UNIVERSITÄT BERN	KURS	TYP	BLATT	AUSGABE	
INFORMATIK	KI	PA	1(1)	HS 10	

PROGRAMMIERAUFGABE 1

Maximale Gruppengrösse: 2 Personen
Abgabe: 27. Oktober 2010
Abzugeben: Listing, Ausdruck eines Ablaufs auf Papier sowie Email des Sourcecodes an Özcan Altin (oezcan.altin@students.unibe.ch).

Teilaufgabe 1

Auf der KI-Übungsseite[1] Sie ein Zip-File (prog1.zip), welches Sie für die Lösung dieser Aufgabe benötigen. Laden Sie das Zip-File von der KI-Seite und entpacken Sie. Im Zip-File finden Sie folgende Dateien:

- graph.jar: dieses Jar-File beinhaltet ein Graphen-Framework. Hauptklasse des Jars ist ein Visualisierungstool, mit dem Graphen angezeigt werden können. In dieser Programmieraufgabe sollen Sie das Jar-File als Bibliothek für eine Graph-Datenstruktur verwenden. (D.h., im Build-Path Ihrer Entwicklungsumgebung einfügen und nicht als Jar "doppelklicken".)
- doc-Verzeichnis: Javadoc zum graph.jar.
- (un-)directed1.gxl, (un-)directed2.gxl: vier Graphen, im GXL-Format gespeichert. Anhand dieser Graphen können Sie die von Ihnen entwickelten Methoden testen.
- ki-Verzeichnis: ein Interface (GraphSearch.java) sowie eine How-To Klasse (GraphHowTo.java), welche zur Lösung der Übungsaufgabe benötigt werden.

Versuchen Sie, das GraphHowTo.java zu kompilieren und arbeiten Sie sich mit dessen Hilfe in das Graph-Package ein (Graphen laden bzw speichern, Modifikation von Graphen, Modifikation von Graphelementen wie Knoten und Kanten, etc.). Studieren Sie dazu den Code von GraphHowTo.java.

Teilaufgabe 2

Implementieren Sie die Graphsuche GRAPHSEARCH wie sie in der Vorlesung behandelt wurde (Algorithmus siehe Folie KI 2/8). Achten Sie hierbei darauf, dass die Implementation auf dem Graph-Package basiert und das (ebenfalls auf der Übungsseite bereitgestellte) Interface GraphSearch implementiert. Insbesondere sollen folgende Punkte gewährleistet sein:

- Eingabeparameter ist nur der zu durchsuchende Graph. Der Graph ist in GXL spezifiziert und kann gerichtet oder ungerichtet sein. Ausserdem sind im Graphen selbst Start- und Zielknoten wie nachfolgend beschrieben markiert.
- Der Startknoten besitzt ein Attribut „isStart", welches für den Startknoten auf „true" und alle übrigen Knoten nicht exisitiert. Analog dazu gibt es für den Zielknoten ein Attribut „isGoal", welches auf „true" gesetzt ist und für alle übrigen Knoten nicht existiert.

[1] `http://www.iam.unibe.ch/fki/lectures/kunstliche-intelligenz` und dann auf Übungsseite

- Die Knoten im Graphen besitzen 2-dimensionale karthesische Koordinaten (x,y). Die Koordinaten sind unter den Schlüsseln „x“ bzw. „y“ als Float abgespeichert. Als Distanz zwischen den Knoten soll dementsprechend die euklidische Distanz verwendet werden.
- Es sollen gerichtete wie ungerichtete Graphen korrekt durchsucht werden können.
- Die zu durchsuchenden Graphen sind im GXL-Format spezifiziert. Einige Beispiele von Graphen im GXL-Format sind auf der KI-Übungsseite zu finden. (Das Graph-Package bietet Ein-/Ausgabefunktionalität für dieses Graphformat.)

Testen Sie GRAPHSEARCH anhand der Graphen auf der Übungsseite.

Teilaufgabe 3

Den oben implementierten Algorithmus setzen wir nun für die Lösung der folgenden Aufgabenstellung ein:

Es ist eine Steuerung für einen kleinen Roboter zu programmieren. Dieser Roboter – nennen wir ihn mal „James“, weil er ungemein fleissig sein soll – ist für den Einsatz im Haushalt gedacht, wo er Aufgaben wie kochen, putzen, waschen, Kaffee bringen usw. erledigen soll. Bei der Erledigung dieser Aufgaben gilt es als durchaus wünschenswert, dass „James“ Hindernisse in der Wohnung wie Tische, Stühle, Schränke usw. meidet, da ansonsten das Mobiliar der stolzen Besitzer des Roboters schon nach wenigen Stunden arg ramponiert wäre.

