Thema:

Landminenerkennung durch Robotereinsatz

Ausarbeitung

im Rahmen des Seminars „Agenten in simulierten Umgebungen“

im Fachgebiet Informatik
am Lehrstuhl für Informatik

Themensteller: Dr. Dietmar Lammers
Betreuer: Dr. Dietmar Lammers

Vorgelegt von: Stefan Fleischer
Stephan Kramer
Alexander Simons

Abgabedatum: 09.01.2005

Inhaltsverzeichnis

1. Einleitung

Motivation: Landminensuche durch Robotereinsatz

Verlegt auf Feldern, Straßen und Fußwegen sind Landminen in jedem dritten Entwicklungsland dieser Welt zu einer "Massenvernichtungswaffe in Zeitlupe" geworden. Nach Angabe der Vereinten Nationen sind derzeit ca. 110 Millionen Minen in über 70 Ländern dieser Welt verlegt. Weltweit existieren über 600 bekannte Minentypen, denen monatlich mehr als 2000 Menschen, zumeist Zivilisten, zum Opfer fallen. Da viele Staaten weiterhin Minen produzieren und exportieren, ist ein Ende dieser humanitären Katastrophe nicht abzusehen.

UNO und EU wollen der Landminengefahr entgegentreten und haben sich das ehrgeizige Ziel gesetzt, sich ihrer bis zum Jahre 2010 zu entledigen. Eine manuelle Minensuche erweist sich allerdings nicht nur als mühselig und langsam, sondern auch als äußerst gefährlich: Auf 1.000 geräumte Minen ereignet sich ein schwerer Unfall. Zudem weisen viele der bekannten Minentypen einen sinkenden Metallgehalt auf, so dass sie mit einem Metalldetektor nur schwer oder gar nicht aufzuspüren sind. Vor diesem Hintergrund erscheint nur der gebündelte Einsatz mehrerer Detektoren sinnvoll.

Für die Entwicklung moderner Minendetektions- und Räumungstechnologien werden derzeit jährlich zweistellige Millionensummen aufgebracht. Die Gesamtkosten für die Räumung der weltweit verlegten Landminen werden auf 33 Mrd. Euro geschätzt[1].

Etliche Einflussfaktoren stellen die Entwicklung von Detektoren vor komplexe Probleme: Neben den verschiedenen Minentypen sind u. a. Landschafts-, Sensor- und Geländetypen ins Kalkül zu ziehen. Solche „Drehschrauben“ und nicht zuletzt die Gefahr der manuellen Minensuche legen die Simulation einer Landminenerkennung durch Roboter nahe. Notwendig wird eine Simulationsumgebung auch zur Reduktion der Kosten: Die Entwicklung und insbesondere der fehlerhafte Einsatz eines nicht optimal getesteten Roboters sind sehr teuer.

Zielsetzung der Arbeit

Die Seminararbeit baut auf einer Simulationsumgebung zur Landminensuche durch Robotereinsatz auf, die im vergangenen Jahr im Rahmen eines Projektseminars am Lehrstuhl für Informatik implementiert wurde.[2]
Da die Aufgabe der für die Entwicklung der Simulationsumgebung verantwortlichen Gruppe eher die Entwicklung der Umgebung selbst als die zur Minensuche einzusetzenden Roboter waren, kam uns die Aufgabe zu, die Intelligenz bzw. die Effizienz jener zu verbessern.
Im Rahmen dieser Ausarbeitung wird zunächst die Simulationsumgebung vorgestellt und in diesem Zusammenhang ihre Erweiterbarkeit diskutiert. Im weiteren Verlauf werden zwei neue Roboterklassen präsentiert, die Klasse `Bender` und ein für die Arbeitsteilung und die Kommunikation zwischen den Robotern verantwortlicher Roboter, der `H725`.

Aufbau

Zunächst werden einige der zum Verständnis der Simulationsumgebung als Multi-Agenten-System notwendige theoretische Grundlagen erläutert und Begrifflichkeiten geklärt.

[1] Quelle: http://www.landmine.de
[2] Download unter: http://wwwmath.uni-muenster.de/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe4b

Im Folgenden werden technologische und architektonische Grundzüge des Systems dargestellt und die Simulationsumgebung vorgestellt sowie bewertet.
Anschließend wird die von den Autoren der Simulationsumgebung entwickelte Roboterklasse `Robbi` auf Stärken und Schwächen analysiert. Auf diesen Ergebnissen aufbauend werden Konzepte zur Erweiterung und Verbesserung des vorliegenden Robotermodells skizziert und deren Realisierungsmöglichkeiten diskutiert.
Den Schluss der Arbeit bildet die Bewertung der Simulationsumgebung im Hinblick auf deren Erweiterbarkeit und die Möglichkeit der Programmierung neuer Roboterklassen sowie die Motivation weiterer alternativer Simulationsideen.

2. Theoretische Grundlagen

2.1. Der Agent

Für den Begriff des „Agenten" hat sich bis heute noch keine einheitliche Definition durchgesetzt. In den Wirtschaftswissenschaften wird jemand, der im Auftrag eines anderen handelt, als Agent bezeichnet, während in der Informatik der Begriff des Agenten für ein Stück Software oder ein System verwendet wird.
Für den Agenten aus Sicht der Informatik lassen sich jedoch in Anlehnung an menschliche Agenten einige charakteristische Eigenschaften hervorheben:
Ein Agent sucht und sammelt Informationen. Er erledigt komplexe Aufgaben und besitzt die Fähigkeit, sein Wissen zu erweitern (*Intelligenz*).

Die wichtigsten Eigenschaften eines Agenten sind[3]:

- *„Situated":* „sich in einer Umgebung befinden", Agenten nehmen ihre Umgebung über Sensoren wahr und verändern sie über Effektoren.
- *Reaktiv:* Agenten können Veränderungen in ihrer Umwelt registrieren und auf sie reagieren, indem sie ihr Verhalten umstellen.
- *Autonom:* Agenten bestimmen ihre Aktionen selbst.
- *Sozial:* Agenten kooperieren untereinander.
- *Rational:* Agenten besitzen Ziele und verfolgen diese.
- *Anthropomorph:* Agenten sollen sich dem Menschen ähnlich verhalten, ihnen werden Absichten und Wünsche zugeschrieben.

Agenten teilen sich ihre Umwelt mit anderen Agenten, es entstehen *Multi-Agenten-Systeme*.

2.2. Multi-Agenten-Systeme

Unter einem Multi-Agenten-System (MAS) versteht man ein System von mehreren Agenten. Nicht die individuellen Fähigkeiten der einzelnen Agenten stehen im Vordergrund, sondern Kooperation und Kommunikation zwischen den Agenten bzw. die Interaktion der Agenten mit ihrer Umwelt.

[3] Vgl. Klügl: Multiagentensimulation (2001), S. 14 f.

Auch für MAS lassen sich einige charakteristische Eigenschaften und Anforderungen aufzählen, die vom Grad der Autonomie der einzelnen Agenten determiniert werden[4]:

- Jeder Agent besitzt lediglich eine beschränkte Sicht auf das Gesamtsystem, d.h. er verfügt nur über einen begrenzten Grad an Information und beschränkte Problemlösefähigkeiten.
- Die Informationen mobiler Agenten sollten lokal gespeichert werden (dezentrale Datenverwaltung).
- Die Ausführung der Berechnungen und Aktionen sollte ebenfalls lokal erfolgen.
- Im Idealfall besteht in einem Multi-Agenten-System keine zentrale Kontrolle.

