicht nur als Patch, sondern als ein Patch mit diversen zusätzlichen Flags. Diese geben eine weiterschließende Auskunft über das Patch der intention. Diese Flags werden von Filtern bestimmt, die noch im Folgenden genauer beschrieben werden.

Jedes Patch der Welt ist eine mögliche intention. Bei dieser Betrachtung wird die interne Weltrepräsentation hinzugezogen und durch Filtern die optimale intention für den aktuellen Zustand bestimmt. In jeder Runde überprüft die `action` Methode ob die aktuelle intention noch immer die optimale darstellt. Durch diese ständige Aktualisierung sind zwar Performanzeinbußen hinzuzurechnen, jedoch ist ein stupides „Vorbreirennen“ an einem optimaleren Ziel ausgeschlossen.

6.2 Filter

Im Folgenden wird beschrieben, wie durch unterschiedliche Parameter versucht wird, die optimale intention mittels diverser Filter zu bestimmen

6.2.1 Vorbetrachtungen

Es wird sich bemüht, bestimmte Grundeigenschaften autonomer Agenten beizubehalten, um dem Wert eines „Multi-Agenten-Systems“ standhaft zu bleiben.

Dies ist vor allem bei den kommunikativen Agenten zu beachten. So werden diese nicht durch, z.B. einen globalen Agenten koordiniert, sondern entscheiden autonom, an welcher Stelle sie gemeinsam handeln müssen. Weiterhin verfolgen sie die „soziale“ Eigenschaft, indem sie mittels der `get-experience` Methode andere kommunikative Agenten abfragen und so auch deren Entfernung zu bestimmten Survivors autonom kalkulieren können. Das proaktive Anfordern von Hilfe, zum Retten von entdeckten Survivors, erfolgt indirekt durch Eintragen des Survivors in die interne Weltrepräsentation, welche in einem späteren Zeitpunkt durch den Rest des kommunikativen Teams abgefragt wird. Somit wissen weitere Agenten von diesem Umstand bescheid. Weiterhin beschließt nun jeder Agent autonom, ob und welchem Survivor bzw. kommunikativen Agenten er zur Hilfe „eilt“.

6.2.2 Filter: Survivor / unentdeckt

Hierbei werden intentions aussortiert, die kein desire, also weder Survivor noch unentdeckt repräsentieren.

6.2.3 Filter: Distanz

Dieser Filter bevorzugt bei zwei möglichen intentions die intention, welche sich am nahesten zum Rescue-Agenten befindet, demzufolge schneller erreichbar ist.

6.2.4 Filter: optimale Distanz zum Rand

Dieser Filter bevorzugt die intention, deren Abstand zum nächsten Rand der Welt so optimal ist, dass kaum Abstand zwischen Rand und dem äußeren Ende des View-Radius des Agenten besteht.

Eine derartige Vorgehensweise resultiert aus dem Wunsch, weitestgehend keinen Patch mehrmals zu überlaufen. Dies wäre ohne den Filter im dargestellten Szenario der Fall :

