Methoden

Methode	Ausgeführte Aktion	mögliche Ablauffehler
Schritt()	macht einen Schritt in die Blickrichtung	steht vor Wand / Quader / Roboter; kann nicht so hoch springen;
LinksDrehen()	dreht sich nach links (um 90°)	
RechtsDrehen()	dreht sich nach rechts (um 90°)	
Hinlegen()	legt vor sich einen Ziegel hin	steht vor Wand / Quader / Roboter; maximale Stapelhöhe erreicht;
Aufheben()	hebt einen Ziegel auf, der vor ihm liegt	steht vor Wand / Quader / Roboter; kein Ziegel vor dem Roboter;
MarkeSetzen()	setzt an seiner Position eine Marke	
MarkeLoeschen()	löscht an seiner Position eine Marke	
QuaderAufstellen()	stellt vor sich einen Quader auf	steht vor Wand / Quader / Roboter; vor ihm liegen schon Ziegel; vor ihm ist eine Marke;
QuaderEntfernen()	entfernt einen Quader, der vor ihm steht	steht vor Wand / Roboter; vor ihm ist kein Quader;
Warten(float dauer)	wartet „dauer“-viele Sekunden	
TonErzeugen()	gibt einen Ton von sich	
MeldungAusgeben (String was)	übergibt eine Meldung an das Weltfenster, das diese im Textfeld darstellt	

Logische Abfragefunktionen

Negationen durch den üblichen `Not-Operator`, z.B. `!karol.IstWand()`

Abfragefunktion	der Roboter meldet **`true`**,
IstWand()	wenn er vor der Wand oder vor einem Quader steht und in diese Richtung schaut
IstZiegel()	wenn er vor einem Ziegel oder Ziegelstapel steht und zu diesem schaut
IstMarke()	wenn er auf einer Marke steht
IstRoboter()	wenn er vor einem anderen Roboter steht und zu diesem schaut
IstRoboterInSicht()	wenn in seiner Blickrichtung ein anderer Roboter steht, egal wie weit weg. Ziegel und Quader behindern die Sicht.
IstBlickSueden(), **IstBlickNorden(),** **IstBlickWesten(),** **IstBlickOsten()**	wenn er in diese Richtung schaut

Kontrollstrukturen

Name	Notation in RobotKarol	Notation in Java

einseitige bedingte Anweisung	**wenn** *Bedingung* **dann**
	  Anweisung
	  …
	***wenn**
		if (*Bedingung***)**
		{
		  Anweisung;
		  …
		}

zweiseitige bedingte Anweisung	**wenn Bedingung dann**
	  Anweisung
	  …
	sonst
	  Anweisung
	  …
	***wenn**
		if (*Bedingung***)**
		{
		  Anweisung;
		  …
		}
		else
		{
		  Anweisung;
		  …
		}

Wiederholung mit fester Anzahl	**wiederhole** *Anzahl* **mal**
	  Anweisung
	  …
	***wiederhole**
		for (int i=0; i<*Anzahl***; i++)**
		{
		  Anweisung;
		  …
		}

Wiederholung mit Anfangsbedingung	**wiederhole solange**
	  Bedingung
	  Anweisung
	  …
	***wiederhole**
		while (*Bedingung***)**
		{
		  Anweisung;
		  …
		}

Wiederholung mit Endbedingung	**wiederhole**
	  Anweisung
	  …
	***wiederhole solange**
	  Bedingung
		do
		{
		  Anweisung;
		  …
		}
		while (*Bedingung***)**