Berner Fachhochschule

Technik und Informatik

Projektaufgabe Shaky

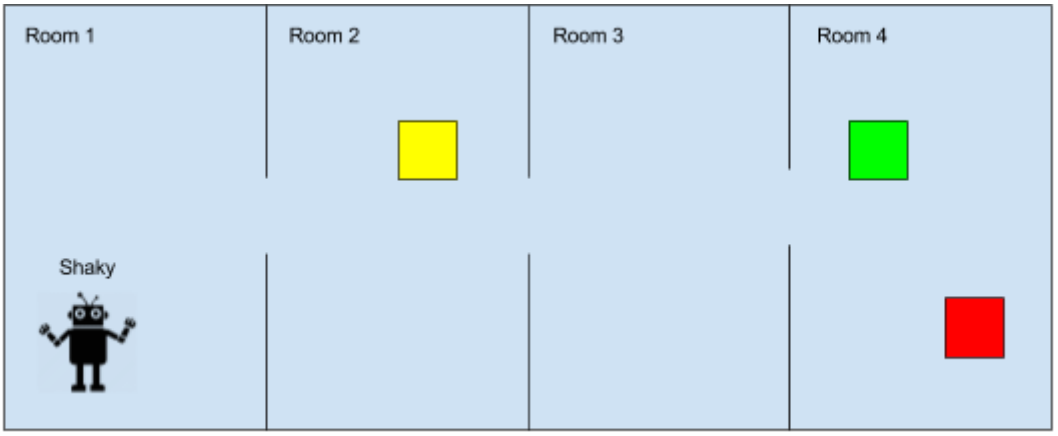
Autoren:	Raphael Laubscher
Institution:	Berner Fachhochschule - Technik und Informatik
Auftraggeber:	Jürgen Eckerle, Dozent für KI, Berner Fachhochschule

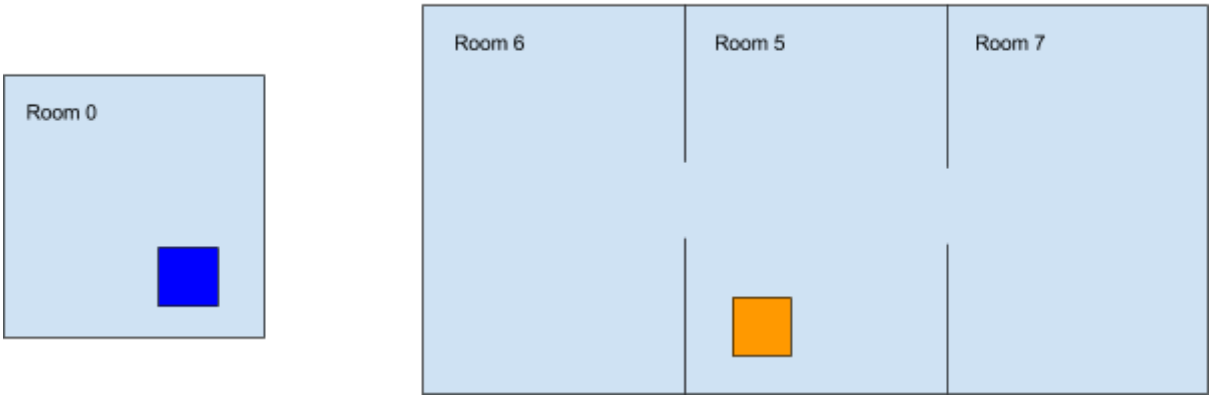
Bern, Juni 2017

Problem Modellierung

Ziele

- Shaky soll alle verbundenen Räume erreichen können.
- Shaky kann eine Box holen, resp. hochheben.
- Shaky kann eine Box in einen anderen Raum bewegen.





Um die drei Ziele zu erreichen wurde folgende Modellierung in Prolog vorgenommen.

Fakten

Prolog	**Beschreibung**
`connected(room1, room2).`	Mit dem Prädikat `connected(X, Y)` wird definiert, dass zwei Räume verbunden sind. Eine Konsequenz dieser Modellierung ist es, dass die Räume nur in der Richtung von room1 nach room2

Bern, Juni 2017

	begehbar sind. Um die Räume auch in der anderen Richtung zu verbinden müsste man einen weiteren Fakt connected(room2, room1) definieren oder die path-Regel weiter unten anpassen. Das führt aber zu Loops, wenn die weiter unten beschriebenen Regeln angewandt werden sollen. Der Agent wechselt sozusagen zwischen Raum 1 und Raum 2 hin und her und bleibt in einen Loop stecken.
`inRoom(shaky, room1).` `inRoom(redBox, room4).`	`inRoom` beschreibt welches Objekt (also entweder eine Box oder ein Agent) sich in welchem Raum befindet.

Das sind bereits alle Fakten die notwendig sind um den Zustand der oben skizzierten Welt zu beschreiben.

Regeln

Prolog	**Beschreibung**

`path(X, Y):-`
`  connected(X, Y).`
 	
`path(X, Y):-`
`  connected(X, Z),`
`  path(Z, Y).`
	Die Regel `path` gibt ist erfüllt, falls es einen Weg von `X` nach `Y` gibt. Die Regel muss rekursiv definiert sein, damit auch Räume, die nicht direkt verbunden sind, zu erreichen sind.
	Die obere Regel bildet den Basecase der Rekursion. Die untere Regel ist rekursiv definiert, wobei wichtig ist, dass der rekursive Aufruf `path(Z, Y)` am Schluss der Regel steht. Dies verhindert Loops.