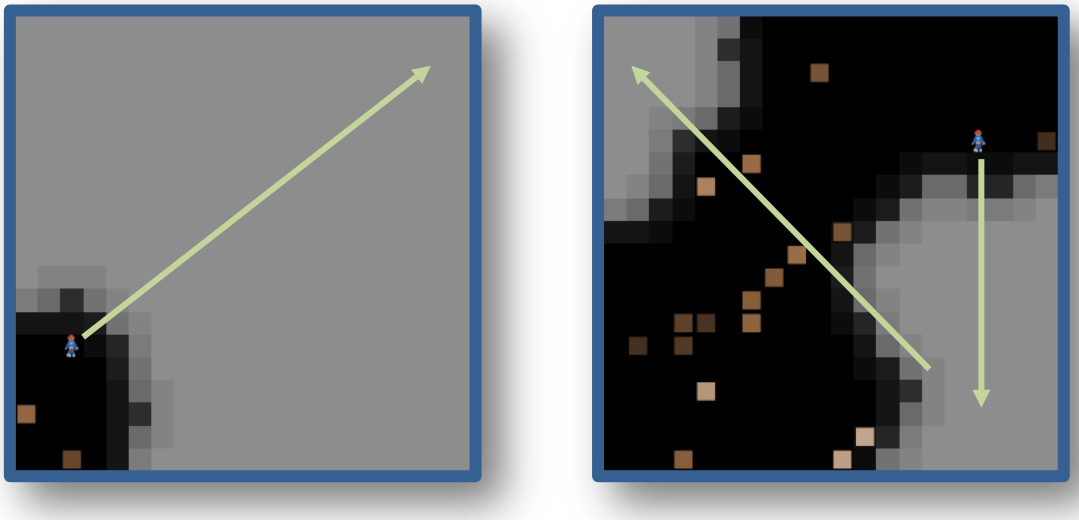


Abbildung 6: Notwendigkeit des Border Filters

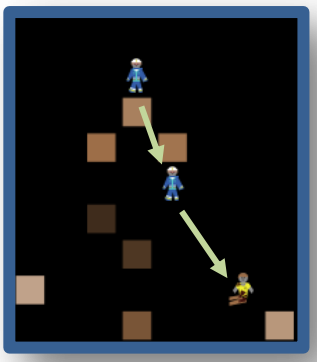
Würde der Agent sich nicht am Rand orientieren und sich quer durch die Mitte über das Spielfeld bewegen, müsste er sich im nächsten Schritt für einen unentdeckten Bereich entscheiden und in dem darauffolgenden Schritt den gesamten, bereits entdeckten Mittelbereich überqueren. Dies wäre ein unnötiger Aufwand und lässt sich meist durch diesen Filter vermeiden, es sei denn ein anderer Filter steht stärker im Vordergrund, zum Beispiel, wenn ein Survivor entdeckt wurde.

6.2.5 Filter: neue Erfahrungen im ersten Schritt

Dieser Filter gibt jener intention den Zuschuss, welche im ersten Schritt zum Erreichen dieser intention schon die meiste neue Erfahrung aufbietet. Damit wird auf dem Weg zu diesem Ziel bereits so gut wie möglich die Welt weiter erkundet.

6.2.6 Filter: Distanz zu Intelligenten

Hierbei wird versucht, die Distanz zu intelligenten Agenten weitestgehend so hoch wie möglich zu halten. Damit soll wie im folgenden Szenario dargestellter Missstand vermieden werden.



Hierbei ist zu erkennen, dass der obere Agent dem unteren nachläuft. Dies würde bedeuten, dass der untere vorher sämtliche Survivor rettet, bevor der obere sie überhaupt erreicht. Somit wäre der obere Agent nutzlos.

Abbildung 7: Notwendigkeit des "Distanz zu Intelligenten" Filters

6.2.7 Filter: „gehört zu den Dichtesten“

Dieser Filter kommt bei kommunikativen Agenten zum Einsatz sobald ein Survivor entdeckt wurde. Die Einstellungen des Benutzers können es möglich machen, dass nicht alle kommunikativen Agenten zum Retten eines Survivors benötigt werden. Demzufolge werden nur die zu dem Survivor dichtesten Agenten zur Rettung herangezogen. Alle weiteren orientieren sich an anderen intentions.

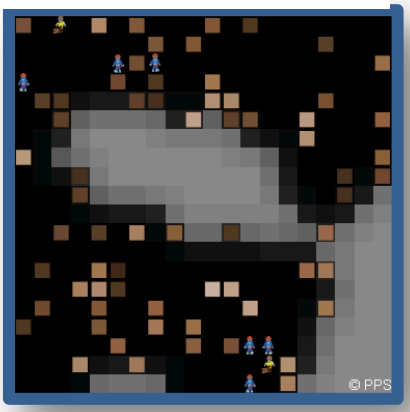


Abbildung 9: Auswirkungen des "gehört zu den Dichtesten" Filters

6.2.8 Filter: „nicht letzter“

Auch dieser Filter kommt nur bei kommunikativen Agenten zum Einsatz. Angenommen ein Agent entdeckt einen Survivor, stellt sich neben ihn und wartet auf eintreffende Hilfe. Vorausgesetzt die weiteren kommunikativen Agenten sind relativ weit weg, muss der wartende Agent relativ viel Zeit in das Warten investieren. In dieser Zeit kann er aber eigentlich noch andere, in der Nähe liegende intentions verfolgen. Dies kann er solange machen, bis er merkt, dass er, von der Entfernung zum Survivor her, der weiteste ist. Dann sollte er sich auch auf den Weg machen zum Survivor.

