Rupprecht-Gymnasium München
Studienseminar 2019/2021
Fachseminar Mathematik/Informatik

Robot Karol blockbasiert - ein Experiment im Informatikunterricht der 7. Klasse

Schriftliche Hausarbeit zur zweiten Staatsprüfung für das Lehramt an Gymnasien im Fach Informatik

Vorgelegt von: StRef Franziska Rieger
Datum: 05. August 2020
Seminarlehrer: StD Klaus Reinold

1 Vorwort

- Robot Karol ist ein weitverbreitetes Tool, indem textuell programmiert wird
- V.a. in bayerischen Gymnasien verbreitet
- Übergang in ein graphisches Tool fällt vor allem nicht-Informatiklehrern schwer und sie scheuen davor zurück
- nicht nur die Lehrkraft, sondern auch die Schüler müssen sich in ein neues Tool mit neuen Beispielen einarbeiten
- Übergang soll mit der Integration von Karol in Snap! erleichtert werden

2 Snap! und Robot Karol

2.1 Snap!

- Snap! ist eine Programmieroberfläche, bei der durch Drag and Drop programmiert wird.
- Jedes Objekt wird in einem separaten Editor programmiert
- Oberfläche von Snap! vorstellen (Übersicht über die Fenster und Aufgaben der einzelnen Fenster)

2.2 Robot Karol

- Beim Programm Robot Karol werden Objekte der Klassen ROBOTER, ZIEGEL, MARKE und QUADER in einer Welt angezeigt.
- Ziegel, Marken und Quader werden vom Roboter erzeugt und in der Welt gesetzt.
- Ziegel und Quader direkt vor den Roboter, Marken im Gegensatz dazu direkt auf den Feld von Karol
- Quader können nicht entfernt werden
- maximale Sprunghöhe und Rucksack sind optionale Werte, die angepasst werden können.

- Rucksack enthält Ziegel. Die Rucksackgröße sowie die Anzahl der Ziegel im Rucksack können abgefragt werden

3 Integration von Robot Karol in Snap!

Um die Vorteile der visuellen Programmierung auch mit dem beliebten Beispiel *Robot Karol* nutzen zu können, wird die Umgebung und das Konzept des Roboters in *Snap!* umgesetzt. Dies führt zu einem Umbau der Klassenstruktur von *Robot Karol* und dem Hinzufügen von Variablen und Methoden. Diese sind nicht alle für den Benutzer sichtbar, sodass eine gewohnte Programmierung des Objektes *ROBOTER* mit Anweisungen und Bedingungen, die analog zu den Methoden in *Robot Karol* benannt sind, möglich ist. Im Folgenden wird das allgemeine Konzept der Umsetzung, sowie alle hinzugefügten Variablen, Methoden und Bedingungen beschrieben.

3.1 Allgemeines Konzept

Karol wird mit Hilfe von vier Objekten in *Snap!* integriert. Die Objekte *Feld*, *Rand1* und *Rand2* stellen den Hintergrund bzw. die Welt des Karol dar und werden nur zu Beginn des Programmes einmal erzeugt und verschoben. Nach jedem Verschieben wird ein Abdruck des entsprechenden Objektes hinterlassen, sodass dadurch die Welt aufgebaut wird.
Zu Beginn des Programmes werden Klone vom Objekt *Feldelement* erzeugt, die jeweils ein Feldelement repräsentieren. Dieser Klon kann sein Aussehen bzw. sein Kostüm ändern und steht, je nach Kostüm, für den Roboter, einen Ziegel oder nichts und ist unsichtbar. Alle geklonte Feldelemente werden in einer Liste verwaltet und können anhand der x-, y-, und z-Koordinate eindeutig identifiziert werden. Die Koordinate des linken oberen Feldes ist (1/1). Nach rechts bzw. unten verlaufen die positiven x- bzw. y-Achsen.
Das einzige Objekt, das durch den Bediener programmiert wird, ist das Objekt *Roboter*. Dieses wird allerdings nicht über die Welt bewegt, sondern es sendet an den entsprechende Klon des Objektes *Feldelement* die Information, dass dieses aktuell den Roboter repräsentiert und das Aussehen bzw. Kostüm entsprechend ändern muss. Das Objekt *Roboter* ist selbst nie auf der Welt sichtbar.