Die hier vorgestellte Simulationsumgebung erfüllt diese Anforderungen: Jeder Agent verfügt über individuelle Fähigkeiten (in Form der zur Verfügung gestellten Algorithmen) und individuelles Wissen. Es gibt keine globale Systemkontrolle, die Autonomie der Agenten wird nicht beschnitten. Das Wissen der einzelnen Agenten wird durch die Interaktion der Agenten mit ihrer Umwelt koordiniert.
Zur sinnvollen Interaktion bedarf es dabei einer gemeinsamen Agentensprache, eines gemeinsamen Interaktions-, bzw. Kommunikationsprotokolls und einer gemeinsamen Weltsicht.
Eine Organisation, die sich mit Standardisierung in Bezug auf Multi-Agenten-Systeme beschäftigt, ist die FIPA.

2.3. FIPA-Standards

Die FIPA (Foundation for Intelligent Physical Agents) ist eine 1996 gegründete Non-Profit-Organisation, die es sich zur Aufgabe gemacht hat, Standards für heterogene, interaktive Agentensysteme zu definieren.
Der Nachrichtenaustausch erfolgt hierbei über Low-Level-Protokolle in einer gemeinsamen Agentensprache, der FIPA-ACL (*Agent Communication Language*).
Eine FIPA-ACL-Nachricht enthält im Wesentlichen den Nachrichtentyp, die Teilnehmer (Sender, Empfänger), die Inhaltsbeschreibung der Nachricht, sowie Interaktionskontrollen, die das Interaktionsprotokoll und Nachrichtenkennungen festlegen.
Mögliche Nachrichtentypen können z.B. request, answer oder refuse sein. Die Inhaltsbeschreibung umfasst nicht nur den tatsächlichen Inhalt der Nachricht, sondern legt auch die verwendete Sprache, die Kodierung sowie die Ontologie der Nachricht fest.
Es existieren noch zahlreiche weitere Spezifikationen der FIPA, wie z.B. zur abstrakten Agentenarchitektur, zum Agentenmanagement und zum Nachrichtentransport über heterogene Netzwerke hinweg. Für unsere Arbeit haben sie allerdings keine praktische Relevanz.

3. Die Simulationsumgebung

3.1. Technologien: Jade[5]

Die Simulationsumgebung wurde mit Hilfe der MAS-Entwicklungsumgebung Jade (*Java Agent Development Framework*) implementiert. JADE ist ein Open-Source-Projekt der TILAB (Telecom Italia Laboratori) aus Italien, das vollständig in Java implementiert ist und ständig weiterentwickelt wird[6].

[4] Vgl. JenSycWool (1998).
[5] Ausführliche Beschreibungen vgl. Ausarbeitung Eibl, Sonnenberg, Sauerberg (WS 03/04)
[6] http://jade.tilab.com

Jade folgt den vorgestellten FIPA-Standards und stellt eine Bibliothek von Klassen zur Erzeugung von Agenten, eine Agentenplattform sowie eine Reihe grafischer Tools bereit.
Die durch das Jade-Framework verwaltete Agentenplattform ist verteilt und - da sie vollständig in Java implementiert ist - betriebssystemübergreifend. Das Ausführen der verschiedenen Aktivitäten der Agenten erfolgt nebenläufig.
Für die Interaktion zwischen den Agenten stellt Jade eine Bibliothek verschiedener FIPA-Interaktionsprotokolle bereit.

3.2. Architektur[7]

Neben den Roboteragenten umfasst die Simulationsumgebung noch die Administrationsagenten. Diese werden im Folgenden erläutert.

Serveragent

Der Serveragent ist beim Start der Simulationsumgebung für die korrekte Initialisierung aller relevanten Komponenten verantwortlich. Ferner registriert / deregistriert er Roboterinstanzen, sobald diese im System hinzugefügt / entfernt werden.
Zusätzlich zu diesem Aufgabenbereich bildet er die zentrale Schnittstelle zwischen den Robotern und dem Rest der Simulationsumgebung. Er regelt die gesamte Kommunikation zwischen den Roboterinstanzen und der simulierten Umwelt. Alle Anfragen (beispielsweise für eine Sensorinformation oder Bewegung) und Antworten werden von ihm bearbeitet.

Simulationsagent

Der Simulationsagent hat in der vorliegenden Version zwei Aufgaben: Er berechnet die Energieabzüge der Roboter durch Bewegung und Sensoreinsatz und besorgt die Informationen, die für die Abwicklung dieser Aktionen notwendig sind.
Da in der Realität selten ideale Bedingungen für die Sensoren existieren, wäre eine dritte Aufgabe dieses Agenten die Verfälschung von Informationen, die vom Feldagenten an die Roboterinstanzen weitergeleitet werden (so sollte der optische Sensor auch bei Regen in der Lage sein, aus den gegebenen Informationen vernünftige Schlussfolgerungen zu ziehen). In der aktuellen Version der Simulationsumgebung ist diese Funktionalität jedoch noch nicht implementiert.

Feldagent

Die Umwelt, die von den Robotern nach Minen durchsucht werden soll, wird in der Simulationsumgebung in Form einer in einzelne Felder unterteilten Karte repräsentiert.
Der Feldagent ist die Komponente, die die Informationen über den Zustand jedes einzelnen Feldes sowie die in ihm hinterlegten Sensorinformationen bereit hält. Außerdem ist er für die Erstellung des Spielfeldes und die Zustandsänderungen der Felder verantwortlich (z.B. Explosion einer Mine). Zusätzlich speichert er auch Position und Zustand jedes einzelnen Roboters.

Remote-Robot-Management-Agent

Die Ausführung des Simulationslaufes der Roboter kann auch verteilt erfolgen. Der Remote-Robot-Management-Agent unterstützt den Anwender dabei.

[7] Ausführliche Beschreibungen vgl. Ausarbeitung Eibl, Sonnenberg, Sauerberg (WS 03/04)

3.3. Software

Anforderungen an das System

Die Umgebung soll die Simulation einer Landminensuche durch Robotereinsatz ermöglichen. Neben den allgemeinen Anforderungen an ein Multi-Agenten-System[8] sind die Hauptanforderungen an ein solches System:

- *Bezug zur Realität:* Die Umwelt und das Verhalten der Roboter sollten möglichst realitätsnah modelliert werden können. Faktoren, die in diesem Zusammenhang eine Rolle spielen, sind beispielsweise Beschaffenheit des Terrains, Wetter- und Temperaturverhältnisse, Energieverbrauch der Roboter oder Kommunikation zwischen ihnen.
- *Benutzerfreundlichkeit:* Die zahlreichen Einflussfaktoren, die bei der Simulation der Landminensuche zu berücksichtigen sind, sollten die Handhabbarkeit der Simulationsumgebung nicht negativ beeinflussen. Vielmehr sollte es dem Anwender möglich sein, Karten ohne zu großen Zeitaufwand zu entwickeln, um die erstellten Roboterklassen testen zu können.
- *Erweiterbarkeit:* Die Entwicklung einer Simulationsumgebung, die sämtliche in der Realität für die Landminensuche relevanten Faktoren abbildet, ist ein langwieriger Prozess. Die Umgebung sollte deshalb nachträgliche Erweiterungen oder Korrekturen ermöglichen.