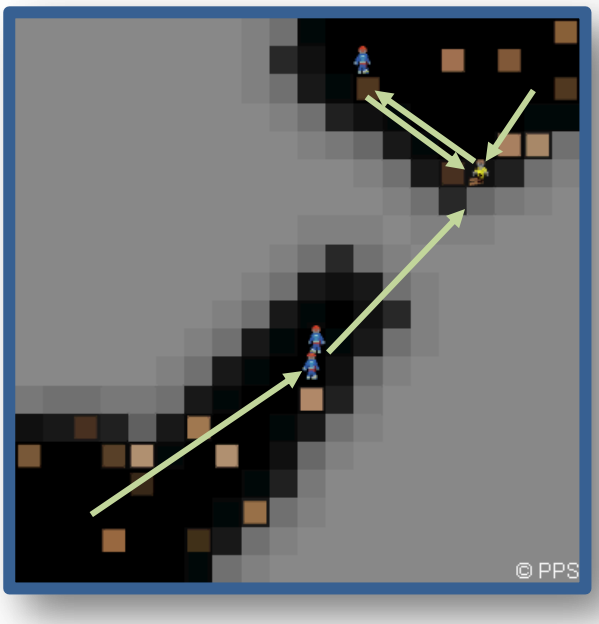


Abbildung 8: Auswirkungen des "nicht letzter" Filters

6.2.9 Probleme / Lösungen durch Filter

Besonders durch den letzten Filter entstehen Probleme bei der Bestimmung der intentions in einem Team. Teilweise kommt es vor, dass sich kommunikative Agenten ständig „im Weg rumstehen“, da immer einer versucht, ein noch anderes Ziel zu verfolgen, weil er weiß, dass er nicht der am weitesten Entfernte ist. Dieser Umstand resultiert in einem ständigen Tauschen der Positionen der kommunikativen Agenten, aber nicht in einem Erreichen des Survivors all jener. Die Lösung dieses Problems besteht darin zu überprüfen, ob sämtliche benötigte Agenten einen minimalen Abstand zueinander besitzen oder sich allesamt in einem spezifizierten Radius um den Survivor befinden. Ist dies der Fall, sollte kein weiteres Ziel verfolgen.

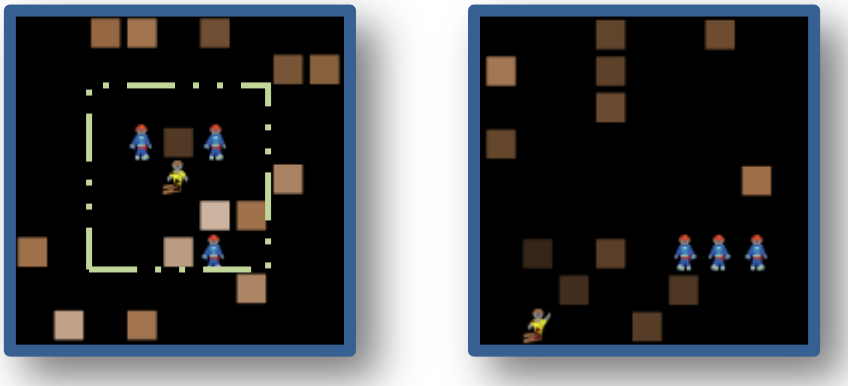


Abbildung 10: Lösungen zu Filter Problemen

Ein weiteres Problem, welches sich herauskristallisiert bei kommunikativen Agenten besteht bei engen Zugängen zu Survivors. Im folgenden Szenario ist ein kommunikativer Agent dargestellt, der den Zugang zu dem Survivor für die anderen benötigten Agenten versperrt, weil er sich in einem wartenden Zustand befindet.