3.2 Variablen

Variablen, die einmalig zu Beginn des Programmes gesetzt werden sind *Länge*, *Breite* und *Höhe*. Diese stehen für die Größe der Welt und können im Programmverlauf nicht geändert werden. Eine weitere Variable, die beim Abspielen eines Algorithmus nicht geändert wird, ist *Sprunghöhe*. Diese Variable hat in *Snap!* die gleiche Funktion wie in *Robot Karol*.
Zu Verwaltung der Klone vom Objekt *Feldelement* wird die Variable *Matrix* verwendet. Dies ist eine Liste mit *(Länge x Breite)* - vielen Einträgen. Jeder Eintrag steht für ein Feld auf der Welt (siehe Abbildung 1).

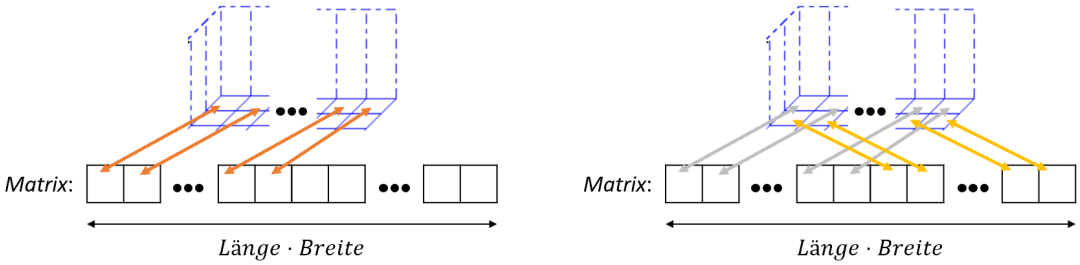


Abbildung 1: Aufbau der Variable *Matrix*

Da sich auf jedem Feld *Höhe* - viele Feldelemente befinden wird jedes Feld in der Matrix durch eine Liste der Länge *Höhe* repräsentiert. Diese Liste enthält die Referenzen auf die jeweiligen Feldelemente, die sich auf dem Feld befinden (siehe Abbildung 2).
Variablen, die während dem Durchlaufen des Programmes auf Grund der Bewegung des Roboters geändert werden sind *x-Koordinate*, *y-Koordinate*, *z-Koordinate* und *Blickrichtung*. Diese stehen für die aktuelle Position und die Blickrichtung des Karols. Anhand dieser Parameter wird der Klon vom Objekt *Feldelement* ermittelt, der anhand der Blickrichtung das passende Karol Kostüm wählt.