Modellierung der Umwelt

Das von den Robotern nach Minen zu durchsuchende Terrain wird in der Simulationsumgebung als Landkarte abgebildet. Die Erstellung der Landkarten wird mit Hilfe eines Editors möglich, kann aber auch vom System selbst in Form einer automatischen Kartengenerierung übernommen werden.
Der Editor ermöglicht es dem Anwender, auf zahlreiche Faktoren Einfluss zu nehmen:

- *Kartengröße:* Aufgeteilt in eine Anzahl von Feldern gleicher Größe (Kacheln) kann der Anwender die Größe der Karte (Breite, Höhe) festlegen.
- *Landschaftstyp:* Die Simulationsumgebung differenziert zwischen zwei verschiedenen Typen von Landschaften: Grasland und Wüste.
- *Felder:* Die einzelnen Felder repräsentieren ein Terrain von 2 m^2. Für jedes Feld kann der Nutzer verschiedene Einstellungen vornehmen:
    - *Feldtyp:* Auf der Karte können Feldwege, Straßen, Hindernisse, Wasser und Brücken abgebildet werden.
    - *Minen:* In den Feldern können Minen hinterlegt werden. Es besteht die Möglichkeit, Minen mit und ohne Metallgehalt zu platzieren.
    - *Höhenstufe:* Die Simulationsumgebung unterscheidet zwischen acht verschiedenen Höhenebenen, wobei die Differenz zwischen zwei benachbarten Feldern den Grad des Energieverbrauchs der Roboter determinieren.
    - *Bild:* Jedem Feld wird (manuell oder automatisch) ein Bild zugewiesen, um eine Auswertung über Foto- und Infrarotsensor zu ermöglichen.

Die Simulationsumgebung stellt zudem die Möglichkeit bereit, erstellte oder generierte Landkarten unter dem Dateityp `*.map` zu speichern.

[8] Vgl. Kap. 2.2.

Robotereinsatz

Die zur Minensuche einzusetzenden Roboter sollten Agenten-ähnliche Eigenschaften aufweisen[9]. Die Roboter der Simulationsumgebung werden deshalb als Agenten realisiert.
Roboter können intelligent und autonom agieren. Zudem interagieren Roboter mit ihrer Umwelt und handeln rational sowie sozial: Zum Erreichen ihrer Ziele kooperieren sie mit anderen Robotern.
Um einen Roboter in der Umgebung anzumelden, muss eine Instanz einer Roboter-Klasse gebildet werden. Der Roboter wird bei korrekter Eingabe im Feld (1,1) platziert.
Die Simulationsumgebung stellt den Robotern verschiedene Sensoren zur Verfügung, um die in den einzelnen Feldern hinterlegten Informationen abfragen zu können: Mit Hilfe des Metalldetektors wird ein Feld auf seinen Metallgehalt hin überprüft. Ein positiver Ausschlag muss dabei nicht zwangsläufig auf eine Mine hindeuten, Felder können auch Metall enthalten, wenn in ihnen keine Mine hinterlegt wurde. Da auch Minen ohne Metallgehalt modelliert werden können, stellt die Simulationsumgebung den Robotern die Möglichkeit bereit, mit Hilfe eines Foto-, bzw. Infrarotsensors eine Bildanalyse durchzuführen. Zum Erkennen von Hindernissen, d.h. Feldern, die ein Roboter nicht passieren kann, können die Roboter auf einen Stoßsensor zurückgreifen.
Die Simulationsumgebung verfügt über eine bereits vordefinierte Roboterklasse, die weniger zur Simulation als vielmehr zum Testen der Umgebung entwickelt wurde: die Klasse `ManualRobotAgent`.
Roboter dieser Klasse folgen keinem vordefinierten Algorithmus, sondern lassen sich ausschließlich manuell steuern.

3.4. Beurteilung

Diese (vorläufige) Bewertung bezieht sich mehr auf die Anwendbarkeit und Benutzerfreundlichkeit als auf die Erweiterbarkeit der Simulationsumgebung und die Möglichkeit der Implementierung neuer Roboterklassen. In den folgenden Kapiteln werden die Klasse `Robbi` und die neuen Roboterklassen `Bender` und `H725` vorgestellt. Erst diese Kapitel motivieren und veranschaulichen eine auf die Erweiterbarkeit bezogene Bewertung[10].

Installation

Die Installation der Simulationsumgebung gestaltete sich schwieriger als erwartet, da sich die aktuelle Version von Jade als inkompatibel erwies.
Deshalb musste die auf der Homepage des Seminars angebotene Version Jade 3.0 verwendet werden.

Benutzerfreundlichkeit und Handhabbarkeit

Einmal installiert stellte sich die Simulationsumgebung als äußerst leicht verständlich, erlernbar und intuitiv anwendbar heraus.
Gerade die Kartenerstellung erwies sich als gelungen: Die Möglichkeit der automatischen Kartengenerierung vereinfacht das Testen neuer Roboterklassen enorm. Der Anwender muss lediglich Rahmendaten wie Kartengröße oder Landschaftstyp angeben, den Rest der Karte erstellt die Simulationsumgebung. Ein weiteres nützliches Feature ist die Möglichkeit, vor der Generierung den Minen- und Wasseranteil sowie die Höhenstufe des Terrains in Form

[9] Vgl. Kap. 2.1.
[10] Vgl. Kap. 5.

von Prozentsätzen anzugeben. So können neue Roboterklassen in kurzer Zeit auf verschiedensten Terrains getestet werden, ohne eine Vielzahl von Karten manuell anfertigen zu müssen.
Aber auch bei der manuellen Kartenerstellung kommt die Simulationsumgebung dem Anwender entgegen: Die in den Feldern hinterlegten Bilder lassen sich automatisch, für mehrere Felder vorgesehene Einstellungen großflächig zuweisen.

Realitätsnähe

Die Kartenerstellung ist jedoch gerade deshalb für den Anwender so leicht handhabbar, weil sie sich auf relativ wenige beeinflussbare Faktoren beschränkt. Zudem besitzen diese Determinanten für die Simulation nur begrenzt Relevanz:
Eine Differenzierung zwischen den Landschaftstypen (Grasland, Wüste) erscheint beispielsweise nur dann sinnvoll, wenn deren unterschiedliche Umweltbedingungen berücksichtigt werden. Derartige Faktoren wirken sich in der Realität u.a. auf die Sichtverhältnisse, den Energieverbrauch oder die Kommunikationsmöglichkeiten der Roboter aus. In der aktuellen Version der Simulationsumgebung werden derartige Einflüsse jedoch nicht berücksichtigt.
Auch die verschiedenen Feldtypen beeinflussen nicht alle das Verhalten der Roboter: Felder, in denen Minen oder Hindernisse hinterlegt sind, können (oder sollen) vom Roboter zwar genauso wenig betreten werden wie Felder, in denen sich Wasser befindet. Jedoch ließ sich nicht feststellen, ob die Unterscheidung zwischen Feldweg, Asphalt und Boden Auswirkungen auf die Agenten hat (z.B. geringerer Energieverbrauch).

Zusammenfassend ist die Kartenerstellung leicht erlernbar, schnell durchführbar und relativ realitätsnah. Allerdings existieren in der Realität etliche Einflussfaktoren, wie beispielsweise Temperatur- und Wettereinflüsse die (noch) nicht in der Simulationsumgebung berücksichtigt sind.