Um diesem Umstand zu vermeiden, wird in einem Fall des wartenden Zustands versucht, sich am Survivor in eine optimalere Ecke zu bewegen.

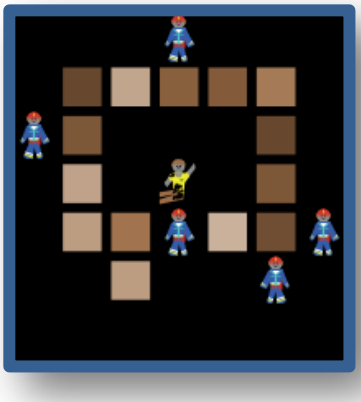


Abbildung 11: weitere durch Filter entstandene Probleme

6.2.10 Auswirkungen / auftretende Phänomene durch Filter

Das folgende Szenario zeigt mögliche Auswirkungen der Filter auf das Bewegungsmuster eines intelligenten Agenten. In erster Linie ist eine optimale Ausnutzung des gesamten Feldes zu erkennen. So wird kaum eine Stelle mehrfach beschritten. Ein weiteres Phänomen, welches so nicht unbedingt prognostizierbar war, besteht in der fast dauerhaften „Zick-Zack“-förmigen Bewegung des Agenten.

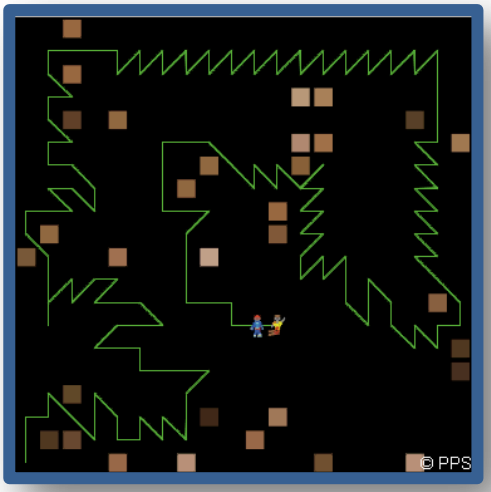


Abbildung 12: Auswirkungen durch Filter

6.3 Means-Ends-Reasoning

Sobald eine optimale intention bestimmt wurde, demzufolge durch die Deliberation ermittelt wurde „was“ zu erreichen gilt, muss festgestellt werden „wie“ dieses Ziel, diese intention erreicht wird. Demzufolge wird ein Plan-Algorithmus eingeführt.

6.3.1 Vorbetrachtungen

Während der Entwicklung stellt sich die Frage zwischen Performanz und Genauigkeit. Ein reaktives System, welches sofort eine passende Antwort auf eine neue Konstellation besitzt, bietet zum Beispiel gegenüber einem deliberativen System deutliche Geschwindigkeitsvorzüge. Das Problem, welches sich hier herausstellt, besteht jedoch in der Variabilität der Parameter, die allesamt durch den Benutzer verändert werden können. Ein Abspeichern sämtlicher Konstellationen in beispielsweise einer Wissensdatenbank wäre utopisch.

6.3.2 Der Dijkstra Algorithmus

Die aktuelle intention stellt einen Patch dar, den es am schnellsten zu erreichen geht. Dieses Problem wird von sogenannten „routing“ bzw. „pathfinding“ Algorithmen aufgegriffen, auch bekannt als „Kürzeste-Wege“-Algorithmen. Der Dijsktra-Algorithmus ist ein bekannter Vertreter dieser Gruppe. Er bietet neben den Vorteilen, dass er garantiert der kürzesten Weg findet bzw. mit hundertprozentiger Wahrscheinlichkeit[12] aussagen kann, ob überhaupt ein Weg zum Ziel besteht, auch Nachteile, vor allem in der Performanz. Der Dijsktra geht bei einem Suchdurchlauf sämtliche Punkte des Wegesystems rekursiv durch und versucht somit von jedem Punkt aus verschiedene Wege zu verbinden, die in ihrer Gesamtheit die kürzeste Verbindung zwischen Start und Ziel darstellen. Bei größeren Wegesystemen kann dies schnell zu langen Rechenzeiten führen. Um diese starken Verzögerungen zu vermeiden. Wird der Algorithmus dahingehen optimiert, dass er zielgerichtet auf den Endpunkt versucht, eine Verbindung zu finden. Weiterhin wird global die aktuell beste Verbindung gespeichert, so dass in den verschiedenen Rekursionsstufen ständig eine aktuelle Abbruchbedingung existiert. Nach Abschluss des Algorithmus, wird der erste Schritt, der zum Erreichen der intention führt, in der Instanzvariablen `next-patch-to` gespeichert.