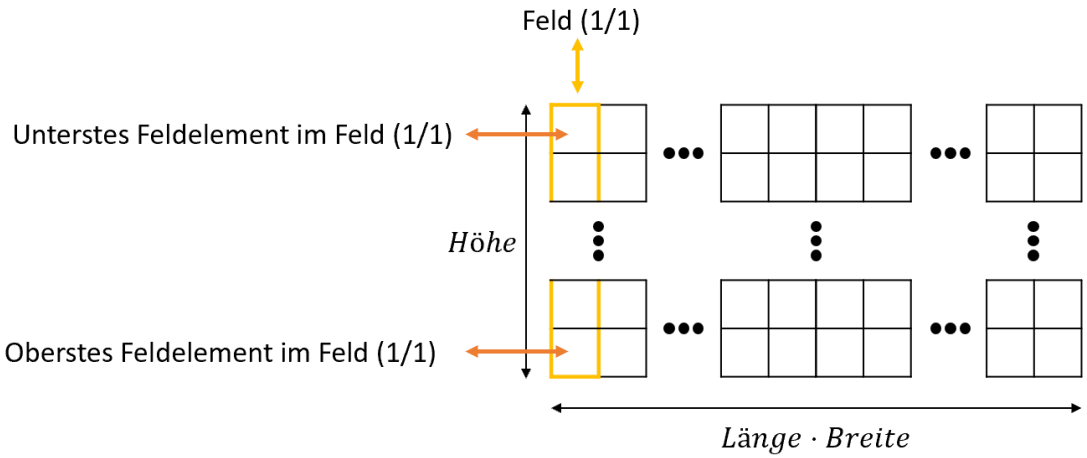


Abbildung 2: Aufbau der Variable *Matrix* mit enthaltenen Listen

3.3 Methoden zum Erzeugen der Welt

Es gibt zwei Methoden bzw. Blöcke, mit denen in *Snap!* eine Karol Welt erzeugt werden kann. Diese unterscheiden sich nur in der Anzahl der Eingabeparameter. Bei beiden Methoden müssen die Länge und Breite der neuen Welt übergeben werden. Bei einer Methode wird zusätzlich die Höhe der Welt, die Startposition und die Sprunghöhe des Karols sowie dessen Blickrichtung festgelegt. Werden diese nicht gesetzt, so ist die Höhe der Welt fünf, die Startposition (1/1), die Sprunghöhe eins und die Blickrichtung ist Süden. Die unterschiedliche Anzahl der Eingabeparameter wird auch in den Methodenköpfe in Abbildung 3 deutlich.

Abbildung 3: Methodenköpfe zum Erzeugen einer Welt

Innerhalb der Methoden werden die Variablen entsprechend der Übergabewerte gesetzt, die Matrix und die Feldelemente erzeugt sowie die Welt gezeichnet. Je nach der Position des Karols wird das Kostüm des entsprechenden Feldelementes geändert. Die Rümpfe der beiden Methoden befinden sich in Anhang **??**.

Um die Welt zu zeichnen wird der Block *Male Welt* aufgerufen (siehe Anhang **??**). Dieser verschiebt die Objekte *Feld*, *Rand1* und *Rand2* entsprechend der Länge, Breite und Höhe der Welt und erzeugt nach jedem Verschieben einen Abdruck des entsprechenden Objektes.
Des Weiteren wird zu Beginn des Programmes die Matrix initialisiert und deklariert. Wie bereits beschrieben besteht die Matrix aus *(Länge x Breite)* - vielen Einträgen, die jeweils eine Liste der Länge *Höhe* enthalten. Die Klone des Objektes *Feldelement* werden erzeugt, entsprechend der Parameter verschoben, in der Liste abgespeichert und anschließend unsichtbar gemacht. Diese Aktionen werden in der Methode *New Matrix* ausgeführt (siehe Anhang **??**).
Ein optionaler Block, der zu Beginn des Programmes ausgeführt werden kann, ist der Block *Setze Ziegel*. Mit Hilfe dieser Methode kann die Welt geändert und zu Beginn darin Ziegel erzeugt werden. Der Block hat als Eingabeparameter die Koordinate des Feldes, auf dem der Ziegel erzeugt werden soll, sowie die Farbe und die Anzahl der Ziegel auf diesem Feld. Wird die Methode ausgeführt, so wird die entsprechende Liste in der Matrix ausgewählt und in dieser das unterste, nicht sichtbare Feldelement bestimmt. Das Kostüm dieses Elements wird dann auf die entsprechende Farbe des Ziegels geändert. Die genaue Funktionsweise dieser Methode ist in Anhang **??** dargestellt. Ein Beispiel für eine erzeugte Welt, die Ziegel enthält befindet sich mit den entsprechenden Matrixeinträge in Anhang **??**.

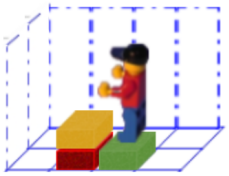
3.4 Methoden von Robot Karol in Snap!

Die wichtigsten Methoden, mit denen der Roboter in der Umgebung *Robot Karol* programmiert wird, sind auch in *Snap!* umgesetzt und tragen den selben Namen.
Der Block *Schritt* bewegt den Karol ein Feld in die aktuelle Blickrichtung nach vorne. Hierzu prüft das Programm zunächst, ob die Höhendifferenz im Rahmen liegt und in dieser Richtung überhaupt ein Feld vorhanden ist. Das Feldelemente, dessen Kostüm aktuell den Roboter darstellt, muss nach dem Ausführen des Schrittes wieder unsichtbar werden. Daher wird dieses Element zwischengespeichert und dessen Kostüm am Ende des Schrittes geändert. Je nach Blickrichtung werden die Variablen *x-Koordinate*, *y-Koordinate* und *z-Koordinate* angepasst. Anschließend ändert das Feldelement, das nach dem Ausführen des Schrittes für Karol steht das Kostüm entsprechend. Der Code des Blockes *Schritt* ist in Anhang **??**. Ein Beispiel für die Belegung der Einträge der Variable *Matrix* vor und nach dem Ausführen der Methode *Schritt* ist in Abbildung 4 dargestellt.

