Handbuch

Erstellung eines MultiAgentenSystems (MAS) mit Simulationsumgebung zur Roboter-Minensuche

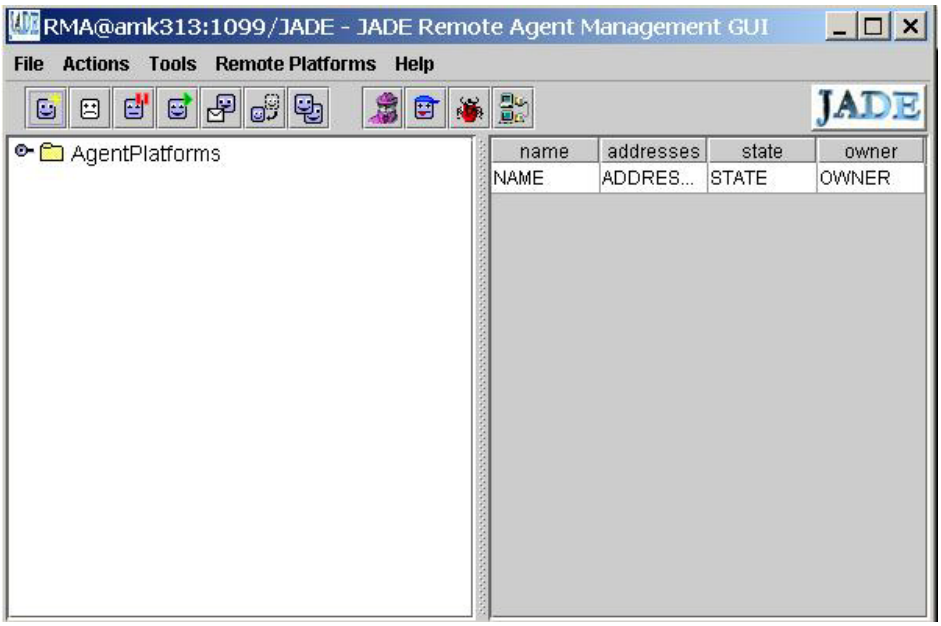
Verfasser: Dominik Eibl, Chistian Sonnenberg, Kolja Sauerberg

Inhaltsverzeichnis

1 Installation

Im Folgenden wird eine kurze Anleitung zur Installation der Simulationsumgebung zur Erkennung von Landminen gegeben. Das Programm benötigt als minimale Voraussetzung Java (ab Version 1.4, jdk oder run time environment) und die Jade-Plattform. Diese lassen sich bei www.sun.com bzw. http://jade.tilab.com kostenlos herunterladen. Wie sich Java installieren lässt, entnehmen Sie bitte der ausführlichen Beschreibung von Sun.

Kopieren Sie anschließend JADE in ein Verzeichnis ihrer Wahl. Und binden Sie folgende Dateien in den CLASSPATH ein: jade.jar, iiop.jar, base64.jar und jadeTools.jar (bei WIN XP -> Arbeitsplatz Rechtsklick → Eigenschaften → Erweitert → Umgebungsvariablen). Diese Dateien befinden sich im lib-Verzeichnis des Jade-Verzeichnisses. Zum Testen rufen Sie die Eingabeaufforderung auf und geben nach einem eventuellem Neustart java jade.Boot –gui ein. Sind die Pfade richtig gesetzt, so öffnet sich folgendes Fenster:



Hierbei handelt es sich um die grafische Benutzerschnittstelle der JADE-Software (eine detaillierte Beschreibung von JADE entnehmen Sie bitte dem JADE-Handbuch). Diese benötigen Sie zwar im Moment nicht, aber der erfolgreiche Aufruf zeigt, dass Java und Jade korrekt installiert sind. Anschließend können die Dateien

(hauptsächlich jpegs und class-Dateien) der Simualtionsumgebung in ein Verzeichnis Ihrer Wahl kopiert werden; auch dieses Verzeichnis muss im CLASSPATH mit eingebunden werden. Nachdem dies erfolgt ist, rufen Sie bitte die Eingabeaufforderung auf, wechseln in das Verzeichnis, in dem die Dateien der Simulationsumgebung liegen und geben folgenden Befehl ein:

```
java jade.Boot server:ServerAgent
```

Bitte achten Sie auf Groß-/Kleinschreibung. Daraufhin öffnet sich folgendes Fenster:

Wenn Sie dieses Bild sehen, ist die Installation erfolgreich verlaufen, und Sie können mit dem nächsten Kapitel fortfahren.

2 Tutorial

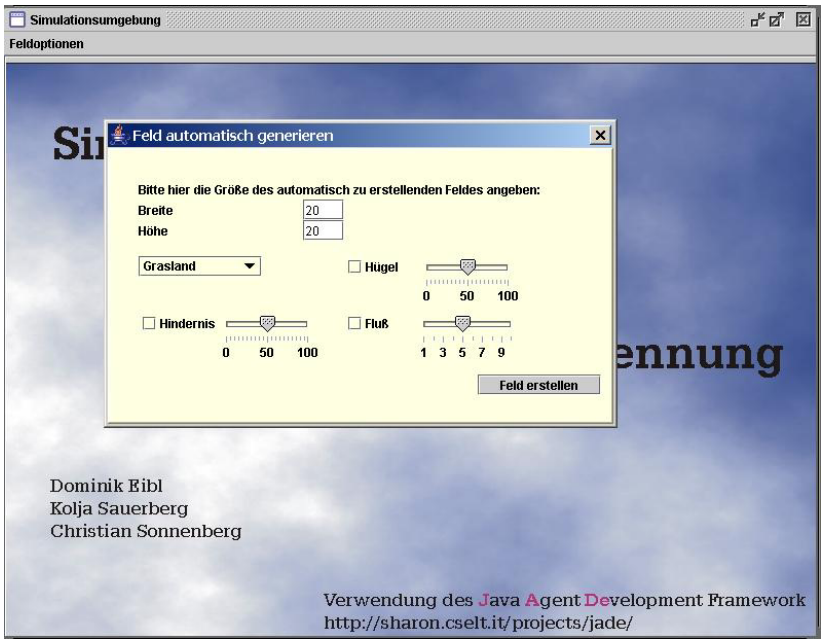
2.1 Spielfeld erstellen

Nach dem Start der Simulationsumgebung klicken Sie auf das Menü ***`Feldoptionen`***.