Robotereinsatz

In der Simulationsumgebung werden die Roboter als Pfeile repräsentiert, deren Spitzen in die Richtung deuten, in die die Roboter blicken. Auf diese Weise kann der Anwender nicht nur nachvollziehen, wo sich die Roboter befinden, sondern auch, welche Felder sie gerade auf Minen überprüfen.
Die einzelnen Kacheln der Karte sind zunächst grau eingefärbt. Hat ein Roboter ein Feld kontrolliert, wird es „aufgedeckt“, die im Feld hinterlegten Informationen (Mine, Hindernis, o.ä.) werden sichtbar gemacht. Auf diese Weise lässt sich die Effizienz einer Roboterklasse sehr gut beurteilen: Benötigt ein Roboter einen langen Zeitraum, um eine Karte zu erkunden, kontrolliert er die Felder in der falschen Reihenfolge oder gar mehrmals.
Die grafische Gestaltung der Simulation ist gut gelungen. Die Simulationsumgebung zeichnet sich auch hier durch einige Features aus, die das Testen der Roboter vereinfachen: So können beispielsweise sämtliche Felder direkt aufgedeckt oder alle Minen sichtbar gemacht werden.
Besonders hervorzuheben ist die Möglichkeit, Felder auch auf Minen zu überprüfen, die kein Metall enthalten, stellt dies in der Realität doch eines der größten Probleme dar.

Leistung und Stabilität

Leider kam es im Laufe der Anwendung häufig zu Fehlern und ungewollten Abstürzen der Simulationsumgebung. Aus Gründen des Zeitaufwands ist davon abzusehen, mehr als vier Roboter gleichzeitig ein Terrain mittlerer Größe bearbeiten zu lassen.

4. Roboter

4.1. `Robbi`

`Robbi` ist eine von den Entwicklern der Simulationsumgebung implementierte Roboterklasse, die in dieser Seminararbeit - neben dem `ManualRobotAgent` - zum Testen der Simulationsumgebung und als Ausgangspunkt für die Entwicklung neuer Roboterklassen verwendet wurde.

4.1.1. Was `Robbi` kann…

`Robbi` verwendet zur Erkundung des Terrains einen Touchsensor und einen Metalldetektor. Ein zu untersuchendes Feld wird dabei zunächst mit dem Touchsensor überprüft: Befindet sich auf dem Feld ein Hindernis (oder ein anderer Roboter, den `Robbi` als ein solches erkennt), wird die Kachel in einer von ihm nach 100 Aktionen selbst erstellten Karte gelb eingefärbt. Wurde jedoch kein Hindernis in dem Feld hinterlegt, checkt `Robbi` das Feld mit dem Metalldetektor. Vermutet er in dem Feld eine Mine (positiver Ausschlag des Metallsensors), färbt er es rot ein. Ist dies nicht der Fall, wird das entsprechende Feld in seiner Karte grün markiert.
Um die Funktionalität des Algorithmus vollständig nutzen zu können, sollten zwei Roboter mit den Namen „Hans“ und „Herse“ kurz hintereinander angemeldet werden. Die beiden Roboter kooperieren miteinander: Trifft einer der Roboter auf eine in einem Feld hinterlegte Mine, so kontaktiert er den anderen, um ihm die Position der gefundenen Mine mitzuteilen. Dieser färbt die entsprechende Kachel in seiner Karte blau ein.
`Robbi` ist in der Lage sämtliche Minen aufzuspüren, sofern die Karte weder Wasserfelder enthält noch Felder, in denen Minen ohne Metallgehalt hinterlegt wurden. Zudem muss der Energievorrat ausreichend sein, um die Karte vollständig zu erkunden.

4.1.2. …was `Robbi` nicht kann: Verbesserungsmöglichkeiten

Zum Aufspüren der Minen verwendet `Robbi` lediglich einen Metalldetektor. Somit ist er nicht in der Lage, Minen zu entdecken, die kein Metall enthalten, selbst wenn diese sich über der Erde befinden. Foto- und Infrarotsensor wurden der Roboterklasse nicht zur Verfügung gestellt. Die logische Konsequenz hieraus ist, dass `Robbi` nicht nur äußerst anfällig für Felder ist, auf denen sich Minen ohne Metallgehalt befinden: Vielmehr kann er auch keine Felder betreten, die lediglich Metall - jedoch keine Mine - enthalten, da er dort eine Mine vermuten wird.
Roboterklassen sollte somit die Möglichkeit der Bildanalyse zur Verfügung gestellt werden.
Des Weiteren kann `Robbi` nicht feststellen, ob sich auf einem Feld Wasser befindet. Dies ist problematisch, da Roboter bei Betreten eines entsprechenden Feldes abgemeldet werden. Ein Roboter sollte Felder auf ihren Wassergehalt hin überprüfen, nachdem er festgestellt hat, dass dort kein Hindernis hinterlegt wurde (Feuchtigkeitssensor[11]).
`Robbi` benötigt zudem sehr lange, um eine Karte vollständig zu erkunden. Die Reihenfolge, in der er die einzelnen Felder überprüft, wirkte schon bei den ersten Simulationen willkürlich und wenig effizient: `Robbi` bewegt sich so lange in eine Richtung bis er auf ein Hindernis, eine Mine oder den Kartenrand trifft und wechselt dann *zufällig* seine Richtung. So kann es passieren, dass sich `Robbi` immer wieder „hin und her“ bewegt. Ein Roboter sollte sich so-

[11] Vgl. Kap. 4.2.4.

mit die Felder, die er bereits kontrolliert hat, merken, um die Wege, die er im Laufe der Simulation zurücklegen muss, möglichst kurz zu halten.
Auch der Einsatz zweier Instanzen der Klasse `Robbi` erwies sich als ineffizient, da die Roboter untereinander nicht koordiniert werden: Sie gehen oft zu lange Wege, kontrollieren Felder mehrmals und kommunizieren nur wenig miteinander. Die Roboter sollten also aufeinander abgestimmt werden.

4.2. `Bender`

Die Klasse `Bender` ist eine neu implementierte Roboterklasse, die ausgehend von den dargestellten Schwächen der Klasse `Robbi` implementiert wurde. Die verschiedenen Ansätze zur Verbesserung und deren Realisation werden im Folgenden vorgestellt.

4.2.1. Was `Bender` kann…

4.2.1.1. Rundumcheck

Die Unfähigkeit der Klasse `Robbi`, sich Felder zu merken, die bereits überprüft wurden, stellt einen der Hauptkritikpunkte dar und wurde in der Klasse `Bender` korrigiert.
Zudem betritt `Robbi` ein Feld (nachdem er es überprüft und für sicher befunden hat) direkt, während `Bender` zunächst sämtliche ihn umgebende Kacheln checkt (sofern nicht schon geschehen).
Für die Simulation ergeben sich daraus eine Reihe von Verbesserungen:

Vollständigkeit

Da sich `Robbi` bei der Erforschung der Karte so lange in eine Richtung bewegt, bis er auf ein Hindernis, eine Mine oder die Kartengrenze stößt, kann es passieren, dass er die Karte gar nicht komplett erforscht.