Feld (1/1) (2/1) (3/1) (4/1) (2/1) (2/2) (2/3) (2/4)

Matrix

unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Rot**	**Grün**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Gelb**	**Karol**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar



Feld (1/1) (2/1) (3/1) (4/1) (2/1) (2/2) (2/3) (2/4)

Matrix

unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Rot**	**Grün**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Gelb**	unsichtbar	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Karol**	unsichtbar	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar

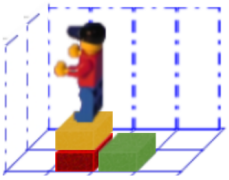


Abbildung 4: Beispiel für die Einträge in der Variable *Matrix* vor (oben) und nach (unten) dem Ausführen der Methode *Schritt*

Außer mit der Methode *Schritt* kann Karol auch mit den Blöcken *RechtsDrehen* und *LinksDrehen* bewegt werden. Abhängig von der aktuelle Blickrichtung wird die Variable *Blickrichtung* neu gesetzt und das Kostüm des aktuellen Feldelementes verändert. Der Code zu den Blöcken ist in Anhang **??** dargestellt.

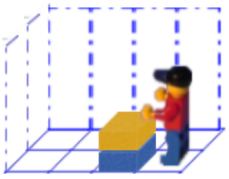
Mit den Blöcken *Hinlegen* und *Aufheben* wird ein Ziegel hingelegt bzw. aufgehoben. Die Methode *Hinlegen* hat als Übergabewert die Farbe des Ziegels. Diese kann rot, grün, blau oder gelb sein. Beim Ausführen dieses Blocks wird zunächst überprüft, ob das Feld vor Karol existiert und die Liste, die in der Matrix für dieses Feld steht, ausgelesen. In dieser Liste muss das unterste Feldelement, dessen Kostüm nicht durchsichtig ist, zum Ziegel werden (siehe Abbildung 5). Die Farbe des Ziegels bzw. des Kostüms wird entsprechend der übergebenen Farbe geändert. Die Methode *Hinlegen* ist in Anhang **??** dargestellt.

Zum Aufheben eines Ziegels wird die Methode *Aufheben* ausgeführt. Das Programm prüft zunächst, ob das Feld vor Karol existiert und dieses mindestens einen Ziegel enthält. Da ein Ziegel entfernt werden soll, wird das Kostüm von diesem Ziegel auf unsichtbar geändert. Das Skript von diesem Block ist im Anhang **??** dargestellt und ein Beispiel für die Belegung der Matrix in Anhang **??**.

Feld (1/1) (2/1) (3/1) (4/1) (2/1) (2/2) (2/3) (2/4)

Matrix

unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Blau**	**Karol**
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Gelb**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar



Feld (1/1) (2/1) (3/1) (4/1) (2/1) (2/2) (2/3) (2/4)

Matrix

unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Blau**	**Karol**
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Gelb**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	**Grün**	unsichtbar
unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar	unsichtbar

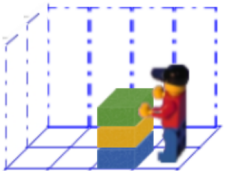


Abbildung 5: Beispiel für die Einträge in der Variable *Matrix* vor (oben) und nach (unten) dem Ausführen der Methode *Hinlegen*