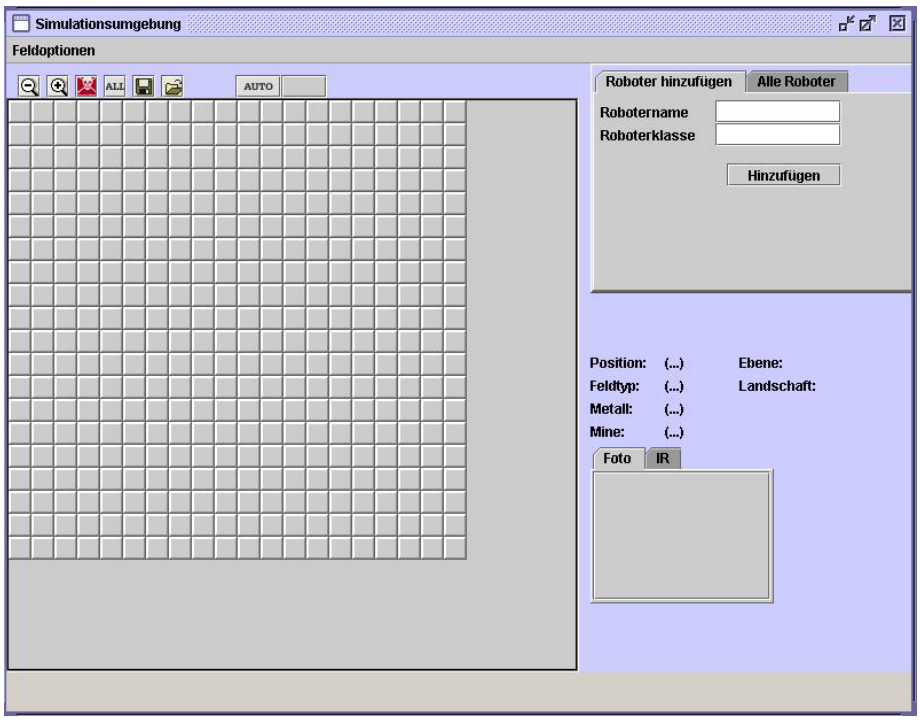
Von hier aus haben Sie die Möglichkeit, eine Landkarte automatisch generieren zu lassen, eine Karte manuell zu erstellen, eine bereits erzeugte Karte nachzubearbeiten, eine Karte zu speichern und eine Karte zu laden.

2.1.1 Automatische Erstellung

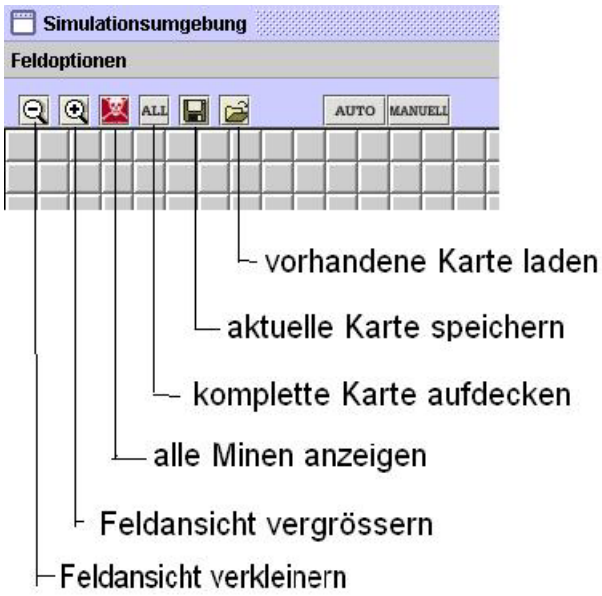
Klicken Sie auf Karte automatisch erstellen, und es erscheint folgender Dialog:



Hier können Sie die Höhe und Breite Ihres Simulationsfeldes festlegen; der Standardwert erzeugt ein 20x20 Kacheln großes Feld. Weiter haben Sie die Wahl zwischen 2 Landschaftstypen (***Grasland*** und ***Wüste***), verschiedenen Intensitäten an Hindernis- und Hügelvorkommen sowie der Einstellung der Flußbreite. Anschließend klicken Sie auf ***Feld erstellen*** und es wird ein solches automatisch generiert:



Die Kacheln stellen das noch verdeckte 20x20 Feld dar, in dem die Roboter später nach Minen suchen sollen. Die Symbole über diesem Feld haben folgende Bedeutung:

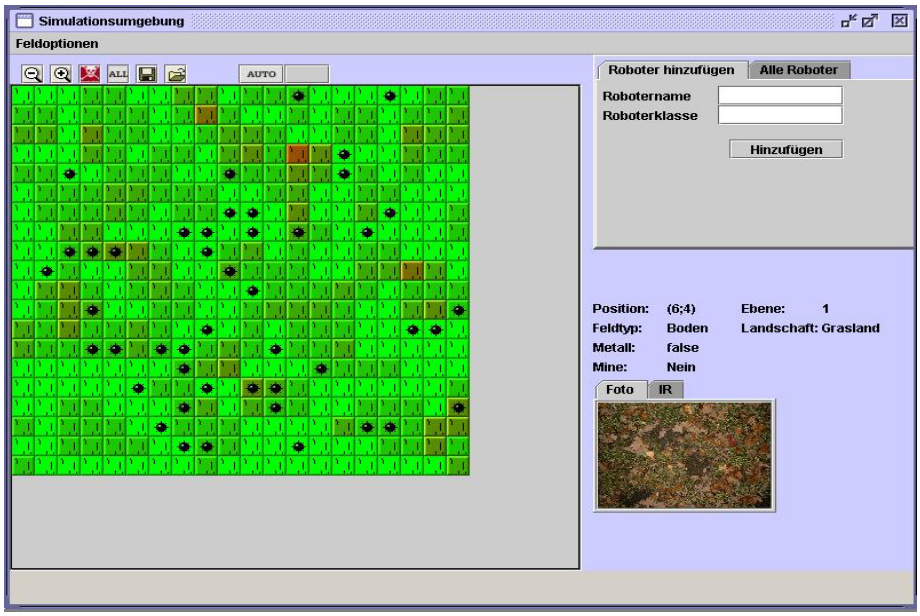


Mit den Schaltflächen „Auto“ und „Manuell“ können Sie eine neue Karte generieren; das geschieht wie gerade gesehenen automatisch oder manuell (s. u.).

Das Speichern und Laden von Karten ist nützlich, wenn Sie Simulationen mit identischen Umweltsituationen wiederholen möchten (z.B. um verschiedene Algorithmen zu testen und zu verbessern). Klicken Sie zum Speichern auf das Diskettensymbol und wählen Sie einen Namen für diese Karte. Die Datei wird im gewählten Verzeichnis als .map-Datei gespeichert.

Zum Öffnen einer bereits vorhandenen map-Datei, klicken Sie auf das entsprechende Symbol und wählen die gewünschte Datei aus.

Wenn Sie auf den Button mit der Aufschrift „All“ klicken, wird das verdeckte Feld aufgedeckt. Das könnte folgendermaßen aussehen:

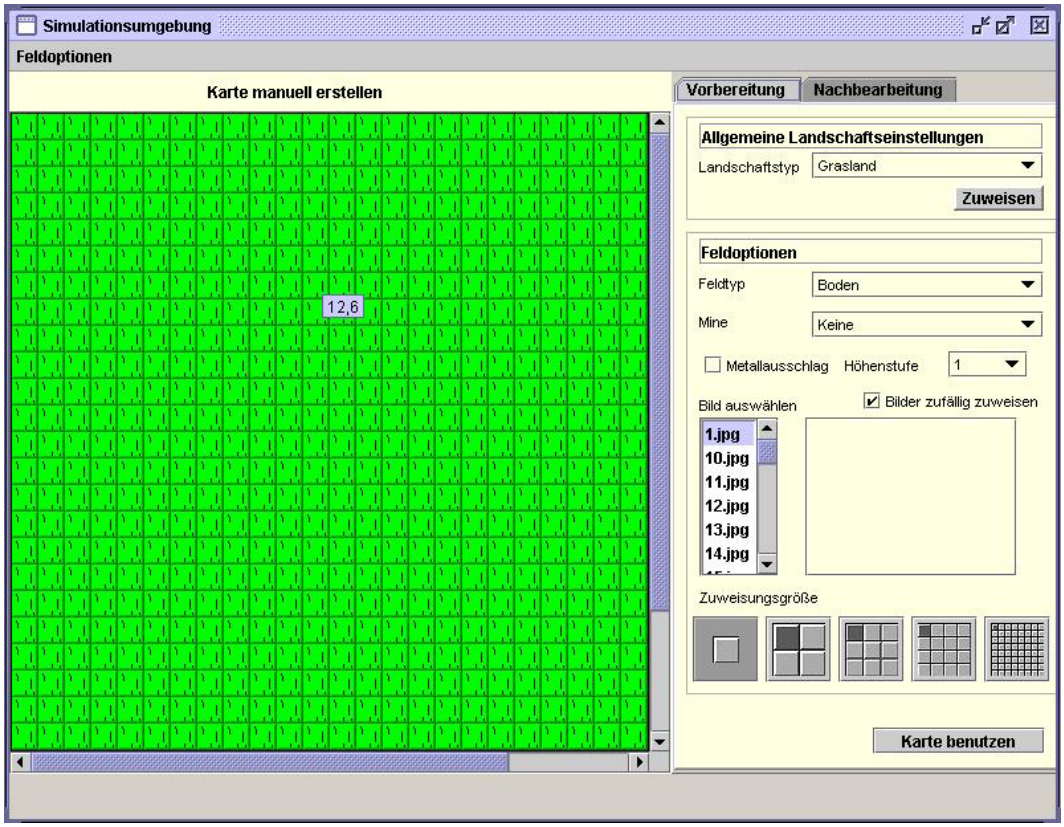


Durch Klicken auf die jeweiligen Kacheln erhalten Sie Informationen zu diesem Feld. Diese werden auf der rechten Seite angezeigt. Dazu gehören die Position, der Feldtyp (Brücke, Boden, Feldweg, Asphalt etc), der boolesche Metallwert, das Vorhandensein einer Mine, die Höhenebene und der Landschaftstyp. Die Höhenebenen haben den Zweck, unterschiedliche Energieabzüge von Robotern vorzunehmen, wenn diese sich über das Feld hinweg bewegen. Pro Kachel werden ein Kamerabild sowie ein Infrarotbild angezeigt. Die Bilder sind in Bilderverzeichnissen vorgehalten und können beliebig ausgetauscht werden. Dazu müssen sie lediglich durchnummeriert werden. Bevor es nun zum Robotereinsatz geht, soll noch kurz die manuelle Kartenerstellung vorgestellt werden.