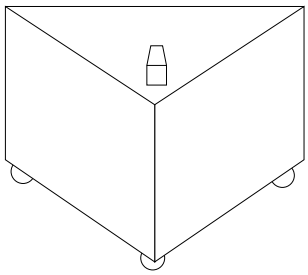

Abbildung 1: „James“

In Abbildung 1 ist „James“ schematisch dargestellt (für uns belanglose Dinge wie Staubsaugergreifer, Geschirrhaltevorrichtung usw. wurden auf diesem Bild weggelassen): Er ist also dreieckig, kann sich auf seinen Rollen in allen Richtungen frei bewegen, wobei aber keine Rotationen möglich sind, und verfügt über eine Kamera, die er schwenken kann, um sich ein Bild seiner Umwelt zu machen. „James“ kennt auch die Räume, in denen er sich bewegen soll (Konfigurationsraum). Weil er weiss, wo Hindernisse stehen, kann er sich gewisse „Sperrgebiete“ berechnen, wo er nicht durchrollen kann oder soll. In Abbildung 2 ist ein entsprechendes Beispiel dargestellt, wobei die Kamera als Referenzpunkt dient. Grau unterlegt ist das „Sperrgebiet“ für das eingezeichnete quadratische Objekt.

Wenn „James“ losrollt, hat er stets ein Ziel vor Augen. Falls sein Ziel von einem bestimmten Standpunkt aus nicht von einem Hindernis verdeckt ist (falls er also sein Ziel gewissermassen „sieht“ mit der Kamera), kann der kleine Roboter einfach losrollen. Andernfalls muss sich „James“ zu einem Punkt bewegen, den er gerade „sehen“ kann, in der Hoffnung, von

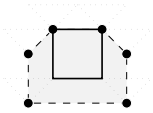

Abbildung 2: Ermittlung von „Sperrgebieten“

dort aus den Zielpunkt zu „sehen“. Um die Anzahl dieser Punkte einzuschränken, legen wir nun fest, dass die möglichen Zielpunkte den Ecken der „Sperrgebiete“ entsprechen. Mit diesen Punkten lässt sich dann ein Sichtbarkeitsgraph aufstellen, in dem festgehalten ist, welche Punkte der Roboter von einem bestimmten Punkt aus anfahren kann. D.h., der Roboter sieht wirklich nur Punkte, die im Sichtbarkeitsgraph nicht verdeckt sind und kann auch nur sie ansteuern.

Kommt nun als letzte Forderung noch hinzu, dass „James“ den kürzesten Weg wählen soll, um an sein Ziel zu gelangen. Seine Besitzer wären wohl kaum erbaut, wenn der Haushaltsroboter stundenlang mit dem Kaffee durch die Wohnung rollen würde, um ihn dann schliesslich kalt abzuliefern. Darum suchen wir mittels A^* den kürzesten Weg zwischen Ausgangs- und Zielpunkt.

Überprüfe die Korrektheit der Steuerung an folgendem Beispiel: In Abbildung 3 ist das Esszimmer von „James“ Arbeitsraum dargestellt. Seine Aufgabe ist es, den Kaffee von der Küche, wo er sich momentan befindet, in das Wohnzimmer zu bringen.

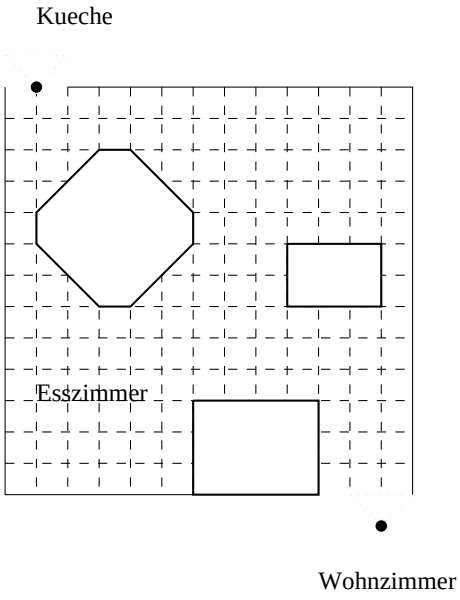

Abbildung 3: Plan des Esszimmers

Wichtig: Da es in dieser Aufgabe primär um den A^*-Algorithmus geht, muss „James“ sich den Sichtbarkeitsgraphen nicht explizite berechnen. Man kann davon ausgehen, dass eine entsprechende Prozedur existiert und kann deshalb vereinfachend den Sichtbarkeitsgraphen manuell/„hard-coded“ eingeben.

Das Programm soll die einzelnen Schritte von „James“ grafisch darstellen. Abzugeben sind das Listing des Programms, der Ausdruck eines Programmlaufes sowie die gesamte Source an die Betreuer (email).