`goTo(Agent, RoomY):-`
`  inRoom(Agent, RoomX),`
`  path(RoomX, RoomY).`
	Die `goTo` Regel ist erfüllt falls ein Agent von Raum X nach Raum Y gelangen kann. Die Regel basiert auf der path Regel mit dem Zusatz, dass noch geprüft wird, ob sich der Agent überhaupt in einem der Verbundenen Räume aufhält.

`get(Box, Agent):-`
`  inRoom(Box, RoomX),`
`  goTo(Agent, RoomX).`
	Mit `get` kann geprüft werden, ob ein Agent eine Box holen kann. Dies ist erfüllt falls der Agent in den Raum gehen kann, in der sich die Box befindet. Diese Regel basiert auf der `goTo` Regel.

`put(Box, Agent, RoomZ):-`
`  get(Box, Agent),`
`  goTo(Agent, RoomZ).`
	Die `put` Regel ist erfüllt, wenn ein Agent eine Box in den gewünschten Raum verschieben kann. Dafür machen wir uns die `get` und die `goTo` Regel zu nutze.

Bern, Juni 2017

Modellierung mit Strips

<table>
<tr><td>World State</td><td>Wie erwähnt ist es für diese Problemdefinition ausreichend die Welt mit connect und inRoom Fakten zu beschreiben. Der World State setzt sich also nur aus diesen Fakten zusammen.

<pre>
WorldState = [
  connected(room1, room2),
  connected(room2, room3),
  connected(room3, room4),
  connected(room5, room6),
  connected(room5, room7),
  inRoom(shaky, room1),
  inRoom(yellowBox, room2),
  inRoom(greenBox, room4),
  inRoom(redBox, room4),
  inRoom(orangeBox, room5),
  inRoom(blueBox, room0)
]
</pre>
Wenn der Agent den Raum wechselt oder eine Box verschiebt, ändern sich nur die inRoom Fakten. Entweder ist der Agent in einem anderen Raum und/oder eine der Boxen. Die connected Fakten sind fix und beschreiben die Räume, die sich nicht ändern.</td></tr>
<tr><td>path(X, Y)
(ohne Türen)</td><td>Die Path ohne Türen ändert am WorldState nichts. Die alte Regel kann sozusagen beibehalten werden. Nur die Preconditions müssen geprüft werden, die Add- und Delete-List bleiben leer.

<table>
<tr><td>PC</td><td>path(X, Y) (siehe Definition Path Regel oben)</td></tr>
<tr><td>ADD</td><td>-</td></tr>
<tr><td>DELETE</td><td>-</td></tr>
</table>
</td></tr>
<tr><td>path(X, Y)
(Überlegungen mit Türen)</td><td>In einer Welt mit Türen könnte man die Türen öffnen und schliessen. Die connected(x, y) Fakten kann man beibehalten, sie würde bedeuten zwei Räume sind durch eine Türe verbunden. Weiter müsste man die Fakten open(x, y) und closed(x, y) einführen. Open und closed sind nötig um zu prüfen ob die Türen offen oder geschlossen sind und um</td></tr>
</table>

<table>
<tr>
<td></td>
<td>diese zu öffnen oder zu schliessen.

Beispiel:

<pre>WorldState = [
  connected(room1, room2),
  connected(room2, room3),
  connected(room3, room4),
  closed(room1, room2),
  open(room2, room3),
  closed(room3, room4)
]</pre>
<table>
<tr><td>PC</td><td><code>path(X, Y)</code> (siehe Definition Path Regel oben)</td></tr>
<tr><td>ADD</td><td><code>open(X, Y)</code> (resp. auch alle dazwischen liegenden Türen)</td></tr>
<tr><td>DELETE</td><td><code>closed(X, Y)</code> (dito.)</td></tr>
</table>
</td>
</tr>
<tr>
<td>goTo(Agent, RoomY)</td>
<td>Agent wechselt in den Raum Y.

<table>
<tr><td>PC</td><td><code>inRoom(Agent, RoomX),
path(RoomX, RoomY).</code></td></tr>
<tr><td>ADD</td><td><code>inRoom(Agent, RoomY)</code></td></tr>
<tr><td>DELETE</td><td><code>inRoom(Agent, RoomX)</code></td></tr>
</table>
</td>
</tr>
<tr>
<td>get(Box, Agent)</td>
<td>Agent holt eine Box.

<table>
<tr><td>PC</td><td><code>inRoom(Box, RoomX),
inRoom(Agent, RoomY),
path(RoomY, RoomX).</code></td></tr>
<tr><td>ADD</td><td><code>inRoom(Agent, RoomX)</code></td></tr>
<tr><td>DELETE</td><td><code>inRoom(Agent, RoomY)</code></td></tr>
</table>
</td>
</tr>
<tr>
<td>put(Box, Agent, RoomZ)</td>
<td>Agent verschiebt eine Box in anderen Raum.</td>
</tr>
</table>

Bern, Juni 2017

PC	`inRoom(Box, RoomX),`
	`inRoom(Agent, RoomY),`
	`path(RoomY, RoomX),`
	`path(RoomX, RoomZ).`

ADD	`inRoom(Agent, RoomZ),` `inRoom(Box, RoomZ)`
DELETE	`inRoom(Box, RoomX),` `inRoom(Agent, RoomY)`