2.1.2 Manuelle Erstellung

Dazu wählen Sie bitte in der Leiste oben unter Feldoptionen den Menupunkt ***Karte manuell erstellen***. Es erscheint ein Dialog, in dem Sie nach der Größe des Feldes gefragt werden:

Anschließend klicken Sie auf ***Ok***, es erscheint folgende Oberfläche:



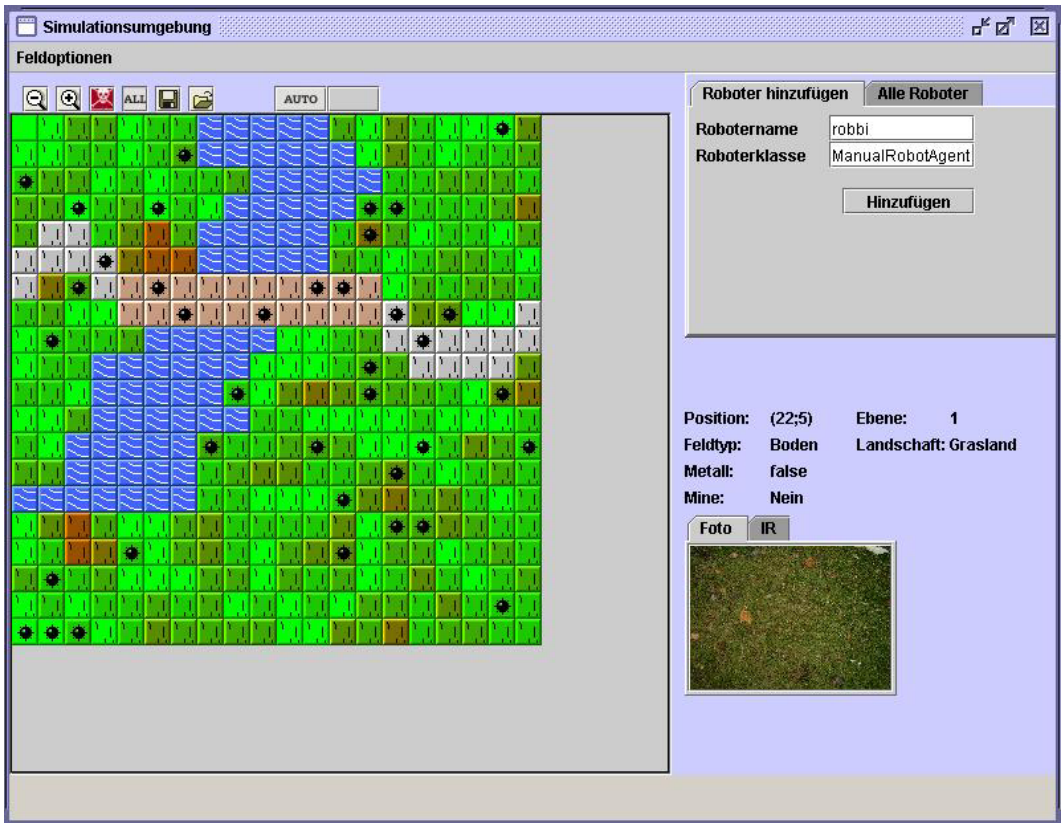
Hier können Sie zunächst einen Landschaftstypen (***Grasland*** oder ***Wüste***) wählen und diesen der Karte zuweisen; dieser Typ gilt für die gesamte Karte. Aus den übrigen Dialog-Feldern geht hervor, wie Sie die einzelnen Felder ausstatten können. Dazu stehen Ihnen die von der automatischen Kartengenerierung bekannten Größen zur Verfügung. Dies sind die Festlegung des Feldtyps (Boden, Fluss, Hindernis etc), das Vorhandensein einer Mine, der Metallausschlag sowie die Höhenstufe. Es wird explizit zwischen Metallausschlag und Vorhandensein einer Mine unterschieden, da nicht jede Mine Metall enthalten muss. Es ist nicht möglich eine Mine in den Fluss zu setzen; das Programm berücksichtigt ausschließlich Landminen. Zusätzlich können

Sie pro Kachel ein Bild zuordnen oder ein Häkchen bei ***Bilder zufällig zuweisen*** setzen. So wird jeder Kachel zufällig ein passendes Bild zugeordnet. Wählen Sie als letztes noch eine Zuweisungsgröße. So können Sie einzelne Felder oder aber größere Bereiche mit denen von Ihnen gewählten Attributen belegen. Diese Felder haben dann alle dieselben Werte. Um eine Kachel bzw. die Bereiche um diese Kachel mit Ihren gewählten Werten zu belegen, klicken Sie auf die gewünschte Kachel. Sind Sie mit dem Erstellen der Karte fertig, können Sie die Karte durch ein Klick auf den Reiter mit der Aufschrift „Nachbearbeitung“ nachbessern oder aber die Erstellung der Karte durch ein Klick auf den Button „Karte erstellen“ abschließen. Das Feld sollte aus Performancegründen 100x100 Kacheln nicht deutlich überschreiten.
Nachdem die Felderstellung abgeschlossen ist, wird nun der Robotereinsatz beschrieben.