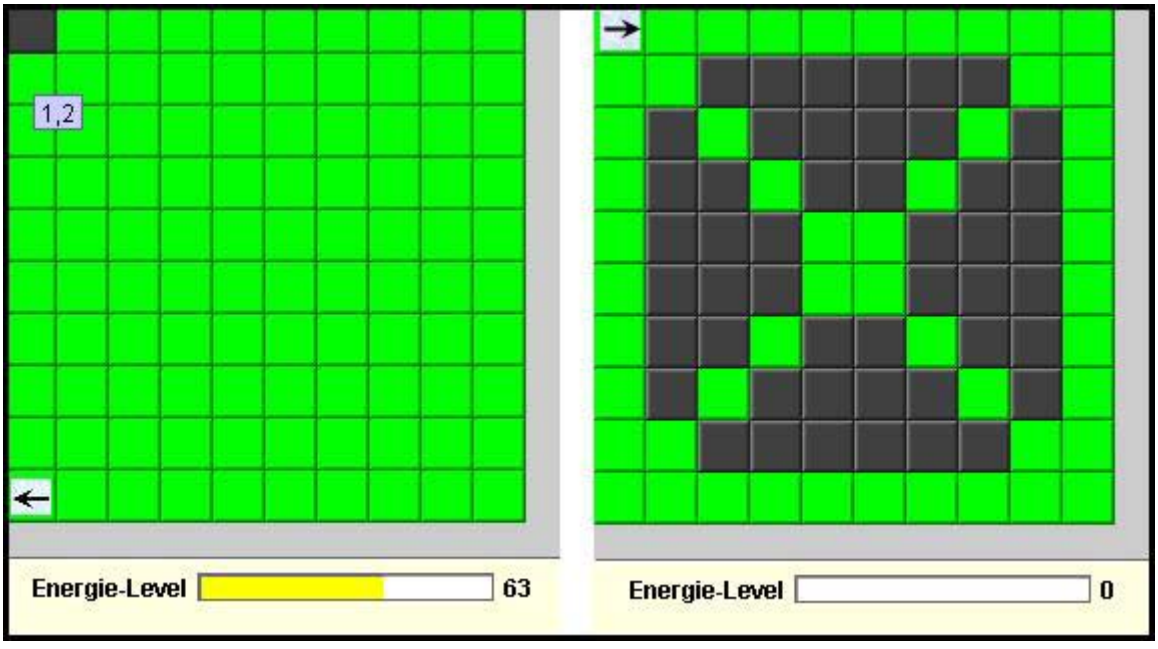
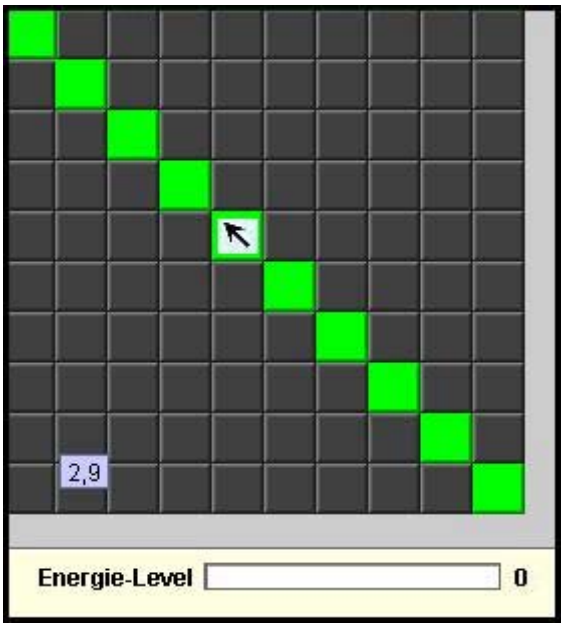


Abb.1: `Bender` vs. `Robbi`: Vollständigkeit

Abb. 1 stellt beispielsweise die Simulation einer Landminensuche in einem 10x10 Felder großen Terrain dar. In dieser Karte wurden in keinem der Felder Minen, Hindernisse, o.ä. hinterlegt.
Da die Richtungswechsel jeweils nur an den Kartengrenzen erfolgen können, ist `Robbi` nicht in der Lage, sämtliche Felder zu überprüfen.

Einsatzzeit

Die Einsatzzeit der Roboter (und damit auch die Dauer der Simulation) verringert sich erheblich. Dies lässt sich durch die Tatsache begründen, dass `Bender` sämtliche auf seinem Weg liegenden Felder überprüft und die Karte somit systematisch erforscht.



`Robbi` dagegen lässt auf seinem Weg viele unerforschte Felder zurück, so dass er sich sehr oft wieder zurück in die Nähe seiner Ausgangsposition bewegen muss, um diese Felder zu kontrollieren.
Da er sich auch nicht merkt, welche Felder er bereits überprüft hat, checkt er nicht nur einzelne Felder mehrmals, sondern bewegt sich auch häufig nach Beschreiten eines Weges direkt wieder zurück. Im Extremfall überprüft `Robbi` in der gewählten Karte nur 10 der 100 Felder (Abb.2).
Die Ursache für dieses Fehlverhalten liegt in der Tatsache begründet, dass `Robbi` zufällig entscheidet, welche Richtungsänderung er vollzieht.

Abb.2: Problematik des zufälligen Richtungswechsels

Energieverbrauch

Mit der Verringerung der Einsatzzeit sinkt auch der Energieverbrauch der Roboter.

4.2.1.2. Kommunikation

Die Klasse `Robbi` wurde so implementiert, dass sie am effizientesten beim gemeinsamen Einsatz zweier Instanzen, `Hans` und `Herse`, arbeitet. Die beiden Roboter kommunizieren miteinander, indem sie sich gegenseitig die Positionen gefundener Minen mitteilen. Dass die beiden Roboter ansonsten jedoch weitgehend unabhängig arbeiten, Felder wiederholt kontrollieren und sich gegenseitig behindern (beim Aufeinandertreffen der Roboter erkennen diese sich als Hindernisse), schien Motivation genug, ein Prinzip der Arbeitsteilung und Koordination zwischen den Robotern einzuführen.
Realisiert wurde dies durch die Implementierung einer neuen Roboterklasse: `H725`. Roboter dieser Klasse suchen selber keine Minen, sondern sind ausschließlich für die Kommunikation und Koordination der anderen Roboter zuständig. Hierzu verwaltet `H725` eine Liste sämtlicher angemeldeten Roboter und eine Karte, die ständig mit den von den Robotern ermittelten Sensorinformationen abgeglichen wird.

Zum Verständnis der Kommunikation und Koordination der Roboter soll die Funktionsweise der Klasse `H725` im Folgenden vorgestellt werden.
Sämtliche in der Simulationsumgebung angemeldeten Roboter kommunizieren nicht *direkt* miteinander, sondern *indirekt* über den `H725`. Es lassen sich vier verschiedene Situationen unterscheiden, in denen ein Roboter mit `H725` kommuniziert:

Anmeldung: Die Initialisierung einer neuen Instanz der Klasse `Bender` bewirkt nicht nur dessen Anmeldung in der Simulationsumgebung selbst, sondern auch bei `H725`. Auf diese Weise bekommt der sich anmeldende Roboter sämtliche Karteninformationen, die `H725` zur Verfügung stehen, übermittelt.

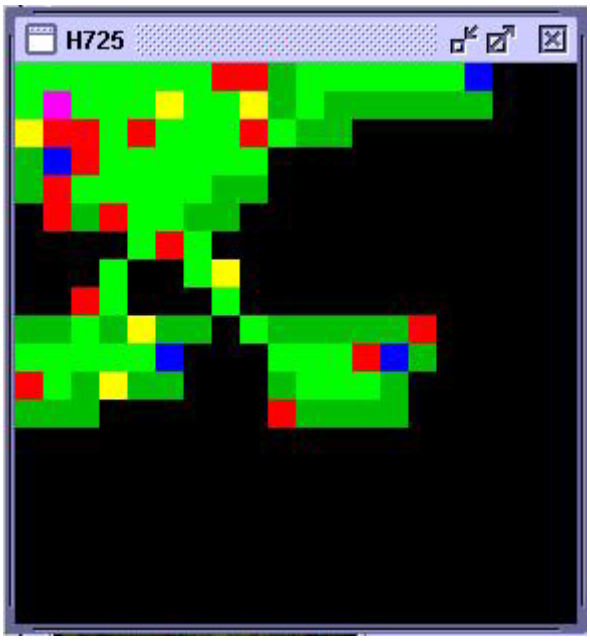
Abmeldung: Da `H725` eine Liste aller aktiver Roboter und deren Positionen verwaltet, ist es notwendig, dass sich ein Roboter auch bei ihm abmeldet.