3.5 Hilfsmethoden

Es gibt einige Methoden, die vom Benutzer nicht aktiv verwendet werden müssen. Diese Methoden haben den Zweck, den Code innerhalb der implementierten Blöcke übersichtlicher und leichter nachvollziehbar zu machen. Damit ist es einem geübten Programmierer möglich, das Konzept und die Arbeitsweise der Blöcke zu verstehen und das Projekt ggf. zu erweitern.
Ein Block, der vor jeder Bewegung und jedem Hinlegen oder Aufheben von einem Ziegel ausgeführt wird, ist *Feld frei* (siehe Anhang **??**). Diese Bedingung liefert die Information, ob das Feld vor Karol überhaupt existiert und die entsprechende Operation ausgeführt werden kann. Der Rückgabewert ist abhängig von der aktuellen Position des Roboters und dessen Blickrichtung.
Eine weitere Methode, die vor dem Ausführen eines Schrittes aufgerufen wird, ist der Block *Höhendifferenz im Rahmen*. Hierbei handelt es sich um eine Bedingung, die im Anhang **??** dargestellt ist. Der boolesche Rückgabewert sagt aus, ob die Höhendifferenz zwischen dem aktuellen Feld und dem Feld vor Karol innerhalb der Sprunghöhe liegt. Ist dies der Fall, so liefert die Bedingung den Wert *true* und Karol kann den Schritt machen.
Um die Höhendifferenz zu ermitteln muss das Programm die z-Koordinate des Karols und die Anzahl der Ziegel vom Feld vor Karol auslesen. Dies geschieht mit der Methode *getAnzahlZie-*

gel. Der Zusammenhang zwischen den Methoden wird auch im Sequenzdiagramm der Methode *Schritt* in Abbildung 6 deutlich.
Die Methode *getAnzahlZiegel* erwartet als Übergabewert eine Zahl. Sie liest die Liste, die sich in der Matrix an der Stelle dieser Zahl befindet, aus und zählt, wie viele Elemente in dieser Liste nicht unsichtbar bzw. nicht Karol sind (siehe Anhang **??**). Dieser Wert stimmt dann mit der Anzahl der Ziegel, die sich auf diesem Feld befinden, überein und wird ausgegeben.
Zwei Hilfsmethoden, die im Block *getPositionMatrix* genutzt werden, sind die Methoden *getX-Feld* und *getYFeld* (siehe Anhang **??**). Diese Methoden haben als Rückgabewert die x- bzw. y-Koordinate des Feldes, welches sich vor Karol befindet. Die Werte sind abhängig von der aktuellen Position des Roboters und dessen Blickrichtung. Anhand dieser Werte kann die Position der Liste, die dieses Feld innerhalb der Variable *Matrix* repräsentiert, berechnet werden.