2.2 Robotereinsatz

2.2.1 Ein Roboter zu Testzwecken – der ManualRobotAgent

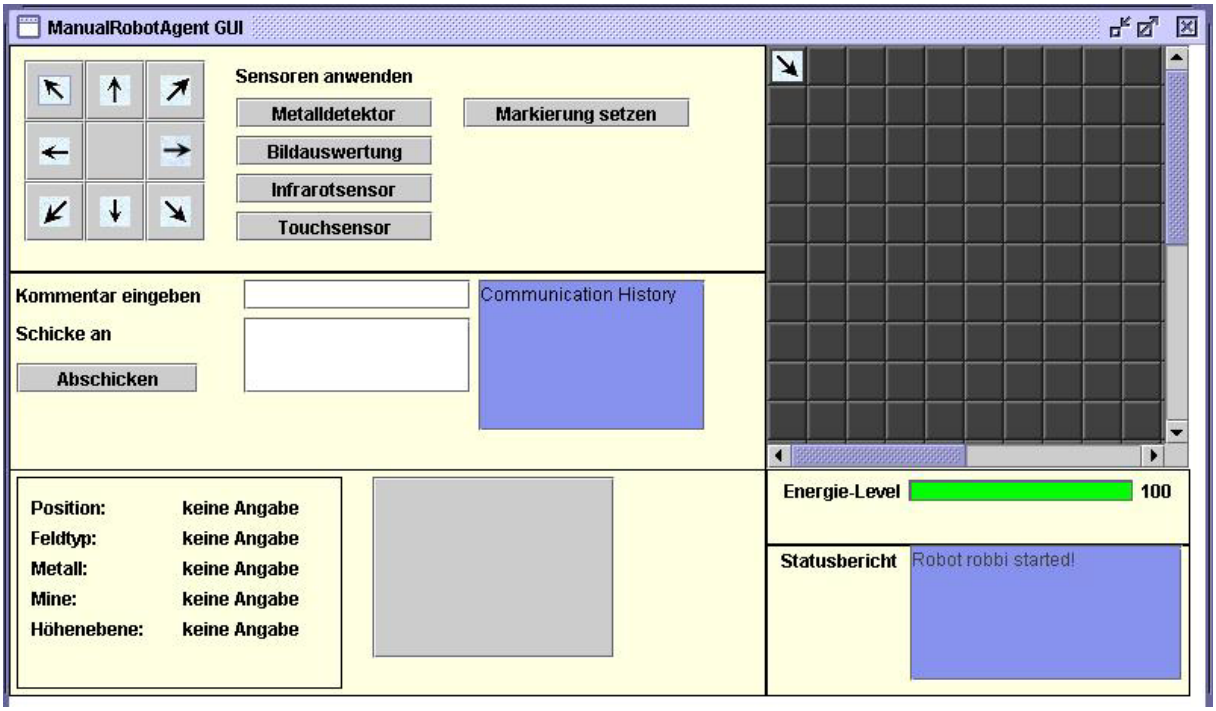
Um einen neuen Roboter in die bereits geschaffene Umwelt zu registrieren, wählen Sie zunächst einen Namen für den Roboter aus (Namen dürfen im System nur einmal vorkommen). Anschließend müssen Sie angeben, welche Java-Klasse dieser Roboter haben soll. Auch Ihre selbst programmierten Roboter können so angemeldet werden. (siehe Beispiel weiter unten).



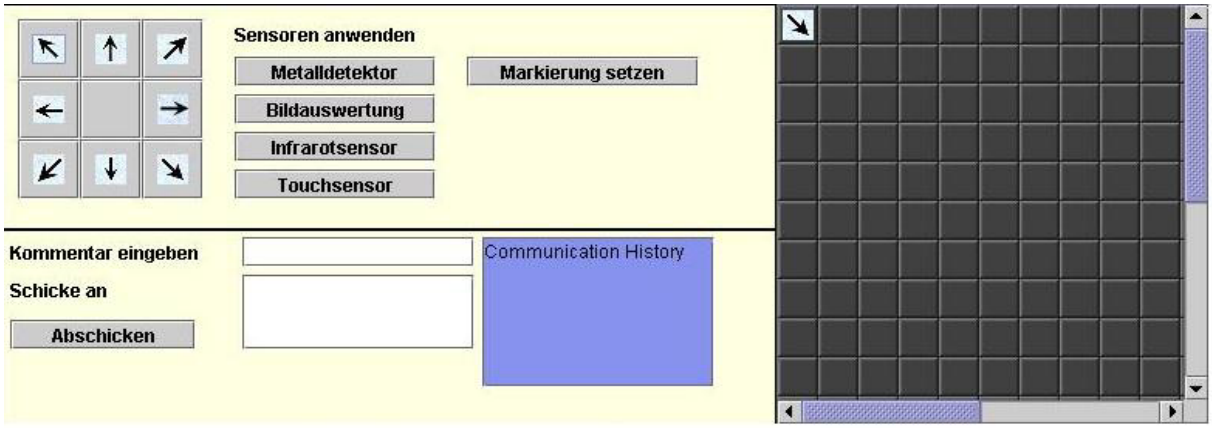
Klicken Sie auf Hinzufügen, und der Roboter erscheint als Pfeil oben links in der Simulationsumgebung. Folgende Prämissen sind dabei zu berücksichtigen: Die Pfeilrichtung gibt immer die Blickrichtung des Roboters wieder. Roboter sind nicht Wasser tauglich, sie werden abgemeldet, wenn sie sich ins Wasser bewegen. Roboter werden ebenso abgemeldet, wenn sie sich auf eine Mine bewegen oder sich unmittelbar neben einer explodierenden Mine befinden. Eine Mine zerstört zusätzlich alles, was sich auf den direkt anliegenden Kacheln befindet. Bei der Detonation einer Mine kann es zu Kettenreaktionen kommen, wenn eine weitere Mine direkt daneben liegt.

Ein Beispiel für eine Roboterklasse ist die bereits implementierte Klasse ManualRobotAgent. Diese Roboter folgen keinem vorher implementierten Algorithmus, sondern lassen sich ausschließlich manuell steuern.