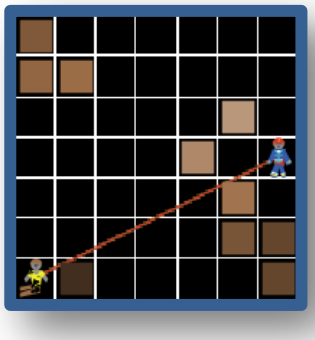


Abbildung 14: Dijkstra-Ausgangssituation

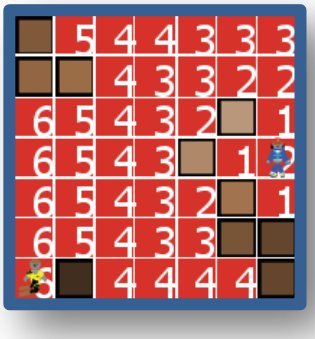


Abbildung 15: Dijkstra-normaler Vorgang

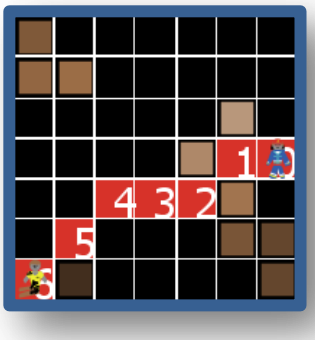


Abbildung 13: Dijkstra-optimierter Vorgang

7 Schlussbetrachtungen

7.1 Der Umgang mit der Sprache „NetLogo“

Für das optimale programmieren in NetLogo ist es wichtig wenigstens einen Überblick der ~ 500 Befehle zu haben. Dieser Befehlssatz weicht teilweise stark von

[12] bezüglich der internen Weltrepräsentation

anderen Logo-Sprachen ab. Neben dem Standard Befehlswortschatz existieren diverse Abkürzungen[13] zu den Befehlen sowie Erweiterungen durch extensions[14]. Prinzipiell ist Logo wohl als Einstiegssprache gedacht, da die Syntax fast selbsterklärend ist, wie z.B. `ask [turtles-here] of one-of [neighbors]` bei dem alle turtles eines Nachbarn abgefragt werden. Aber vor allem beispielsweise der Zugriff auf Listen oder Arrays gestaltet sich durch eine lange Schreibweise äußerst umständlich. Die Agentenkommunikation als Programmierparadigma nimmt dem MAS-Programmierer das Entwickeln der Zwischenagentenkommunikation teilweise ab. Jedoch ist dies auch nicht ohne zu realisieren und wirkt damit aufgezwungen. Es scheint teilweise, dass die Vorteile von Logo generell nur bei der Entwicklung von MAS-Systemen vorhanden sind. Weiterhin sind manche Befehle teilweise recht umständlich oder allgemeine Programmiertechniken überhaupt nicht vorhanden. So existiert bei für eine `ifelse` Abfrage kein `else if` Zweig. Zusätzlich besteht keine Möglichkeit der objektorientierten Programmierung, keine Möglichkeiten des Debuggings, sowie lässt sich eine Endlosschleife nur durch einen Abbruch des Prozesses beenden. Diese Nachteile lassen sich aber bezüglich der Entwicklung von MAS deutlich mit Vorteilen aufwiegen, die das Erlernen dieser Sprache und dieses Frameworks rechtfertigen.

7.2 Ausblick

Weitergehende Entwicklungen sollten sich vor allem die Frage nach dem Realismus stellen. So wurden in dieser Arbeit einige teilweise unrealistische Gedankenzüge vertreten. Allein durch die Spezifikation der Aufgabenstellung und der Einführung des *Penalty-moves*, welches im realen Leben ein „Beamen“ bedeuten würde, vernachlässigt die Umsetzbarkeit.