Überprüfen eines Feldes: Hat ein Roboter ein Feld erforscht, teilt er die Beschaffenheit des Feldes `H725` mit. Dieser verwaltet sämtliche Feldinformationen in einer zentralen Karte, so dass sie - falls notwendig - korrigiert werden kann: Befindet sich ein Roboter auf einem Feld, das von einem anderen überprüft wird, erkennt dieser ihn als Hindernis und teilt `H725` dies mit. Da dieser jedoch die Positionen aller Roboter speichert, ist er in der Lage, die Fehleinschätzung des Roboters zu erkennen und es diesem auch mitzuteilen

Roboter ändert seine Position: Bewegt sich ein Roboter zu einem neuen Feld, übermittelt er `H725` seine neue Position.

Die Roboter erhalten somit ständig sämtliche zur Verfügung stehenden Karteninformationen: Nachdem einer der Roboter `H725` neue Sensordaten übermittelt hat, schickt dieser die Daten direkt an alle anderen Roboter.

Die Verwaltung der Informationen durch `H725` wird in Abb.3 dargestellt:



- Feld, auf dem eine Mine vermutet wird
- Feld, auf dem sich ein Hindernis befindet
- Feld, auf dem sich ein Roboter aufhält
- Feld, auf dem sich `H725` befindet
- Feld, das überprüft wurde
- Feld, das überprüft und betreten wurde

Abb.3: Verwaltung der Karteninformationen: `H725`

4.2.1.3. Arbeitsteilung

Um die Wahrscheinlichkeit einer Störung der Roboter untereinander möglichst gering zu halten, wird das Terrain von `H725` unter den Robotern aufgeteilt. Die Zerlegung des Terrains erfolgt dabei nach dem Flinsch-Einsatzgebiet-Zerlegungs-Algorithmus und wird von `H725` übernommen.

Der Flinsch-Einsatzgebiet-Zerlegungs-Algorithmus (FEZA)[12]

Der FEZ-Algorithmus unterteilt die gesamte Karte in eine Anzahl von Einsatzgebieten, die der Zahl der angemeldeten Roboter entspricht. Das logische Ziel bei der Aufteilung des Terrains ist dabei, den Robotern möglichst gleich große Einsatzgebiete zuzuweisen. Hierzu wird die Quadratwurzel aus der Anzahl der Roboter gezogen und mit ihrer Hilfe das Terrain unter ihnen aufgeteilt.
Unter Umständen kann es vorkommen, dass ein Gebiet zunächst von keinem der Roboter bearbeitet wird. Dies ist der Fall, wenn es sich bei der Anzahl der Roboter um eine Primzahl größer zwei handelt, da diese nicht für die Zerlegung sinnvoll geteilt werden kann. Haben jedoch alle Roboter (bis auf einen oder zwei) ihr Einsatzgebiet abgearbeitet, beginnen sie die komplette Karte aufzuklären, so dass keine unerforschten Sektoren zurückbleiben.

Zuweisung der Einsatzgebiete

Die Gebietszuweisung erfolgt nach dem Zeitpunkt der Anmeldung der Roboter: Der zuerst angemeldete Roboter bekommt dabei das am weitesten süd-östlich gelegene Einsatzgebiet, der jüngst instanzierte Roboter das am weitesten nord-westlich gelegene Einsatzgebiet zugewiesen.

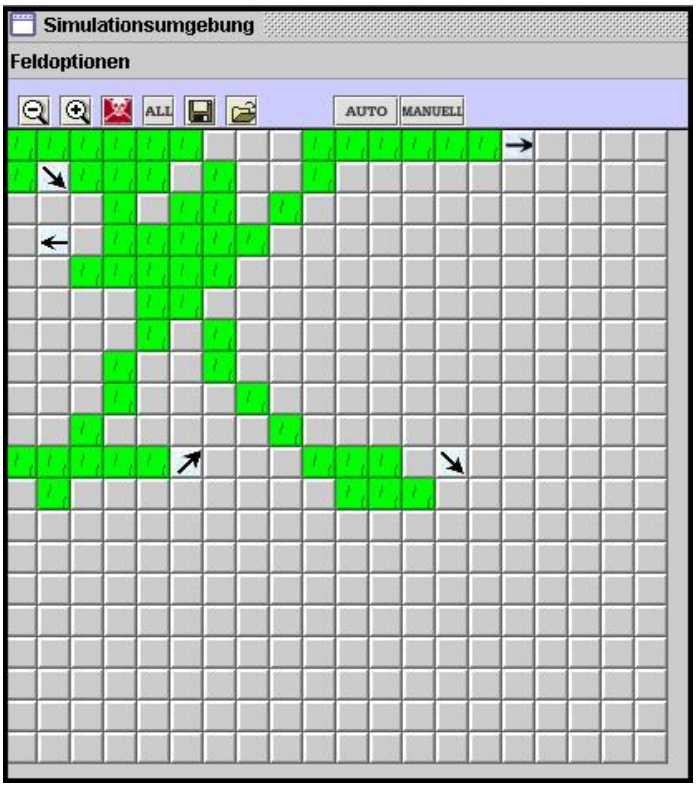


Abb. 4: Aufteilung in Einsatzgebiete

Dies resultiert aus der Tatsache, dass Roboter in der Simulationsumgebung auf dem nord-westlich gelegenen Feld (1,1) initialisiert werden. Die Wahrscheinlichkeit, dass sich ein kürzlich angemeldeter Roboter sehr weit Richtung Süd-Osten bewegt hat, ist dabei geringer als bei einem älteren Roboter, so dass sich die Wege, die die Roboter zu ihren Einsatzgebieten zurücklegen müssen, verkürzen.
Abb. 4 veranschaulicht die Aufteilung des Terrains in einzelne Einsatzgebiete anhand der Karte `metallkarte.map`: Der auf Feld (2,2) platzierte Roboter ist der `H725`.

[12] siehe Quellcode: `H725Commands.java`, Methode: `reorganizeWorkingArea`

Die Anzahl der Einsatzgebiete entspricht der Anzahl der Roboter. Jeder der Roboter überprüft die in seinem Einsatzgebiet gelegenen Felder. Während der am weitesten süd-östlich operierende Roboter als erstes initialisiert wurde, wurde der auf Feld (2,4) positionierte `Bender` als letztes angemeldet. Dieser wird trotz der späten Abmeldung sehr wahrscheinlich als erster sein Einsatzgebiet erforscht haben, da sämtliche anderen Roboter Teile seines Feldes auf dem Weg zu ihren Einsatzgebieten bereits überprüft haben.

4.2.1.4. Wegfindung

Die Roboter verfügen zu jeder Zeit über sämtliche zur Verfügung stehenden Karteninformationen, so dass sie auch feststellen können, welche Felder noch überprüft werden müssen. Für den Fall, dass einer der Roboter sämtliche umliegenden Felder bereits kontrolliert hat, muss er sich zu einem dieser Felder bewegen. Dies sollte möglichst effizient geschehen, d.h. der Roboter sollte in der Lage sein, den kürzesten Weg zu dem entsprechenden Feld zu berechnen.
Die in der Simulationsumgebung verwendeten Karten können als Graphen im mathematischen Sinn angesehen werden. Die Bewertungen der Kanten des Graphen entsprechen dabei den Entfernungen zwischen den einzelnen Feldern. Ein Algorithmus zur Wegfindung in Graphen ist der A*-Algorithmus[13]. Dieser Algorithmus ist ein informiertes Suchverfahren, das den kürzesten Weg zwischen zwei Knoten unter Einbeziehung einer Heuristik berechnet und wurde in der Klasse `Bender` implementiert.