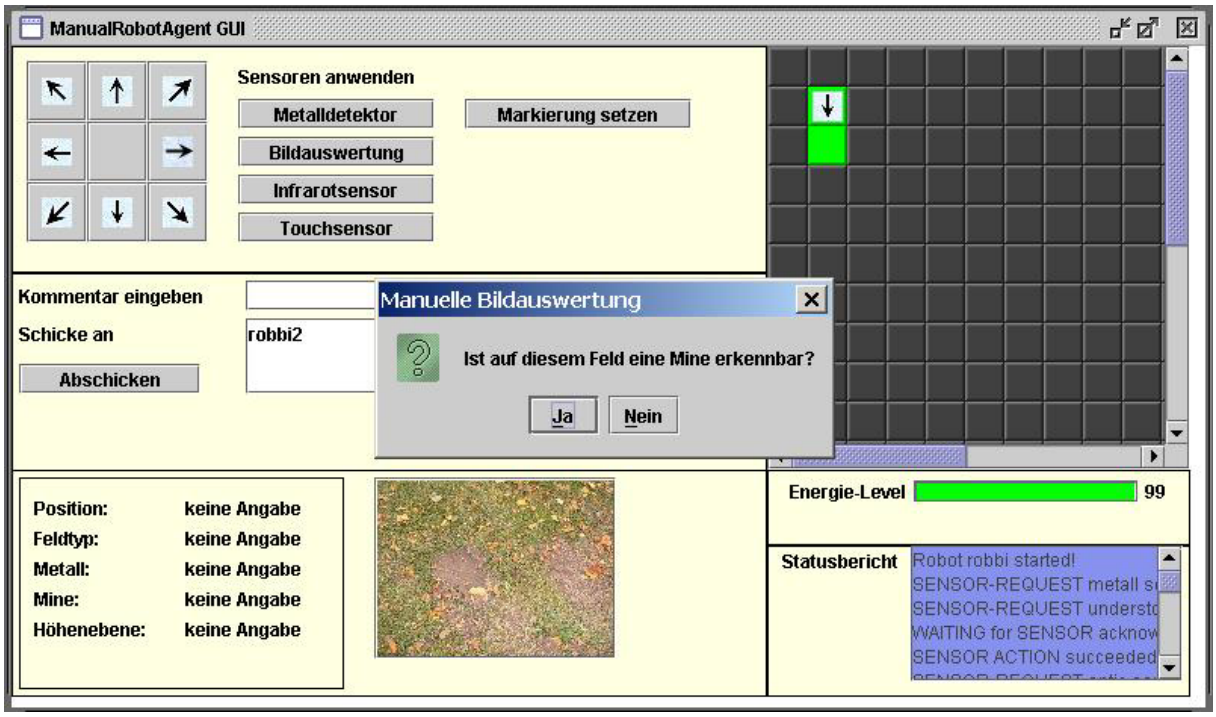
Wählen Sie zum Starten des ManualRobotAgent einen beliebigen Roboternamen und tragen in das Feld Roboterklasse ***`ManualRobotAgent`*** (genaue Schreibweise beachten) ein, und klicken Sie auf die Schaltfläche hinzufügen. Nun erscheint ein GUI, mit dem Sie den Roboter steuern können (siehe Bild). Bevor das ManualRobotAgent-GUI erläutert wird erfolgen noch einige Informationen zum An- und Abmelden. Es können theoretisch beliebig viele Roboter angemeldet werden. Da alle Roboter auf dem Feld mit den Koordinaten (1,1) starten, muss dieses Feld frei sein, wenn ein neuer Roboter registriert wird. Das bedeutet, dass ein zuvor angemeldeter Roboter dieses Feld vorher verlassen muss. Roboter werden auch im Weiteren als Hindernisse betrachtet. Ein Hindernis bedeutet, dass dieses Feld nicht betreten werden kann.



Die einzelnen Bestandteile des GUI werden im Folgenden erläutert.



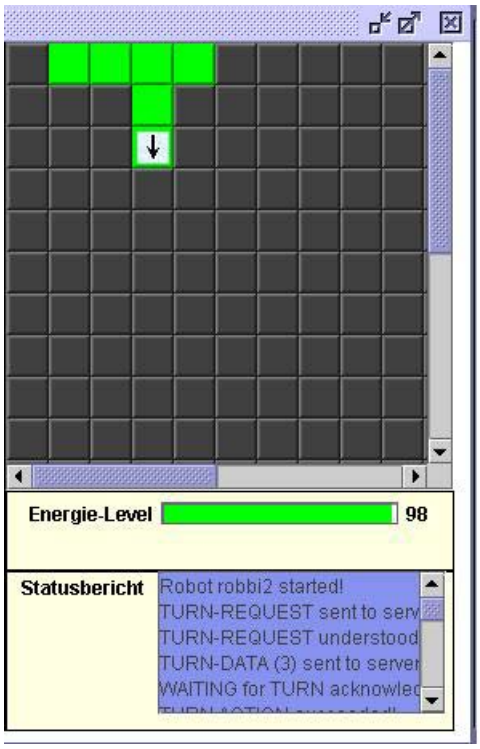
Der Pfeil auf den verdeckten Kacheln ist der neue Roboter, seine Blickrichtung ist gleich der Richtung, in die der Pfeil zeigt. Nun kann das nächste Feldstück (Kachel) untersucht werden; dieses ist genau die Kachel, auf die der Roboter guckt und nicht die, auf die er bereits steht. Auf diese Kachel können nun verschiedene Sensoren angewendet werden. Ein Klick auf den Metalldetektor, lässt das betrachtete Feld entweder grün oder rot erscheinen. Grün bedeutet, dass es keinen Metallausschlag gegeben hat, rot bedeutet, dass Metall im Boden ist. Das heißt aber nicht, dass auch tatsächlich eine Mine an dieser Stelle ist. Ganz im Gegenteil: üblicherweise werden heutzutage Plastikminen verlegt, da diese billiger sind und schwerer aufgefunden werden können. Des Weiteren stehen noch die Bildauswertung und der Infrarotsender zur Verfügung. Ein Klick auf einen dieser Button lässt



jeweils eine Frage mit Bild aufpoppen, auf die dann geantwortet werden muss, ob eine Mine zu sehen ist. Die verschiedenen Pfeilrichtungen auf der linken Seite des GUI, geben auf einen Klick hin die Blickrichtung des Roboters wieder. Es gibt 8 Richtungen; dabei entspricht die Richtung „Nord“ der 1, „Nordost“ der 2 etc. Ein Klick auf einen dieser Pfeile bewirkt eine Änderung des Roboters in diese Richtung. Ein Klick in die Mitte der Pfeile lässt den Roboter auf die vorher betrachtete Kachel gehen.