Intelligente Agenten können theoretisch einen eingeschlossenen Survivor nicht retten. Parallel wurde der Gedanke an eine absichtliche Kollision mit eventueller Penalty auf einen eingeschlossenen Survivor geschürt. Dieser Gedanke wurde jedoch in Hinblick auf den Realismus wieder verworfen.

[13] z.B. create-turtles ≙ crt
[14] z.B. für Array / Sound

Weiterhin sollte sich die Frage gestellt werden, ob ein Nicht-Durchblicken einer Mauer bzw. ein Blickwinkel statt einer Rundumsicht nicht realistischer wäre. Jedoch könnte beides durch den Einsatz von Sonar, Infrarotsensoren, Rundumkamera oder Ähnlichem theoretisch realisiert werden.

7.3 Fazit

Es wurde gezeigt, dass NetLogo bezüglich der Entwicklung eines MAS Systems geeignet ist. Auch bezogen auf den „Rescue Team simulator“. Es konnten erfolgreich Strategien und Algorithmen konzeptioniert und in NetLogo umgesetzt werden.

8 Verzeichnisse

8.1 Abkürzungsverzeichnis

BDI Belief Desires Intentions

MAS Multi-Agenten-Simulation

8.2 Glossar

Belief	engl. „Glauben“; im Sinne BDI: Annahmen über die Umwelt
Desires	engl. „Wünsche“; im Sinne BDI: Erstrebenswerte Zustände
Dijkstra	Beispiel für einen → Pathfinding Algorithmus
Episode	Ein Simulationsdurchlauf, bis Survivors gerettet/gestorben sind
Experiment	mehrere Episoden
Intentions	engl. „Vorhaben“
Kürzeste Wege	dient der Bestimmung des kürzesten Weges in einem Wegesystem
Pathfinding	→ „Kürzeste Wege“
Patches	stationäre Agenten
Rescue	engl. „Rettung“
Routing-Alg.	→ „Kürzeste Wege“
Survivor	engl. „Überlebender“
Tick	Zeiteinheit, in der sich jeder Agent exakt einmal bewegen darf
Turtle	mobile Agenten
Welt	auch engl. World, virtuelles Simulationsfeld in 2D oder 3D

8.3 Tabellenverzeichnis

8.4 Abbildungsverzeichnis

8.5 Literatur- und Internetverzeichnis

Díaz, D. D. (kein Datum). *Berufsakademie Berlin, Fachrichtung Informatik*. Abgerufen am 25. 10 2007 von Vorlesung: KI-Methoden : http://monettdiaz.com/ba/KIM0708.html#kim-ueber

Dr. Monett Díaz, D. (kein Datum). KI-Methoden (3). *Vorlesungsskript Berufsakademie (01.10.2007)* . Berlin.

Dr. Monett Díaz, D. (kein Datum). KI-Methoden (4). *Vorlesungsskript Berufsakademie (08.10.2007)* . Berlin.

Dr. Monett Díaz, D. (kein Datum). KI-Methoden (5). *Vorlesungsskript Berufsakademie (10.10.2007)* . Berlin.

ORF. (5. November 2007). *news.ORF.at*. Abgerufen am 18. November 2007 von http://news.orf.at/?href=http%3A%2F%2Fnews.orf.at%2Fticker%2F270315.html

Preuß, T. (12. Mai 2006). *handout_mas_120506_dhallau_tpreuss*. Abgerufen am 17. November 2007 von http://www.techfak.uni-bielefeld.de/ags/wbski/lehre/digiSA/SS06/Seminar_Multiagentensysteme/folien/handout_mas_120506_dhallau_tpreuss.pdf

Schönmann, F. (19. Oktober 2006). *bdi-slides*. Abgerufen am 18. November 2007 von http://defined.de/studium/bdi/bdi-slides.pdf

Wilensky, U. (2007). *NetLogo 4_0 User Manual*. Abgerufen am 17. November 2007 von http://ccl.northwestern.edu/netlogo/docs/