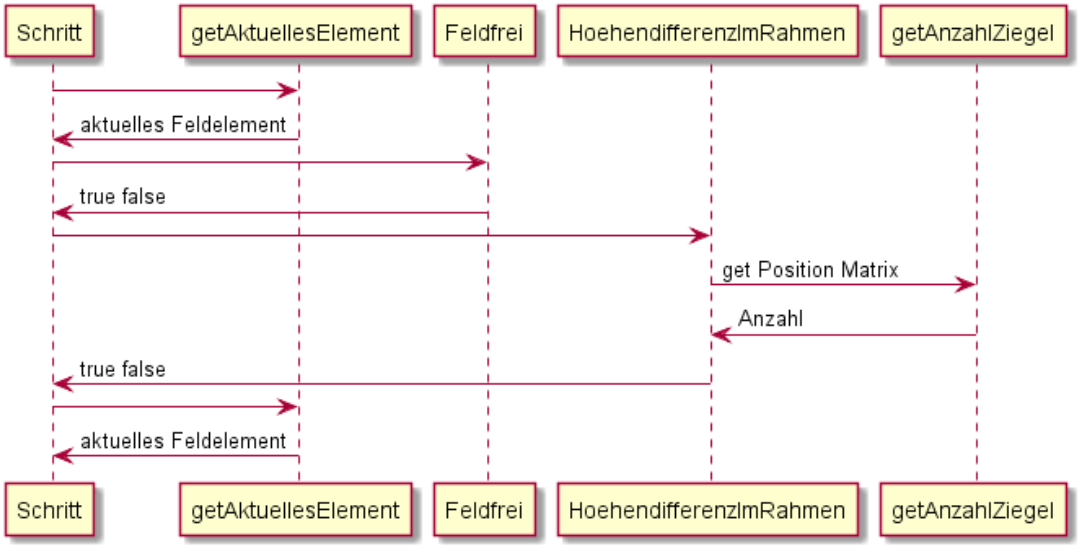


Abbildung 6: Sequenzdiagramm der Methode *Schritt*

Im Sequenzdiagramm der Methode *Schritt* in Abbildung 6 ist dargestellt, dass die Methode *getAnzahlZiegel* als Übergabewert den Rückgabewert der Methode *getPositionMatrix* hat. Die Methode berechnet die Position der Liste, die innerhalb der Variable *Matrix* für das Feld steht, das sich vor Karol befindet. Eine genaue Darstellung von diesem Block befindet sich im Anhang **??**. Der Index wird nicht beim Ausführen eines Schrittes, sondern auch beim Hinlegen und Aufheben von Ziegeln benötigt (siehe Sequenzdiagramm der Methode *Aufheben* in Abbildung 7).

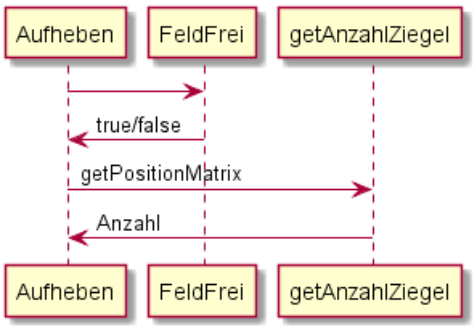


Abbildung 7: Sequenzdiagramm der Methode *Aufheben*

Nach dem Ausführen einer Drehung oder eines Schrittes muss das Aussehen bzw. das Kostüm des Feldelementes, welches aktuell für Karol steht, verändert werden. Das Kostüm ist von der aktuellen Blickrichtung abhängig. Zum Aktualisieren des Kostüms wird die Methode *Aktualisiere Kostüm* verwendet (siehe Anhang **??**).
Um das Kostüm des aktuellen Elementes zu verändern, muss das entsprechende Feldelement von der passenden Liste in der Variable *Matrix* ausgelesen werden. Dies geschieht mit der Methode *get Aktuelles Element* (siehe Anhang **??**). Die Position der Liste wird anhand der Variablen *x-Koordinate* und *y-Koordinate* berechnet. Die *z-Koordinate* gibt Auskunft über die Position des Feldelementes innerhalb der Liste.

3.6 Bedingungen

Die Bedingungen *IstZiegel*, *NichtIstZiegel*, *IstWand*, *NichtIstWand*, *IstNorden*, *IstSüden*, *IstWesten* und *IstOsten* wurden analog zu den Bedingungen in *Robot Karol* umgesetzt.
Bei der Bedingung *IstZiegel* wird mit der Methode *getAnzahlZiegel* die Anzahl der Ziegel auf dem Feld vor Karol ermittelt. Liegt diese bei 0, so ist dort kein Ziegel und die Methode liefert den Wert *false*. Die Bedingung *NichtIstZiegel* funktioniert invers zu *IstZiegel* und greift auf diese zurück (siehe Anhang **??**).
Der Block *IstWand* verwendet die Methode *Feld frei* und testet, ob das Feld vor Karol existiert. Ist dies der Fall, so liefert die Bedingung *IstWand true* zurück. Analog zu *IstZiegel* und *NichtIstZiegel* funktioniert die Bedingung *NichtIstWand* inverse zu *IstWand* (siehe Anhang **??**).
Die Rückgabewerte der Methoden *IstNorden*, *IstSüden*, *IstWesten* und *IstOsten* sind nur von der Variabel *Blickrichtung* abhängig. Die genaue Funktionsweise der Bedingungen sind in Anhang **??** dargestellt.

4 Erweiterungsmöglichkeiten

Das Programm bietet noch viele Möglichkeiten, es zu erweitern und zu verbessern.
Beispielsweise können weitere farbige Ziegeln mit eingebaut werden. Außerdem könnte die Figur des Roboters durch eine andere Figur ersetzt werden. Dies birgt die Möglichkeit, die Aufgabenstellung auch für Schülerinnen motivierender zu gestalten.
Des weiteren gibt es noch Konzepte von Robot Karol, die in Snap! nicht umgesetzt wurden. Hierzu zählen der Quader und der Rucksack.