Wird auf einem Feld eine Mine vermutet, so kann mittels der Schaltfläche ***Markierung setzen*** ein Fähnchen auf die betrachtetet Kachel gesetzt werden.

Betrachten wir die rechte Seite des GUI ein wenig genauer.

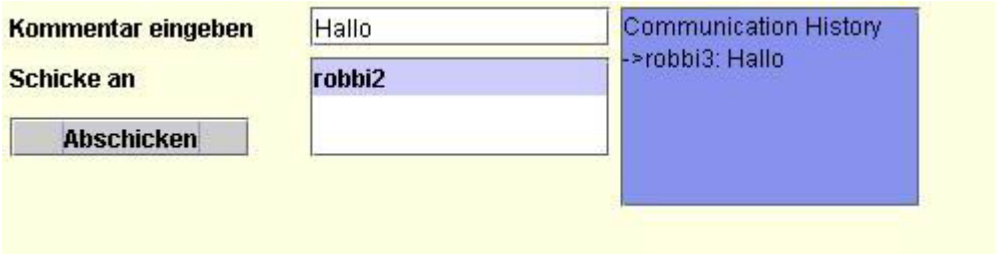


Wie im Bild zu sehen ist, hat der Roboter bereits ein paar Schritte gemacht und einige Felder als begehbar erkannt, ohne dass es einen Sensorausschlag gegeben hat. Dies erkennt man an den grün eingefärbten Kacheln. Hätte ein Sensor angeschlagen, würde die Kachel rot eingefärbt werden, Wenn die Bildaufnahme bzw. der Infrarotsensor eine Mine anzeigen, wird die Kachel mit einem Fähnchen markiert. Die Kacheln werden nur angezeigt, wenn das Spielfeld kleiner 2500 Kacheln groß ist, da es sonst zu Performanceproblemen kommt.

Der Roboter hat nur ein begrenztes Maß an Energie, welches je nach Einsatz von Sensoren, Drehungen oder Bewegungen mehr oder weniger stark sinkt; hier finden die unterschiedlichen Höhenstufen in der Simulationsumgebung ihre Berechtigung; wenn der Roboter eine Steigung meistern muss, wird ihm mehr Energie abgezogen als wenn er keinen Höhenunterschied bewältigen muss. Tritt der Roboter auf eine Mine oder läuft ins Wasser, sinkt das Energielevel sofort auf 0, zusätzlich wird er abgemeldet. Läuft der Roboter so lange bis sein Energielevel auf 0 gefallen ist, ist der Roboter bewegungslos. Allerdings bleibt er registriert.

Im Statusbericht, der unter der Energieleiste zu sehen ist, wird protokolliert, welche Aktionen abgelaufen sind. Hier lassen sich alle Schritte, die der Roboter gemacht hat und dessen Kommunikation mit den anderen Agenten von der Registrierung bis hin zur Abmeldung nachvollziehen.

Roboter (bzw. hier sein Bediener) haben die Möglichkeit mit anderen Robotern zu kommunizieren. Im speziellen manuellen Fall hier kann ein Kommentar im vorgesehenen Feld eingegeben werden. Anschließend müssen alle Adressaten, die die Nachricht erreichen soll, markiert werden und der ***Abschicken***-Button gedrückt werden. Somit erscheint bei allen Adressaten im blauen Feld die gesendete Mitteilung.



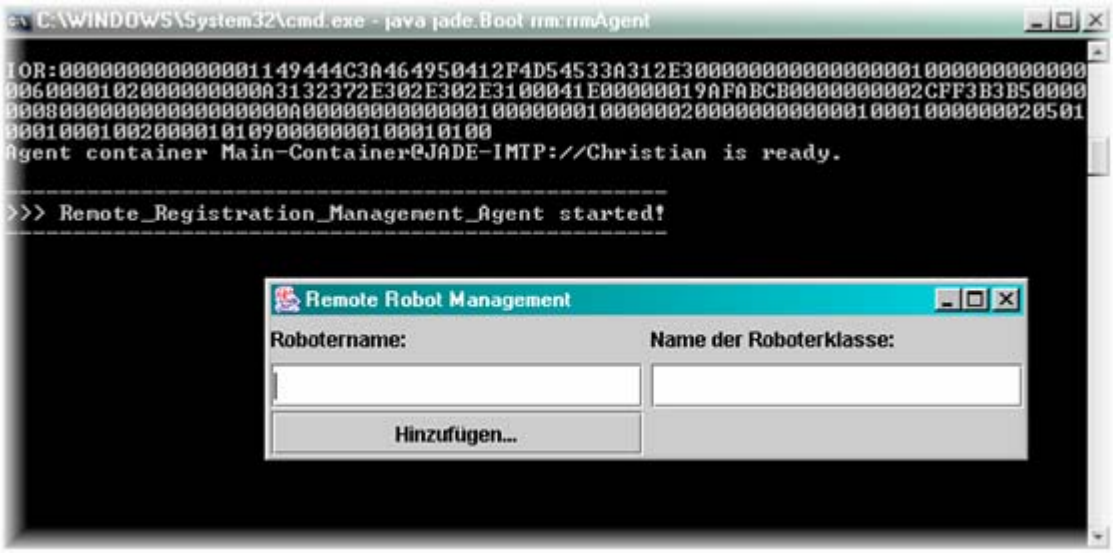
Die Kommunikation ist z.B. sinnvoll, wenn die Roboter ihre Karten synchronisieren wollen.