4.2.2. …was Bender nicht kann

Auch wenn `Bender` im Vergleich zu `Robbi` erheblich effizienter arbeitet, kann es beim Einsatz durchaus zu Problemen kommen, die in dieser Arbeit noch nicht korrigiert wurden.

Fehlinformationen

Der gemeinsame Einsatz mehrerer Roboter kann im Falle von Begegnungen zur fehlerhaften Speicherung von Karteninformationen führen: Nachdem einer der Roboter ein Feld überprüft und für frei (kein Hindernis) und sicher (keine Mine) befunden hat, kann sich evtl. ein anderer Roboter vor ihm auf das Feld bewegen. Möchte der Roboter nun den `move`-Befehl ausführen, ist er nicht in der Lage, sich auf das entsprechende Feld zu bewegen. Dies resultiert daraus, dass jeder Roboter in einem eigenen Thread läuft, die Ausführungsreihenfolge der Threads jedoch vom Betriebssystem verwaltet wird. Zudem hat seine Programmlogik keine Informationen über den Ausgang der Aktion, da die `move`-Anweisung keinen Rückgabewert bereitstellt, anhand dessen der Roboter den Erfolg der Aktion beurteilen kann. So bleibt der Roboter auf dem Feld stehen, währen die Karte verzerrt wird.
Dieses Problem wird durch den Einsatz von `H725` zwar nicht behoben, dessen Eintrittswahrscheinlichkeit jedoch relativ gering gehalten: Da das Terrain in einzelne Einsatzgebiete zerlegt wird, sinkt auch die Wahrscheinlichkeit, dass sich die Roboter gegenseitig behindern. Da sämtliche Roboterinstanzen jedoch auf Feld (1,1) initialisiert werden, kann dieses Problem nicht vollständig ausgeschlossen werden.

[13] siehe Quellcode: `AStar.java`

Minensuche

Die Klasse `Bender` ist nicht in der Lage, Foto- oder Infrarotsensor zu verwenden, da der für diese Seminararbeit zur Verfügung stehende Zeitrahmen für die Implementierung derartiger Bildanalyse-Algorithmen nicht ausgereicht hätte.
Des Weiteren kann der `H725` keine Minen suchen. Es wäre durchaus denkbar und sinnvoll, die Aufgaben des `H725`, d.h. Kommunikation und Koordination der anderen Roboter, einer Instanz der Klasse `Bender` zu übertragen (z.B. dem zuerst initialisierten Roboter). Darauf wurde in dieser Arbeit allerdings aus Vereinfachungsgründen verzichtet.
Der `H725` bewegt sich nach seiner Initialisierung zudem in jedem Fall auf das Feld (2,2), das er auch nicht auf vorhandene Minen überprüft. Die Korrektur und Verbesserung dieses Verhaltens stellt allerdings kein Problem dar, wurde in der vorliegenden Klasse jedoch noch nicht vorgenommen.

4.3. Weitere - nicht realisierte - Verbesserungsmöglichkeiten

Mit denen im Rahmen dieses Seminars implementierten Erweiterungen ist die Palette von Verbesserungsmöglichkeiten noch längst nicht erschöpft. Es existieren noch eine Vielzahl von Ausbauvarianten, deren Realisierungen jedoch zu viel Zeit in Anspruch genommen hätten.

Arbeitsteilung

Eine offensichtliche Verbesserung wäre eine intelligentere Zuteilung der Arbeitsgebiete. Wie dargestellt, wird jedem Agenten bei dessen Anmeldung bei `H725` ein Einsatzgebiet (je nach Zeitpunkt der Initialisierung der Roboter) zugewiesen. Basierend auf der Annahme, dass sich Roboterinstanzen, die zu einem frühen Zeitpunkt erstellt wurden, mit einer höheren Wahrscheinlichkeit einem süd-östlichen Einsatzgebiet angenähert haben als Roboter, die später initialisiert wurden, werden diesen auch entsprechend gelegene Einsatzgebiete zugeteilt. Diese Zuweisung ist jedoch ineffizient, wenn die Positionen, auf denen sich die Roboter befinden, zu weit vom Einsatzgebiet entfernt sind.
Sinnvoller ist eine Methode, die den Robotern ihre Suchräume so zuweist, dass die Summe der Strecken von den aktuellen Positionen zu den jeweiligen Einsatzgebieten minimiert wird.
Des Weiteren wäre eine an den Landschaftsgegebenheiten orientierte Gebietseinteilung sinnvoll: Ein für einen Roboter ungünstig gelegener, weil schlecht erreichbarer, Sektor sollte besser von einem für die Bearbeitung des Sektors günstiger positionierten Roboter übernommen werden.
Zudem ist eine dynamische Einteilung und Zuweisung der Einsatzgebiete wünschenswert. Die Flächenberechnung der Einsatzgebiete sollte sich dabei am Fortschritt der Simulation orientieren, so dass jedes Gebiet dieselbe Menge an ungeprüften Land zugewiesen bekommt. In der aktuellen Version ist es zumeist so, dass der zuletzt initialisierte Roboter seine Arbeit *zuerst* beendet, da die anderen Roboter auf dem Weg zu ihren Einsatzgebieten Teile seines Terrains überprüfen. `Bender` stellt seine Arbeit außerdem nach Aufklärung seines Einsatzgebietes ein. Effizienter wäre die erneute Zuweisung eines noch nicht erforschten Gebietes.
Auf diese Weise sind sowohl Zeit- als auch Materialersparnisse möglich, da die bereits vorhandenen Ressourcen effektiver verteilt würden.

Rundumcheck

Die Reihenfolge, in der Bender die noch nicht erforschten Felder überprüft, motiviert einen weiteren Optimierungsansatz, der in der vorliegenden Version nicht realisiert wurde. Bender checkt zunächst alle umliegenden Felder und bewegt sich dann - falls möglich - nach Nord-Osten.

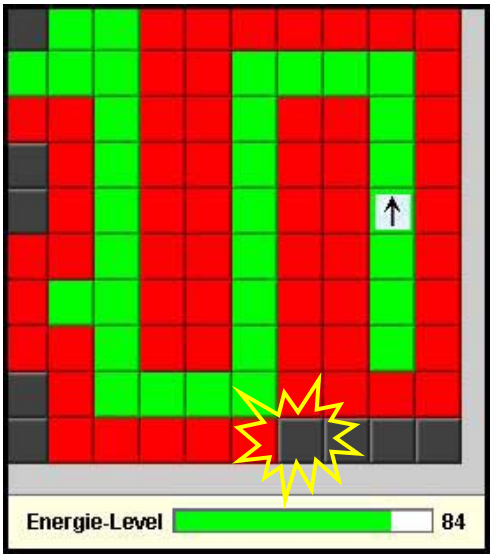


Abb. 5: Bender kehrt um

Diese Strategie ist nicht optimal, da die Möglichkeit besteht, Felder zu übersehen, die mit geringem Aufwand von der aktuellen Position des Roboters aus überprüft werden könnten. Zu einem späteren Zeitpunkt muss `Bender` daher wieder zurückkehren, um diese Felder aufzuklären.
Abb. 5 verdeutlicht diesen Sachverhalt und zeigt eine `Bender`-Instanz, die es versäumt hat, das gelb markierte Feld auf ihrem Weg zu überprüfen. Da dieses Feld das einzige noch zu kontrollierende Feld auf der Karte darstellt, muss `Bender` umkehren.