2.2.2 Verteilter Robotereinsatz

Die Roboter können von mehreren Rechnern aus auf die Simulationsumgebung des Feldagenten zugreifen. Dazu muss auf den beteiligten Rechnern zunächst, wie oben beschrieben, Java, Jade und die Dateien für die Simulationsumgebung installiert werden. Anschließend wird der Roboter über den Remote-Robot-Management-Agent (**`rrmAgent`**) gestartet. Dies geschieht durch den folgenden Befehl (Groß-/Kleinschreibung beachten):

```
java jade.Boot –host Rechnername –container rrm:rrmAgent
```

Es ist zu beachten, dass der host der Rechner ist, auf dem ServerAgent und damit auch der Feldagent und der Simulationsagent laufen.
Es erscheint eine Eingabemaske, in welche der Robotername und die Roboterklasse einzutragen sind.



Ist dies geschehen und kein weiterer Roboter mit dem gleichen Namen beim Serveragenten registriert, wird nach kurzer Zeit das RobotAgent-GUI des gerade angemeldeten Roboters gestartet. Ein weiteres Feldagenten-GUI wird nicht gestartet; dieses läuft nur auf dem host-Rechner.

Der **`rrmAgent`** kann nur von einem anderen Rechner aus aufgerufen werden, da Jade nur einmal gleichzeitig aufgerufen werden kann. Sowohl der host-Rechner als auch der Client benötigen die Jade-Plattform.

2.2.3 Wie programmiere ich meinen eigenen Roboter?

Alle Agenten (Roboteragenten sowie Simulations-, Feld- und Serveragent) erben von der Klasse **`Agent`**, die in einer JADE-Bibliothek (**`jade.core`**) implementiert ist. Roboteragenten weisen allesamt besondere Eigenschaften auf, die in der Klasse **`RobotAgent`** – einer Unterklasse von **`Agent`** – verankert sind. Somit sind alle im System vorhandenen Roboter Unterklassen von **`RobotAgent`**. Um einen eigenen Roboter in das System zu integrieren, muss also die Klasse des neuen Roboters von **`RobotAgent`** erben. Der neue Roboter überschreibt die Methode **`runRobot()`** und erzeugt in dieser Methode eine Referenz auf eine weitere selbst geschriebene Befehlsklasse. Diese Befehlsklasse muss eine Unterklasse von **`Commands`** `sein`, welche alle existierenden Roboterbefehle beinhaltet. Die wichtigsten Methoden sind :

- **`boolean touch()`**, um den touch-Sensor zu bedienen
- **`boolean metall()`**, um den Metallsensor zu bedienen

- **`String optic()`**, um den Fotosensor zu bedienen. Hier ist zu beachten, dass das Bild als base64-kodiertes Stringobjekt zurückgegeben wird. Das Dekodieren dieses Strings muss der Programmierer selbst übernehmen
- **`String infraRed()`**, um den Infrarotsensor zu betätigen. Der Rückgabewert ist ebenfalls ein base64-kodiertes Stringobjekt
- **`void turn(int direction)`, `void turnRight()`, `void turnLeft()`**, um sich zu drehen
- **`void move()`**, um sich fort zu bewegen

In dieser Befehlsklasse (genauer: in der **`run()`**-Methode) hat der Roboterprogrammierer alle Freiheiten, einen Algorithmus für den zu testenden Roboter zu implementieren. Weil JADE nicht threadsicher ist, muss der vom Programmierer erzeugte Algorithmus in einem Thread ablaufen, der in der **`execute()`**-Methode erzeugt wird. Der Thread führt nach seiner Erzeugung automatisch die **`run()`**-Methode aus. Um die Thread-Eigenschaften zu nutzen, muss die selbstgeschriebene Befehlsklasse das Interface **`java.lang.Runnable`** implementieren (siehe Bsp.).

2.2.4 Robbi – ein einfaches Beispiel für die Implementierung eines Roboteragenten

```
public class Robbi extends RobotAgent {

     public void runRobot() {

          Commands cmd = new Operation1(this);
          cmd.execute();

     }
}

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Operation1 extends Commands
     implements Runnable {
     public Operation1(RobotAgent agent) {
          super(agent);
     }
```

```
    public void execute() {
        Thread t = new Thread(this);
        t.start();
    }

    public void run() {
        setWaitCycle(200);
        boolean finish = false;
        int i = 0;
        int blickrichtung = 4;
        int x = 0; int y = 0;
        while (!finish) {
            i++;
            if (! touch() ) {
                move();
                turnRight();
            }
            if (i == 100) {
                finish = true;
            }
        }
}
```

2.2.5 Robbi – ein komplexes Beispiel für die Implementierung eines Roboteragenten

Ein komplexeres Beispiel eines Roboters ist im Quellverzeichnis der Simulationsumgebung hinterlegt. Um die Funktionalität dieses Algorithmus vollständig nutzen zu können, müssen 2 Roboter kurz hintereinander registriert werden. Als Roboternamen müssen **`Hans`** und **`Herse`** gewählt werden, die Roboterklasse ist jeweils **`Robbi`**. Als Spielfeld dient **`metallKarte.map`** (auch im selbigen Verzeichnis abgelegt).

Die Roboter untersuchen die betrachteten Felder zunächst mit dem Touchsensor. Wenn kein Hindernis identifiziert wird, wird der Metallsensor benutzt. Da in dem Spielfeld **`metallKarte.map`** nur Metallminen existieren, spüren die Roboter alle Minen auf.

Ortet ein Roboter eine Mine, so kontaktiert er umgehend seinen Agentenkollegen und teilt ihm die Position der gefundenen Mine mit.

Nach 100 Aktionen ruft jeder Roboter eine selbst erstellte Karte auf. Hindernisse werden gelb, selbst gefundene Minen rot, vom anderen Roboter gefundene Minen blau und nicht untersuchte Felder grau markiert.