5. Beurteilung der Simulationsumgebung

5.1. Entwicklung neuer Roboterklassen

Die Realisierung eines einfachen Roboters erwies sich aus programmiertechnischer Sicht als relativ simpel, da sich die elementaren Funktionen eines Roboteragenten auf einige wenige beschränken. Die Durchführung einfacher Funktionen wie Änderung der Blickrichtung, Sensoreinsatz oder Fortbewegung stellten keine Probleme dar. Schwieriger gestaltete sich die sinnvolle Kombination der einzelnen Handlungskomponenten, aus denen die künstliche Intelligenz und das Verhalten resultierten.
So scheiterte beispielsweise der Versuch, eine zu jedem Zeitpunkt konsistente Darstellung der Karte zu gewährleisten, an der Thread-basierten Ausführung der Roboteragenten in Verbindung mit fehlenden Rückgabeinformationen der Simulationsumgebung[14].
Die Simulationsumgebung vereinfachte zwar das Testen neu entwickelter Roboterklassen durch die Möglichkeit der automatischen Kartengenerierung, allerdings erwies sie sich im Laufe der Arbeit als relativ instabil und langsam, so dass sich eine Simulation mit vielen Robotern als sehr zeitaufwendig herausstellte.
Zusammenfassend stellte sich weniger die Implementierung als vielmehr die Entwicklung einer sinnvollen Logik als die komplexere Aufgabe heraus.

[14] Vgl. Kap.4.2.2.

5.2. Erweiterbarkeit der Simulationsumgebung

Die Möglichkeit der Erweiterung der Simulationsumgebung hat sich als relativ komfortabel erwiesen. Der Quellcode ist nachvollziehbar gehalten und an den meisten Stellen verständlich kommentiert.
Im Folgenden wird die Erweiterung der Simulationsumgebung um eine weitere Funktion der Roboteragenten in Form eines Feuchtigkeitssensors beschrieben. Dieser Sensor dient dazu, Wassergebiete zu erkennen, um zu verhindern, dass ein Roboter ein solches Feld betritt. Der Feuchtigkeitssensor ist dem Touch-Sensor sehr ähnlich, da eine Information über die Beschaffenheit der Landschaft „abgefragt" wird. Demzufolge haben wir in der ersten Phase der Realisierung nach sämtlichen Vorkommnissen des Touch-Sensors gesucht und diese analysiert. Auf diesen Kenntnissen aufbauend wurde in der zweiten Phase der Feuchtigkeitssensor analog implementiert. Oftmals konnte der Code sogar direkt übernommen werden, da es sich häufig nur um eine Weiterreichung von Methodenaufrufen handelte. Die entscheidende Stelle im Code stellte sich sogar als noch einfacher zu implementieren heraus als der Touch-Sensor. Dieser überprüft das entsprechende Feld zunächst auf ein Hindernis und kontrolliert danach, ob auf dem besagten Feld noch ein anderer Roboter steht. Im Gegensatz dazu muss der Feuchtigkeitssensor lediglich prüfen, ob das entsprechende Feld ein Wassergebiet ist.
Leider stellte sich die Einbindung dieser Erweiterung in der Praxis für uns als unmöglich heraus. Nach der Kompilierung des geänderten Codes war die Simulationsumgebung nicht mehr zu benutzen, da fortwährend Exceptions geworfen wurden, und infolgedessen Benutzereingaben nicht mehr abgearbeitet wurden.
Eine fehlerhafte Programmierung unsererseits konnten wir recht bald ausschließen, da der ursprüngliche Quellcode dieselben Fehler verursachte. Die Simulationsumgebung lief also nur mit den vorkompilierten Dateien fehlerfrei.
Schließlich haben wir einen Workaround implementiert, der einen Feuchtigkeitssensor simuliert. Diese Methode macht sich einen Sonderfall der Simulationsumgebung zunutze: Der Infrarot-Sensor liefert nämlich im Falle eines Wassergebiets - und zwar nur dann - den String „null" zurück. Auf diese Notlösung haben wir jedoch nur aus Präsentationszwecken zurückgegriffen. Der Code des ordnungsgemäßen Feuchtigkeitssensors befindet wurde jedoch beigefügt und sollte mit einer korrekten unkompilierten Version der Simulationsumgebung funktionieren.

6. Alternative Simulationsideen

Durch die offene Gestaltung sowie der Möglichkeit der Erweiterung der Simulationsumgebung in Form einer Annäherung an realistischere Verhältnisse ergeben sich eine Vielzahl an alternativen Simulationsideen, denen das vorliegende Simulationssystem als Grundlage dienen könnte.
Ein weiterer Schritt in Richtung Realität ist die Verstetigung des diskreten (gekachelten) Bewegungssystems durch eine auf Vektoren basierende Umgebung. Aus dieser Umstellung resultiert ein hohes Maß an Flexibilität und Dynamik, da man von Himmelsrichtungen und befahrenen Flächen in Form von Kacheln Abstand nehmen kann.
Eng verwoben mit der Umstellung auf ein Vektorsystem ist der Einbezug der dritten Dimension in die Simulationsumgebung, die ebenfalls anhand von Vektoren abgebildet werden kann. So wäre eine viel realistischere Simulation von Geländetypen - wie z.B. Hügel oder auch Schlamm - möglich. Ferner könnten das Umkippen von Robotern und etwaige Reaktionen darauf simuliert werden.
Aber auch in Bezug auf den Minensucheinsatz ergeben sich neue Möglichkeiten. Nicht berücksichtigt werden aktuell die Bewältigung von Missionszielen, beispielsweise die Schnei-

senbildung für Konvois in einem minenverseuchten Terrain, oder die partielle Räumung eines Gebietes, in dem primär landwirtschaftlich nutzbares Gebiet gesäubert wird. Vernachlässigt wurde auch die Möglichkeit, bereits mit einer Informationsbasis die Minenräumung zu beginnen (beispielsweise indem man die Karte des Gebiets besitzt, jedoch Unkenntnis über den Aufenthaltsort der Minen hat). Mit erweitertem Wissen über das Gebiet ließen sich Algorithmen programmieren, die effizienter arbeiteten als solche, die völlig ohne derartige Informationen auskommen müssten. Im Zeitalter von satellitengestützten Aufklärungs- und Informationssystemen sicherlich eine realistische Annahme.
Eine Möglichkeit, Effektivität und Effizienz der Roboterklassen bezüglich eines an sie gestellten Problems zu beurteilen, bietet die Implementierung von Bewertungskriterien für einen Einsatz. Dies können triviale Funktionen wie Energieverbrauch, Zeitbedarf oder Materialverlust sein, aber auch komplexere oder gekoppelte Bewertungsfunktionen sind denkbar.
Eine letzte, für das Handling relevante Verbesserungsmöglichkeit ist das Hinzufügen einer Pause-Funktion beziehungsweise einer Replay-Funktion für die geordnete Beobachtung des Simulationsverlaufes.

7. Anhang

7.1. Literaturverzeichnis

- Franziska Klügl: Multiagentensimulation (2001)

- JenSycWool (1998)

- http://www.landmine.de

- http://jade.tilab.com

- Ausarbeitung Eibl, Sonnenberg, Sauerberg (WS 03/04):
  http://wwwmath.uni-muenster.de/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe4b/